Package 'whep'

July 25, 2025

Title Processing Agro-Environmental Data

Version 0.1.0

Description A set of tools for processing and analyzing data developed in the context of the ``Who Has Eaten the Planet" (WHEP) project, funded by the European Research Council (ERC). For more details on multi-regional input–output model ``Food and Agriculture Biomass Input–Output" (FABIO) see Bruckner et al. (2019) <doi:10.1021/acs.est.9b03554>.

License MIT + file LICENSE

Imports cli, dplyr, fs, FAOSTAT, httr, mipfp, nanoparquet, pins, purrr, readr, rlang, stringr, tidyr, withr, yaml

Encoding UTF-8

RoxygenNote 7.3.2

Suggests ggplot2, googlesheets4, here, knitr, pointblank, rmarkdown, testthat (>= 3.0.0), tibble

Config/testthat/edition 3

VignetteBuilder knitr

URL https://eduaguilera.github.io/whep/, https://github.com/eduaguilera/whep

BugReports https://github.com/eduaguilera/whep/issues

Depends R (>= 4.2.0)

LazyData true

NeedsCompilation no

Author Catalin Covaci [aut, cre] (ORCID: <https://orcid.org/0009-0005-2186-5972>), Eduardo Aguilera [aut, cph] (ORCID: <https://orcid.org/0000-0003-4382-124X>), João Serra [ctb] (ORCID: <https://orcid.org/0000-0002-3561-5350>), European Research Council [fnd]

Maintainer Catalin Covaci <catalin.covaci@csic.es>

Repository CRAN

Date/Publication 2025-07-25 09:50:01 UTC

Contents

`add_area_code` *Get area codes from area names*

Description

Add a new column to an existing tibble with the corresponding code for each name. The codes are assumed to be from those defined by the `FABIO` model.

Usage

```
add_area_code(table, name_column = "area_name", code_column = "area_code")
```

Arguments

`table`	The table that will be modified with a new column.
`name_column`	The name of the column in `table` containing the names.
`code_column`	The name of the output column containing the codes.

Value

A tibble with all the contents of `table` and an extra column named `code_column`, which contains the codes. If there is no code match, an `NA` is included.

Examples

```
table <- tibble::tibble(
  area_name = c("Armenia", "Afghanistan", "Dummy Country", "Albania")
)

add_area_code(table)

table |>
  dplyr::rename(my_area_name = area_name) |>
  add_area_code(name_column = "my_area_name")

add_area_code(table, code_column = "my_custom_code")
```

---

add_area_name *Get area names from area codes*

Description

Add a new column to an existing tibble with the corresponding name for each code. The codes are assumed to be from those defined by the FABIO model, which them themselves come from FAOSTAT internal codes. Equivalences with ISO 3166-1 numeric can be found in the *Area Codes* CSV from the zip file that can be downloaded from FAOSTAT. TODO: Think about this, would be nice to use ISO3 codes but won't be enough for our periods.

Usage

```
add_area_name(table, code_column = "area_code", name_column = "area_name")
```

Arguments

table	The table that will be modified with a new column.
code_column	The name of the column in table containing the codes.
name_column	The name of the output column containing the names.

Value

A tibble with all the contents of table and an extra column named name_column, which contains the names. If there is no name match, an NA is included.

Examples

```
table <- tibble::tibble(area_code = c(1, 2, 4444, 3))

add_area_name(table)

table |>
  dplyr::rename(my_area_code = area_code) |>
  add_area_name(code_column = "my_area_code")
```

```
add_area_name(table, name_column = "my_custom_name")
```

`add_item_cbs_code` *Get commodity balance sheet item codes from item names*

Description

Add a new column to an existing tibble with the corresponding code for each commodity balance sheet item name. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_cbs_code(
  table,
  name_column = "item_cbs_name",
  code_column = "item_cbs_code"
)
```

Arguments

`table`	The table that will be modified with a new column.
`name_column`	The name of the column in `table` containing the names.
`code_column`	The name of the output column containing the codes.

Value

A tibble with all the contents of `table` and an extra column named `code_column`, which contains the codes. If there is no code match, an `NA` is included.

Examples

```
table <- tibble::tibble(
  item_cbs_name = c("Cottonseed", "Eggs", "Dummy Item")
)
add_item_cbs_code(table)

table |>
  dplyr::rename(my_item_cbs_name = item_cbs_name) |>
  add_item_cbs_code(name_column = "my_item_cbs_name")

add_item_cbs_code(table, code_column = "my_custom_code")
```

`add_item_cbs_name` *Get commodity balance sheet item names from item codes*

Description

Add a new column to an existing tibble with the corresponding name for each commodity balance sheet item code. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_cbs_name(
  table,
  code_column = "item_cbs_code",
  name_column = "item_cbs_name"
)
```

Arguments

`table`	The table that will be modified with a new column.
`code_column`	The name of the column in `table` containing the codes.
`name_column`	The name of the output column containing the names.

Value

A tibble with all the contents of `table` and an extra column named `name_column`, which contains the names. If there is no name match, an `NA` is included.

Examples

```
table <- tibble::tibble(item_cbs_code = c(2559, 2744, 9876))
add_item_cbs_name(table)

table |>
  dplyr::rename(my_item_cbs_code = item_cbs_code) |>
  add_item_cbs_name(code_column = "my_item_cbs_code")

add_item_cbs_name(table, name_column = "my_custom_name")
```

`add_item_prod_code` *Get production item codes from item names*

Description

Add a new column to an existing tibble with the corresponding code for each production item name. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_prod_code(
  table,
  name_column = "item_prod_name",
  code_column = "item_prod_code"
)
```

Arguments

`table`	The table that will be modified with a new column.
`name_column`	The name of the column in `table` containing the names.
`code_column`	The name of the output column containing the codes.

Value

A tibble with all the contents of `table` and an extra column named `code_column`, which contains the codes. If there is no code match, an `NA` is included.

Examples

```
table <- tibble::tibble(
  item_prod_name = c("Rice", "Cabbages", "Dummy Item")
)
add_item_prod_code(table)

table |>
  dplyr::rename(my_item_prod_name = item_prod_name) |>
  add_item_prod_code(name_column = "my_item_prod_name")

add_item_prod_code(table, code_column = "my_custom_code")
```

`add_item_prod_name` *Get production item names from item codes*

Description

Add a new column to an existing tibble with the corresponding name for each production item code. The codes are assumed to be from those defined by FAOSTAT.

Usage

```
add_item_prod_name(
  table,
  code_column = "item_prod_code",
  name_column = "item_prod_name"
)
```

Arguments

`table`	The table that will be modified with a new column.
`code_column`	The name of the column in `table` containing the codes.
`name_column`	The name of the output column containing the names.

Value

A tibble with all the contents of `table` and an extra column named `name_column`, which contains the names. If there is no name match, an `NA` is included.

Examples

```
table <- tibble::tibble(item_prod_code = c(27, 358, 12345))
add_item_prod_name(table)

table |>
  dplyr::rename(my_item_prod_code = item_prod_code) |>
  add_item_prod_name(code_column = "my_item_prod_code")

add_item_prod_name(table, name_column = "my_custom_name")
```

`build_supply_use` *Supply and use tables*

Description

Create a table with processes, their inputs (*use*) and their outputs (*supply*).

Usage

```
build_supply_use(
  cbs_version = NULL,
  feed_intake_version = NULL,
  primary_prod_version = NULL,
  primary_residues_version = NULL,
  processing_coefs_version = NULL
)
```

Arguments

`cbs_version`	File version passed to `get_wide_cbs()` call.
`feed_intake_version`	File version passed to `get_feed_intake()` call.
`primary_prod_version`	File version passed to `get_primary_production()` call.
`primary_residues_version`	File version passed to `get_primary_residues()` call.
`processing_coefs_version`	File version passed to `get_processing_coefs()` call.

Value

A tibble with the supply and use data for processes. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `proc_group`: The type of process taking place. It can be one of:
  - `crop_production`: Production of crops and their residues, e.g. rice production, coconut production, etc.
  - `husbandry`: Animal husbandry, e.g. dairy cattle husbandry, non-dairy cattle husbandry, layers chickens farming, etc.
  - `processing`: Derived subproducts obtained from processing other items. The items used as inputs are those that have a non-zero processing use in the commodity balance sheet. See `get_wide_cbs()` for more details. In each process there is a single input. In some processes like olive oil extraction or soyabean oil extraction this might make sense. Others like alcohol production need multiple inputs (e.g. multiple crops work), so in this

data there would not be a process like alcohol production but rather a *virtual* process like 'Wheat and products processing', giving all its possible outputs. This is a constraint because of how the data was obtained and might be improved in the future. See `get_processing_coefs()` for more details.

- `proc_cbs_code`: The code of the main item in the process taking place. Together with `proc_group`, these two columns uniquely represent a process. The main item is predictable depending on the value of `proc_group`:
  - `crop_production`: The code is from the item for which seed usage (if any) is reported in the commodity balance sheet (see `get_wide_cbs()` for more). For example, the rice code for a rice production process or the cottonseed code for the cotton production one.
  - `husbandry`: The code of the farmed animal, e.g. bees for beekeeping, non-dairy cattle for non-dairy cattle husbandry, etc.
  - `processing`: The code of the item that is used as input, i.e., the one that is processed to get other derived products. This uniquely defines a process within the group because of the nature of the data that was used, which you can see in `get_processing_coefs()`.

  For code details see e.g. `add_item_cbs_name()`.
- `item_cbs_code`: The code of the item produced or used in the process. Note that this might be the same value as `proc_cbs_code`, e.g., in rice production process for the row defining the amount of rice produced or the amount of rice seed as input, but it might also have a different value, e.g. for the row defining the amount of straw residue from rice production. For code details see e.g. `add_item_cbs_name()`.
- `type`: Can have two values:
  - `use`: The given item is an input of the process.
  - `supply`: The given item is an output of the process.
- `value`: Quantity in tonnes.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default versions (i.e. no arguments).
build_supply_use(
  cbs_version = "20250721T132006Z-8ea47",
  feed_intake_version = "20250721T143825Z-c1313",
  primary_prod_version = "20250721T145805Z-8e12a",
  primary_residues_version = "20250721T150132Z-dfd94",
  processing_coefs_version = "20250721T143403Z-216d7"
)
```

`expand_trade_sources` *Trade data sources*

Description

Create a new dataframe where each row has a year range into one where each row is a single year, effectively 'expanding' the whole year range.

Usage

```
expand_trade_sources(trade_sources)
```

Arguments

trade_sources	A tibble dataframe where each row contains the year range.

Value

A tibble dataframe where each row corresponds to a single year for a given source.

Examples

```
trade_sources <- tibble::tibble(
  Name = c("a", "b", "c"),
  Trade = c("t1", "t2", "t3"),
  Info_Format = c("year", "partial_series", "year"),
  Timeline_Start = c(1, 1, 2),
  Timeline_End = c(3, 4, 5),
  Timeline_Freq = c(1, 1, 2),
  `Imp/Exp` = "Imp",
  SACO_link = NA,
)
expand_trade_sources(trade_sources)
```

---

get_bilateral_trade *Bilateral trade data*

Description

Reports trade between pairs of countries in given years.

Usage

```
get_bilateral_trade(trade_version = NULL, cbs_version = NULL)
```

Arguments

trade_version	File version used for bilateral trade input. See whep_inputs for version details.
cbs_version	File version passed to `get_wide_cbs()` call.

Value

A tibble with the reported trade between countries. For efficient memory usage, the tibble is not exactly in tidy format. It contains the following columns:

- `year`: The year in which the recorded event occurred.

- `item_cbs_code`: FAOSTAT internal code for the item that is being traded. For code details see e.g. `add_item_cbs_name()`.
- `bilateral_trade`: Square matrix of `NxN` dimensions where `N` is the total number of countries being considered. The matrix row and column names are exactly equal and they represent country codes.
  - Row name: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
  - Column name: FAOSTAT internal code for the country that is importing the item. See row name explanation above.

  If `m` is the matrix, the value at `m["A", "B"]` is the trade in tonnes from country `"A"` to country `"B"`, for the corresponding year and item. The matrix can be considered *balanced*. This means:
  - The sum of all values from row `"A"`, where `"A"` is any country, should match the total exports from country `"A"` reported in the commodity balance sheet (which is considered more accurate for totals).
  - The sum of all values from column `"A"`, where `"A"` is any country, should match the total imports into country `"A"` reported in the commodity balance sheet (which is considered more accurate for totals).

  The sums may not be exactly the expected values because of precision issues and/or the iterative proportional fitting algorithm not converging fast enough, but should be relatively very close to the desired totals.

The step by step approach to obtain this data tries to follow the FABIO model and is explained below. All the steps are performed separately for each group of year and item.

- From the FAOSTAT reported bilateral trade, there are sometimes two values for one trade flow: the exported amount claimed by the reporter country and the import amount claimed by the partner country. Here, the export data was preferred, i.e., if country `"A"` says it exported `X` tonnes to country `"B"` but country `"B"` claims they got `Y` tonnes from country `"A"`, we trust the export data `X`. This choice is only needed if there exists a reported amount from both sides. Otherwise, the single existing report is chosen.
- Complete the country data, that is, add any missing combinations of country trade with NAs, which will be estimated later. In the matrix form, this doesn't increase the memory usage since we had to build a matrix anyway (for the balancing algorithm), and the *empty* parts also take up memory. This is also done for total imports/exports from the commodity balance sheet, but these are directly filled with 0s instead.
- The total imports and exports from the commodity balance sheet are balanced by downscaling the largest of the two to match the lowest. This is done in the following way:
  - If `total_imports > total_exports`: Set `import` as `total_exports * import / total_import`.
  - If `total_exports > total_exports`: Set `export` as `total_exports * export / total_export`.
- The missing data in the matrix must be estimated. It's done like this:
  - For each pair of exporter `i` and importer `j`, we estimate a bilateral trade `m[i, j]` using the export shares of `i` and import shares of `j` from the commodity balance sheet:
    * `est_1 <- exports[i] * imports[j] / sum(imports)`, i.e., total exports of country `i` spread among other countries' import shares.

    * `est_2 <- imports[j] * exports[i] / sum(exports)`, i.e. total imports of country `j` spread among other countries' export shares.
    * `est <- (est_1 + est_2) / 2`, i.e., the mean of both estimates.

    In the above computations, exports and imports are the original values before they were balanced.
  - The estimates for data that already existed (i.e. non-NA) are discarded. For the ones left, for each row (i.e. exporter country), we get the difference between its balanced total export and the sum of original non-estimated data. The result is the `gap` we can actually fill with estimates, so as to not get past the reported total export. If the sum of non-discarded estimates is larger, it must be downscaled and spread by computing `gap * non_discarded_estimate / sum(non_discarded_estimates)`.
  - The estimates are divided by a *trust factor*, in the sense that we don't rely on the whole value, thinking that a non-present value might actually be because that specific trade was 0, so we don't overestimate too much. The chosen factor is 10%, so only 10% of the estimate's value is actually used to fill the NA from the original bilateral trade matrix.
- The matrix is balanced, as mentioned before, using the iterative proportional fitting algorithm. The target sums for rows and columns are respectively the balanced exports and imports computed from the commodity balance sheet. The algorithm is performed directly using the mipfp R package.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default versions (i.e. no arguments).
get_bilateral_trade(
  trade_version = "20250721T141553Z-5707e",
  cbs_version = "20250721T132006Z-8ea47"
)
```

`get_faostat_data` *Scrapes activity_data from FAOSTAT and slightly post-processes it*

Description

Important: Dynamically allows for the introduction of subsets as `"..."`.

Note: overhead by individually scraping FAOSTAT code QCL for crop data; it's fine.

Usage

```
get_faostat_data(activity_data, ...)
```

Arguments

`activity_data`	activity data required from FAOSTAT; needs to be one of `c('livestock','crop_area','crop_yield',`
`...`	can be whichever column name from `get_faostat_bulk`, particularly `year`, `area` or `ISO3_CODE`.

Value

data.frame of FAOSTAT for activity_data; default is for all years and countries.

Examples

```
get_faostat_data("livestock", year = 2010, area = "Portugal")
```

---

get_feed_intake *Livestock feed intake*

Description

Get amount of items used for feeding livestock.

Usage

```
get_feed_intake(version = NULL)
```

Arguments

version	File version to use as input. See whep_inputs for details.

Value

A tibble with the feed intake data. It contains the following columns:

- year: The year in which the recorded event occurred.
- area_code: The code of the country where the data is from. For code details see e.g. add_area_name().
- live_anim_code: Commodity balance sheet code for the type of livestock that is fed. For code details see e.g. add_item_cbs_name().
- item_cbs_code: The code of the item that is used for feeding the animal. For code details see e.g. add_item_cbs_name().
- feed_type: The type of item that is being fed. It can be one of:
  - animals: Livestock product, e.g. Bovine Meat, Butter, Ghee, etc.
  - crops: Crop product, e.g. Vegetables, Other, Oats, etc.
  - residues: Crop residue, e.g. Straw, Fodder legumes, etc.
  - grass: Grass, e.g. Grassland, Temporary grassland, etc.
  - scavenging: Other residues. Single Scavenging item.
- supply: The computed amount in tonnes of this item that should be fed to this animal, when sharing the total item feed use from the Commodity Balance Sheet among all livestock.
- intake: The actual amount in tonnes that the animal needs, which can be less than the theoretical used amount from supply.

- intake_dry_matter: The amount specified by intake but only considering dry matter, so it should be less than intake.
- loss: The amount that is not used for feed. This is supply - intake.
- loss_share: The percent that is lost. This is loss / supply.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default version (i.e. no arguments).
get_feed_intake(version = "20250721T143825Z-c1313")
```

---

get_primary_production

Primary items production

Description

Get amount of crops, livestock and livestock products.

Usage

```
get_primary_production(version = NULL)
```

Arguments

version	File version to use as input. See whep_inputs for details.

Value

A tibble with the item production data. It contains the following columns:

- year: The year in which the recorded event occurred.
- area_code: The code of the country where the data is from. For code details see e.g. add_area_name().
- item_prod_code: FAOSTAT internal code for each produced item.
- item_cbs_code: FAOSTAT internal code for each commodity balance sheet item. The commodity balance sheet contains an aggregated version of production items. This field is the code for the corresponding aggregated item.
- live_anim_code: Commodity balance sheet code for the type of livestock that produces the livestock product. It can be:
  - NA: The entry is not a livestock product.
  - Non-NA: The code for the livestock type. The name can also be retrieved by using add_item_cbs_name().
- unit: Measurement unit for the data. Here, keep in mind three groups of items: crops (e.g. Apples and products, Beans...), livestock (e.g. Cattle, dairy, Goats...) and livestock products (e.g. Poultry Meat, Offals, Edible...). Then the unit can be one of:

  - tonnes: Available for crops and livestock products.
  - ha: Hectares, available for crops.
  - t_ha: Tonnes per hectare, available for crops.
  - heads: Number of animals, available for livestock.
  - LU: Standard Livestock Unit measure, available for livestock.
  - t_head: tonnes per head, available for livestock products.
  - t_LU: tonnes per Livestock Unit, available for livestock products.
- value: The amount of item produced, measured in unit.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default version (i.e. no arguments).
get_primary_production(version = "20250721T145805Z-8e12a")
```

---

get_primary_residues *Crop residue items*

Description

Get type and amount of residue produced for each crop production item.

Usage

```
get_primary_residues(version = NULL)
```

Arguments

version	File version to use as input. See whep_inputs for details.

Value

A tibble with the crop residue data. It contains the following columns:

- year: The year in which the recorded event occurred.
- area_code: The code of the country where the data is from. For code details see e.g. add_area_name().
- item_cbs_code_crop: FAOSTAT internal code for each commodity balance sheet item. This is the crop that is generating the residue.
- item_cbs_code_residue: FAOSTAT internal code for each commodity balance sheet item. This is the obtained residue. In the commodity balance sheet, this can be three different items right now:
  - 2105: Straw
  - 2106: Other crop residues
  - 2107: Firewood

  These are actually not FAOSTAT defined items, but custom defined by us. When necessary, FAOSTAT codes are extended for our needs.
- value: The amount of residue produced, measured in tonnes.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default version (i.e. no arguments).
get_primary_residues(version = "20250721T150132Z-dfd94")
```

`get_processing_coefs` *Processed products share factors*

Description

Reports quantities of commodity balance sheet items used for `processing` and quantities of their corresponding processed output items.

Usage

```
get_processing_coefs(version = NULL)
```

Arguments

`version`	File version to use as input. See whep_inputs for details.

Value

A tibble with the quantities for each processed product. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `item_cbs_code_to_process`: FAOSTAT internal code for each one of the items that are being processed and will give other subproduct items. For code details see e.g. `add_item_cbs_name()`.
- `value_to_process`: tonnes of this item that are being processed. It matches the amount found in the `processing` column from the data obtained by `get_wide_cbs()`.
- `item_cbs_code_processed`: FAOSTAT internal code for each one of the subproduct items that are obtained when processing. For code details see e.g. `add_item_cbs_name()`.
- `initial_conversion_factor`: estimate for the number of tonnes of `item_cbs_code_processed` obtained for each tonne of `item_cbs_code_to_process`. It will be used to compute the `final_conversion_factor`, which leaves everything balanced. TODO: explain how it's computed.
- `initial_value_processed`: first estimate for the number of tonnes of `item_cbs_code_processed` obtained from `item_cbs_code_to_process`. It is computed as `value_to_process * initial_conversion_factor`.
- `conversion_factor_scaling`: computed scaling needed to adapt `initial_conversion_factor` so as to get a final balanced total of subproduct quantities. TODO: explain how it's computed.
- `final_conversion_factor`: final used estimate for the number of tonnes of `item_cbs_code_processed` obtained for each tonne of `item_cbs_code_to_process`. It is computed as `initial_conversion_factor * conversion_factor_scaling`.

- `final_value_processed`: final estimate for the number of tonnes of `item_cbs_code_processed` obtained from `item_cbs_code_to_process`. It is computed as `initial_value_processed * final_conversion_factor`.

For the final data obtained, the quantities `final_value_processed` are balanced in the following sense: the total sum of `final_value_processed` for each unique tuple of `(year, area_code, item_cbs_code_processed)` should be exactly the quantity reported for that year, country and `item_cbs_code_processed` item in the `production` column obtained from `get_wide_cbs()`. This is because they are not primary products, so the amount from 'production' is actually the amount of subproduct obtained. TODO: Fix few data where this doesn't hold.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default version (i.e. no arguments).
get_processing_coefs(version = "20250721T143403Z-216d7")
```

`get_wide_cbs` *Commodity balance sheet data*

Description

States supply and use parts for each commodity balance sheet (CBS) item.

Usage

```
get_wide_cbs(version = NULL)
```

Arguments

`version`	File version to use as input. See whep_inputs for details.

Value

A tibble with the commodity balance sheet data in wide format. It contains the following columns:

- `year`: The year in which the recorded event occurred.
- `area_code`: The code of the country where the data is from. For code details see e.g. `add_area_name()`.
- `item_cbs_code`: FAOSTAT internal code for each item. For code details see e.g. `add_item_cbs_name()`.

The other columns are quantities (measured in tonnes), where total supply and total use should be balanced.

For supply:

- `production`: Produced locally.
- `import`: Obtained from importing from other countries.

- `stock_retrieval`: Available as net stock from previous years. For ease, only one stock column is included here as supply. If the value is positive, there is a stock quantity available as supply. Otherwise, it means a larger quantity was stored for later years and cannot be used as supply, having to deduce it from total supply. Since in this case it is negative, the total supply is still computed as the sum of all of these.

For use:

- `food`: Food for humans.
- `feed`: Food for animals.
- `export`: Released as export for other countries.
- `seed`: Intended for new production.
- `processing`: The product will be used to obtain other subproducts.
- `other_uses`: Any other use not included in the above ones.

There is an additional column `domestic_supply` which is computed as the total use excluding `export`.

Examples

```
# Note: These are smaller samples to show outputs, not the real data.
# For all data, call the function with default version (i.e. no arguments).
get_wide_cbs(version = "20250721T132006Z-8ea47")
```

`items_cbs` *Commodity balance sheet items*

Description

Defines name/code correspondences for commodity balance sheet (CBS) items.

Usage

```
items_cbs
```

Format

A tibble where each row corresponds to one CBS item. It contains the following columns:

- `item_cbs_code`: A numeric code used to refer to the CBS item.
- `item_cbs_name`: A natural language name for the item.
- `item_type`: An ad-hoc grouping of items. This is a work in progress evolving depending on our needs, so for now it only has two possible values:
  - `livestock`: The CBS item represents a live animal.
  - `other`: Not any of the previous groups.

Source

Inspired by FAOSTAT data.

items_prod *Primary production items*

Description

Defines name/code correspondences for production items.

Usage

```
items_prod
```

Format

A tibble where each row corresponds to one production item. It contains the following columns:

- `item_prod_code`: A numeric code used to refer to the item.
- `item_prod_name`: A natural language name for the item.
- `item_type`: An ad-hoc grouping of items. This is a work in progress evolving depending on our needs, so for now it only has two possible values:
  - `crop_product`: The CBS item represents a crop product.
  - `other`: Not any of the previous groups.

Source

Inspired by FAOSTAT data.

polities *Polities*

Description

Defines name/code correspondences for polities (political entities).

Usage

```
polities
```

Format

A tibble where each row corresponds to one polity. It contains the following columns: TODO: On polities Pull Request, coming soon

whep_inputs *External inputs*

Description

The information needed for accessing external datasets used as inputs in our modeling.

Usage

```
whep_inputs
```

Format

A tibble where each row corresponds to one external input dataset. It contains the following columns:

- `alias`: An internal name used to refer to this dataset, which is the expected name when trying to get the dataset with `whep_read_file()`.
- `board_url`: The public static URL where the data is found, following the concept of a *board* from the `pins` package, which is what we use for storing these input datasets.
- `version`: The specific version of the dataset, as defined by the `pins` package. The version is a string similar to `"20250714T123343Z-114b5"`. This version is the one used by default if no `version` is specified when calling `whep_read_file()`. If you want to use a different one, you can find the available versions of a file by using `whep_list_file_versions()`.

Source

Created by the package authors.

whep_list_file_versions *Input file versions*

Description

Lists all existing versions of an input file from `whep_inputs`.

Usage

```
whep_list_file_versions(file_alias)
```

Arguments

`file_alias`	Internal name of the requested file. You can find the possible values in the `whep_inputs` dataset.

Value

A tibble where each row is a version. For details about its format, see `pins::pin_versions()`.

Examples

```
whep_list_file_versions("read_example")
```

`whep_read_file` *Download, cache and read files*

Description

Used to fetch input files that are needed for the package's functions and that were built in external sources and are too large to include directly. This is a public function for transparency purposes, so that users can inspect the original inputs of this package that were not directly processed here.

If the requested file doesn't exist locally, it is downloaded from a public link and cached before reading it. This is all implemented using the `pins` package. It supports multiple file formats and file versioning.

Usage

```
whep_read_file(file_alias, type = "parquet", version = NULL)
```

Arguments

`file_alias`	Internal name of the requested file. You can find the possible values in the `alias` column of the `whep_inputs` dataset.
`type`	The extension of the file that must be read. Possible values: • `parquet`: This is the default value for code efficiency reasons. • `csv`: Mainly available for those who want a more human-readable option. If the `parquet` version is available, this is useless because this function already returns the dataset in an `R` object, so the origin is irrelevant, and `parquet` is read faster. Saving each file in both formats is for transparency and accessibility purposes, e.g., having to share the data with non-programmers who can easily import a CSV into a spreadsheet. You will most likely never have to set this option manually unless for some reason a file could not be supplied in e.g. `parquet` format but was in another one.
`version`	The version of the file that must be read. Possible values: • `NULL`: This is the default value. A frozen version is chosen to make the code reproducible. Each release will have its own frozen versions. The version is the string that can be found in `whep_inputs` in the `version` column. • `"latest"`: This overrides the frozen version and instead fetches the latest one that is available. This might or might not match the frozen version. • Other: A specific version can also be used. For more details read the `version` column information from `whep_inputs`.

Value

A tibble with the dataset. Some information about each dataset can be found in the code where it's used as input for further processing.

Examples

```
whep_read_file("read_example")
whep_read_file("read_example", type = "parquet", version = "latest")
whep_read_file(
  "read_example",
  type = "csv",
  version = "20250721T152646Z-ce61b"
)
```

Index