Package: whep (via r-universe)

October 22, 2026

Title Processing Agro-Environmental Data

Version 0.3.0.9000

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 arrow, cli, data.table, deSolve, dplyr, fs, httr, Matrix, methods, nanoparquet (>= 0.5.0), pins, purrr (>= 1.0.0), rappdirs, readr, rlang, scales, stringr, tibble, tidyr, vctrs, withr, yaml, stringi, zoo

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests archive, callr, codetools, DBI, DiagrammeR, ggpattern, ggplot2, here, htmltools, jsonlite, knitr, ncdf4, patchwork, pointblank, readxl, rmarkdown, RSQLite, s2, sf, spelling, terra, testthat (>= 3.0.0), zip

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 (>= 4.1.0)

LazyData true

LazyDataCompression xz

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

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

Date/Publication 2026-10-22 10:13:32 UTC

RemoteUrl https://github.com/eduaguilera/whep

RemoteRef HEAD

RemoteSha 3205ff19c926fc2cb9a18e60cab92af8470986cb

Contents

add_area_code . . . 11
add_area_name . . . 12
add_footprint_product_stage . . . 13
add_item_cbs_code . . . 14
add_item_cbs_name . . . 15
add_item_prod_code . . . 16
add_item_prod_name . . . 17
add_manure_bedding . . . 18
add_polity_code . . . 19
admin_coverage_prototype . . . 22
admin_items_eurostat . . . 22
admin_items_jrc . . . 23
admin_items_nass . . . 24
admin_items_sidra . . . 25
admin_shares_prototype . . . 25
admin_shares_schema . . . 26
admin_source_vocabularies . . . 30
admin_species_eurostat . . . 32
admin_species_nass . . . 33
admin_species_sidra . . . 34
aggregate_grass_to_polity . . . 34
aggregate_unit_extent . . . 35
align_extension . . . 38
allocate_grazing_to_products . . . 39
allocate_level_crops . . . 40
allocate_manure_to_land . . . 43
allocate_manure_transport . . . 45
amg_h_by_input_type . . . 46
animals_codes . . . 47
apply_management_losses . . . 48
assert_footprint_invariants . . . 50
assert_parquet_integrity . . . 51
assert_table_schema . . . 52
attach_provenance . . . 52
attach_scope . . . 53
attribute_fallow_to_crops . . . 54
backcast_admin_shares . . . 55
balance_io_flows . . . 58
balance_ras . . . 59

biomass_coefs . . . 60
build_ag_land_support . . . 67
build_allocation_layer . . . 69
build_carbon_balance . . . 72
build_carbon_inputs . . . 80
build_cbs_prices . . . 86
build_cell_climate_zone . . . 87
build_cell_moisture_regime . . . 88
build_cell_polity . . . 93
build_commodity_balances . . . 96
build_constant_territory_series . . . 107
build_critical_n_binding . . . 110
build_crop_land_extension . . . 112
build_crop_soil_n2o_extension . . . 114
build_crop_water_consumption . . . 117
build_crop_water_use . . . 119
build_cropgrids_land_extension . . . 121
build_detailed_trade . . . 123
build_energy_co2_extension . . . 128
build_fao_arable_fallow_extension . . . 130
build_feed_demand . . . 136
build_feed_intake_local . . . 138
build_food_supply . . . 140
build_footprint . . . 144
build_grass_availability . . . 145
build_grass_availability_lpjml . . . 146
build_grass_natural_carbon_inputs . . . 147
build_grassland_intensity_classes . . . 153
build_grassland_land_extension . . . 156
build_grazing_feed_footprint . . . 158
build_gridded_landuse . . . 159
build_gridded_livestock . . . 164
build_gridded_livestock_emissions . . . 170
build_hayr_land_extension . . . 177
build_historical_land_areas . . . 179
build_human_n . . . 180
build_intake_dispersion . . . 185
build_io_model . . . 187
build_land_balance_footprint . . . 189
build_level_crop_targets . . . 190
build_livestock_ghg_extension . . . 193
build_livestock_nutrient_flows . . . 199
build_loss_wedge . . . 201
build_n_boundary_country . . . 204
build_n_boundary_exceedance . . . 208
build_n_boundary_percapita . . . 213
build_n_deposition . . . 215
build_n_exceedance_extension . . . 218

build_n_inputs . . . 219
build_n_loss_drivers . . . 226
build_n_pathway_exceedance . . . 229
build_n_percapita . . . 231
build_nitrogen_balance . . . 233
build_nourishment_band . . . 239
build_polycell_land_uses . . . 242
build_polycell_support . . . 246
build_primary_prices . . . 250
build_primary_production . . . 251
build_processing_coefs . . . 254
build_protein_quality . . . 255
build_protein_requirement . . . 257
build_protein_score . . . 259
build_regime_yield_ratio . . . 262
build_residue_bedding_supply . . . 267
build_sjos_n_footprint . . . 268
build_sjos_nitrogen . . . 270
build_soil_carbon_inputs . . . 273
build_supply_use . . . 277
build_total_population_grid . . . 279
build_trade_prices . . . 281
build_water_balance . . . 282
calculate_bnf . . . 286
calculate_cohorts_systems . . . 287
calculate_critical_n . . . 289
calculate_crop_bnf . . . 291
calculate_crop_npp . . . 291
calculate_crop_npp_components . . . 292
calculate_crop_residues . . . 293
calculate_crop_roots . . . 294
calculate_enteric_ch4 . . . 295
calculate_food_gini . . . 300
calculate_indirect_n2o_nh3 . . . 301
calculate_livestock_emissions . . . 302
calculate_lmdi . . . 306
calculate_manner_nh3 . . . 313
calculate_manner_nh3_default . . . 315
calculate_manure_emissions . . . 316
calculate_n_leaching . . . 321
calculate_n_surplus . . . 323
calculate_nh3 . . . 325
calculate_nonsymbiotic_bnf . . . 326
calculate_npp_carbon_nitrogen . . . 327
calculate_nue_crops . . . 328
calculate_nue_livestock . . . 329
calculate_potential_npp . . . 330
calculate_residue_destinies . . . 331

calculate_soc_amg . . . 336
calculate_soc_century . . . 337
calculate_soc_dynamics . . . 338
calculate_soc_hsoc . . . 339
calculate_soc_icbm . . . 340
calculate_soc_lpjml . . . 341
calculate_soc_rothc . . . 343
calculate_soil_n2o . . . 344
calculate_system_nue . . . 346
calculate_uncertainty_bounds . . . 347
calculate_weed_bnf . . . 348
cb_processing . . . 348
cbs_trade_codes . . . 349
cft_mapping . . . 350
check_arable_composition . . . 351
check_extent_jumps . . . 353
check_fodder_land_share . . . 354
check_footprint_conservation . . . 355
check_inputs_supplied . . . 357
check_keys_complete . . . 358
check_labels_supplied . . . 360
check_parquet_integrity . . . 361
check_series_jumps . . . 362
check_supply_use_balance . . . 364
check_table_schema . . . 365
classify_sjos_n . . . 368
climate_mcf . . . 369
climate_mcf_ipcc . . . 370
coello_synthetic_n . . . 372
combine_cov . . . 373
combine_row_evidence . . . 374
compare_footprint_methods . . . 375
compute_footprint . . . 376
compute_footprint_balance . . . 379
compute_footprint_paths . . . 380
compute_fp_product_paths . . . 382
compute_leontief_inverse . . . 384
compute_trade_flows_raw . . . 385
consolidate_sources . . . 386
conv_bouwman . . . 390
conv_krausmann . . . 391
correct_n_deposition . . . 392
create_alfredos_typologies . . . 394
create_finn_indicator . . . 395
create_grafs_plot_df . . . 396
create_n_nat_destiny . . . 397
create_n_production . . . 398
create_n_prov_destiny . . . 398

create_n_soil_inputs . . . 400
create_typologies_grafs_spain . . . 401
create_typologies_of_josette . . . 402
create_typologies_whep . . . 403
crops_eurostat . . . 404
crops_manure_n . . . 405
decompose_crop_livestock_conn . . . 405
decompose_cropland_surplus . . . 406
decompose_destiny_mix . . . 407
decompose_manure_losses . . . 408
decompose_semi_natural_surplus . . . 410
decompose_specialization_cov . . . 411
decompose_terr_losses . . . 412
decompose_terr_losses_periods . . . 413
decompose_urban_losses . . . 414
decompose_weighted_ratio . . . 415
disaggregate_ussr . . . 417
drainage_ranges . . . 419
empty_table_from_schema . . . 419
ensure_admin_shares . . . 420
ensure_columns . . . 422
estimate_energy_demand . . . 423
estimate_n_excretion . . . 424
europe_fao_spain_trade . . . 427
evidence_conflicts . . . 428
evidence_for . . . 429
expand_polycell_years . . . 430
expand_trade_sources . . . 431
feed_characteristics . . . 431
feed_taxonomy . . . 432
fertiliser_n2o_modifiers . . . 433
fill_linear . . . 434
fill_proxy_growth . . . 435
fill_sum . . . 437
folded_reporting_areas . . . 439
footprint_scope . . . 441
footprint_sensitivity . . . 442
get_arable_permanent_land . . . 443
get_bilateral_trade . . . 445
get_crop_land_extension . . . 452
get_feed_intake . . . 454
get_polity_geometries . . . 456
get_primary_production . . . 457
get_primary_residues . . . 459
get_processing_coefs . . . 465
get_provenance . . . 467
get_scope . . . 468
get_soc_climate_drivers . . . 468

get_wide_cbs . . . 471
gleam_animal_weights . . . 473
gleam_crop_residue_nitrogen . . . 474
gleam_crop_residue_params . . . 475
gleam_dressing_percentages . . . 476
gleam_energy_use_ef . . . 477
gleam_enteric_params . . . 478
gleam_feed_categories . . . 479
gleam_feed_composition . . . 479
gleam_feed_conversion_ratios . . . 480
gleam_feed_digestibility . . . 481
gleam_field_operation_ef . . . 482
gleam_fracremove . . . 482
gleam_geographic_hierarchy . . . 483
gleam_livestock_categories . . . 484
gleam_mechanization_levels . . . 485
gleam_milk_production . . . 486
gleam_mms_shares . . . 487
gleam_processing_transport_ef . . . 487
glwd_water_fraction . . . 488
grass_access_shares . . . 489
grazing_energy_coefs . . . 490
gridded_fallow_weights . . . 490
harmonize_interpolate . . . 491
harmonize_simple . . . 493
human_kgn_cap_reference . . . 495
human_kgn_cap_total_reference . . . 496
human_n_reference . . . 497
hwsd_texture_usda . . . 498
indirect_n2o_ef . . . 498
interp_vec . . . 500
ipcc_2006_enteric_ef . . . 501
ipcc_2006_manure_ef . . . 502
ipcc_2006_mcf_temp . . . 503
ipcc_2019_bo . . . 504
ipcc_2019_cfi . . . 504
ipcc_2019_enteric_ef_cattle . . . 505
ipcc_2019_enteric_ef_other . . . 506
ipcc_2019_manure_ch4_ef_cattle . . . 507
ipcc_2019_manure_ch4_ef_other . . . 507
ipcc_2019_mcf_manure . . . 508
ipcc_2019_n_excretion . . . 509
ipcc_2019_n2o_ef_direct . . . 510
ipcc_2019_soil_n_ef . . . 511
ipcc_2019_ym . . . 512
ipcc_enteric_ef_tier1 . . . 513
ipcc_manure_ef3 . . . 514
ipcc_tier2_bo_values . . . 516

ipcc_tier2_energy_coefs . . . 516
ipcc_tier2_manure_ash . . . 517
ipcc_tier2_n_retention . . . 518
ipcc_tier2_ym_values . . . 518
items_cbs . . . 519
items_full . . . 519
items_prim . . . 521
items_prod . . . 522
items_prod_full . . . 523
key_lattice_gaps . . . 524
lassaletta_grassland_share . . . 526
liv_lu_coefs . . . 528
livestock_constants . . . 528
livestock_emissions_to_kt . . . 529
livestock_production_defaults . . . 530
make_lpjml_covariate . . . 531
manner_default_technique_mix . . . 532
manner_incorporation_factor . . . 533
manner_params . . . 534
manner_rain_factor . . . 535
manner_rate_factor . . . 535
manure_inorganic_n . . . 536
max_intake_share . . . 537
meisinger_denitrification . . . 538
melt_bilateral_trade . . . 539
mueller_synthetic_n . . . 539
n_attenuation_constants . . . 540
n_boundary_params . . . 541
n2o_efs_disaggregated . . . 542
n2o_efs_ipcc2006 . . . 543
normalize_nourishment . . . 544
nourish_levels . . . 545
nourishment_thresholds . . . 546
plot_compart_factor . . . 547
plot_compart_factor_periods . . . 548
plot_compart_factor_roll . . . 549
plot_compart_factor_roll_panel . . . 550
plot_compart_factor_yearly . . . 552
plot_crop_livestock_conn . . . 553
plot_destiny_mix . . . 553
plot_finn_circularity . . . 554
plot_footprint_sankey . . . 555
plot_input_output . . . 558
plot_input_output_livestock . . . 558
plot_input_output_system . . . 559
plot_loss_decomp . . . 560
plot_loss_decomp_periods . . . 561
plot_loss_decomp_periods_panel . . . 562

plot_loss_decomp_rolling . . . 563
plot_loss_decomp_rolling_panel . . . 564
plot_loss_decomp_yearly . . . 565
plot_national_trade_flows_raw . . . 566
plot_national_trade_validation . . . 566
plot_specialization_cov . . . 567
plot_typology_indicators_panel . . . 568
plot_typology_periods_panel . . . 570
polities . . . 571
polities_cats . . . 573
polity_anchor_drift . . . 575
polity_area_crosswalk . . . 578
polity_bucket_coverage . . . 582
polity_containment . . . 583
polity_coverage_gaps . . . 584
polity_identity_conventions . . . 586
polity_label_aliases . . . 588
polity_label_item_corrections . . . 589
polity_mapping_provenance . . . 590
polycell_example_geometries . . . 593
population_source_reach . . . 593
prepare_livestock_emissions . . . 595
primary_double . . . 596
propagate_fp_uncertainty . . . 597
read_admin_family . . . 598
read_admin_shares . . . 599
read_admin_stats_eurostat . . . 600
read_admin_stats_nass . . . 603
read_admin_stats_sidra . . . 606
read_critical_n . . . 608
read_cru_climate . . . 610
read_emep_deposition . . . 611
read_fbs_population . . . 612
read_glaciated_areas . . . 614
read_glw_density . . . 615
read_glwd_water . . . 618
read_habitual_cv . . . 619
read_hwsd_topsoil_soc . . . 619
read_hyde_population . . . 621
read_level_country_grid . . . 622
read_lpjml_crop_cover . . . 626
read_lpjml_grass_productivity . . . 628
read_lpjml_hydrology . . . 629
read_lpjml_litterfall . . . 632
read_lpjml_natural_cover . . . 633
read_lpjml_npp . . . 634
read_lpjml_regime_yield . . . 635
read_lpjml_wind . . . 638

read_luh2_landuse . . . 639
read_luh2_terrestrial . . . 641
read_n_deposition . . . 642
read_polycell_support . . . 644
read_population . . . 645
read_soil_hydraulic . . . 650
read_soil_ph . . . 651
read_spam_yields . . . 652
read_wpp_population . . . 653
reconcile_admin_allocation . . . 654
record_provenance . . . 658
redistribute_feed . . . 659
regime_yield_crop_mapping . . . 661
regional_mms_distribution . . . 662
regions_full . . . 664
residue_humification . . . 667
resolve_admin_shares . . . 668
resolve_admin_units . . . 671
resolve_polity_label . . . 674
resolve_polity_lineage . . . 677
row_evidence . . . 680
row_evidence_schema . . . 682
row_promotion_status . . . 683
run_spatialize . . . 684
run_typology_sensitivity . . . 690
seam_gate . . . 691
seam_gate_tolerances . . . 696
sjos_levels . . . 698
smil_2001_synthetic_n_global . . . 699
soc_crop_group . . . 700
soc_rate_modifier_amg . . . 701
soc_rate_modifier_century . . . 702
soc_rate_modifier_icbm . . . 703
soc_rate_modifier_lpjml . . . 704
soc_rate_modifier_rothc . . . 705
soc_soil_cover_curve . . . 706
soc_turnover_params . . . 707
soil_cn_ratios . . . 708
soil_hydraulic_by_texture . . . 710
som_ranges . . . 711
spatialize_country_n_to_crops . . . 712
split_manure_management . . . 714
split_regime_yield . . . 715
stamp_inputs_supplied . . . 717
subsoil_no3_reduction . . . 718
summarise_conservation . . . 718
summarize_bnf . . . 719
temperature_adjustment . . . 720

uncertainty_ranges 721
unit_cropland_extent 721
validate_national_trade 722
validate_national_trade_raw 723
whep_clear_cache 724
whep_coef_table 725
whep_inputs 725
whep_list_file_versions 728
whep_read_file 729
whep_registry 733
write_parquet_checked 734
write_table_checked 735

Index **738**

`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_footprint_product_stage` *Add a final-demand product-area stage to footprints.*

Description

Split footprint rows by the area that supplied the final-demand product, using shares from `y_mat`. This preserves the standard footprint totals while adding a FABIO-viewer style phase: origin product -> product/supplier area -> product -> final-demand area.

This is a compact global-view helper. It does not recompute the full origin-sector by product-sector Leontief cube. Instead, each existing footprint row is allocated over the product-area shares observed in final demand for the same `target_area`, `target_fd`, and `target_item`.

Usage

```
add_footprint_product_stage(
  footprints,
  y_mat,
  labels,
  fd_labels,
  max_product_areas = 5,
  other_area_name = "Other",
  min_share = 0
)
```

Arguments

`footprints`	Footprint table from `compute_footprint()` with `target_area`, `target_item`, `target_fd`, and `value`.
`y_mat`	Final demand matrix from `build_io_model()`.
`labels`	Tibble mapping Y rows to `area_code` and `item_cbs_code`.
`fd_labels`	Tibble mapping Y columns to `area_code` and `fd_col`.
`max_product_areas`	Maximum number of supplier/product areas to keep separately for each final-demand area, item, and demand category. Smaller supplier areas are grouped into `other_area_name`.
`other_area_name`	Label for grouped supplier/product areas.
`min_share`	Drop split paths smaller than this percentage of the total input footprint value. Use 0 to keep all split paths.

Value

`footprints` with `product_area`, `product_area_name`, `product_item`, and `product_share` columns. `value` is replaced by the split path value.

Examples

```
# One footprint row of 100, split over the two areas that supply item 20 to
# the final demand of area 1 in the shares observed in `y_mat` (80 / 20).
footprints <- tibble::tibble(
  origin_area = 1L,
  origin_item = 10L,
  target_area = 1L,
  target_area_name = "Target",
  target_item = 20L,
  target_fd = "food",
  value = 100
)

add_footprint_product_stage(
  footprints = footprints,
  y_mat = Matrix::Matrix(c(80, 20), nrow = 2, sparse = TRUE),
  labels = tibble::tibble(
    area_code = c(1L, 2L),
    item_cbs_code = c(20L, 20L)
  ),
  fd_labels = tibble::tibble(area_code = 1L, fd_col = "food")
) |>
  dplyr::select(origin_item, product_area, product_share, value)
```

---

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")
```

`add_manure_bedding` *Add bedding carbon and nitrogen to the housed manure streams.*

Description

Places the bedding supply from `build_residue_bedding_supply()` onto the litter-using, non-grazing streams of a `split_manure_management()` result, in proportion to the nitrogen each of those streams already carries. Grazing deposition (`Pasture/Range/Paddock`) never receives bedding, which is what keeps its manure the fresh excreta it is.

The result is the input rows plus `dm_bedding`, `n_bedding` and `c_bedding`, which `apply_management_losses()` then adds to the manure applied to land.

Usage

```
add_manure_bedding(split, bedding, options = list())
```

Arguments

`split`	A tibble from `split_manure_management()`.
`bedding`	A tibble from `build_residue_bedding_supply()`, keyed on the `year x territory` (x `sub_territory`) columns it shares with `split`.
`options`	A named list. `mms_bedding` selects the litter-using systems: `"ipcc_2019"` (default) or `"with_daily_spread"`; see the section above.

Value

The `split` rows with `dm_bedding`, `n_bedding`, `c_bedding` and `method_bedding_mms`.

Which systems use litter

IPCC 2019 Refinement Vol. 4 Ch. 10 p. 10.94 defines `NbeddingMS` as the bedding nitrogen "to be applied for solid storage and deep bedding MMS if known organic bedding usage". WHEP's `"Poultry Manure"` is the deep-litter system (it takes its EF3 from the IPCC `"Poultry Manure - Deep Litter"` row), so the IPCC pair is `"Solid Storage"` and `"Poultry Manure"` and that is the default. `"with_daily_spread"` additionally beds `"Daily Spread"`, the manure scraped from a barn and spread the same day, which is bedded in practice but which IPCC does not name; it is selectable rather than default for that reason. Liquid and lagoon systems are excluded under both, since straw cannot be pumped.

Moving bedding onto `"Daily Spread"` as well spreads the same mass over more nitrogen, so it lowers the bedding placed per tonne of solid-storage manure and raises the applied C:N of the daily-spread stream instead; it moves no total.

Examples

```
excretion <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~livestock_category,
  ~n_excretion, ~c_excretion, ~vs_excretion,
  2020L, "203", NA, "Cattle_milk", 100, 1000, 210
)
bedding <- tibble::tibble(
  year = 2020L, territory = "203",
  bedding_dm_t = 50, bedding_c_t = 22, bedding_n_t = 0.3
)
add_manure_bedding(split_manure_management(excretion), bedding)
```

---

add_polity_code *Add WHEP polity codes to a table*

Description

Adds periodized `polity_code` information from polity_area_crosswalk to a table with FAOSTAT/FABIO `area_code` values. If a `year` column is present, the mapping is year-aware; otherwise the current/default mapping is used.

When no mapped period covers a row's year, another period of the same area is used as a stand-in and `mapping_status` reports `"out_of_span"` rather than the crosswalk's `"matched"`/`"manual"`. Such a row is attributed to a polity that did not exist in that year, so treat it as a coverage gap: either the area needs the missing period added to the crosswalk, or the reporting area outlived (or predates) every polity mapped to it.

Usage

```
add_polity_code(
  table,
  code_column = "area_code",
  year_column = "year",
  polity_code_column = "polity_code",
  backcast_anchor = 1961L,
  backcast_polity = NULL
)
```

Arguments

`table`	A data frame.
`code_column`	Name of the column containing numeric area codes.
`year_column`	Name of the column containing years. Set to `NULL` to force current/default mapping.
`polity_code_column`	Name of the output polity-code column.

`backcast_anchor`
: First year of reported (non-back-cast) FAOSTAT data, default `1961`. Years before it are matched to the polity active in the anchor year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory rather than reported under their data-year borders. Such a row reports `mapping_status == "backcast_anchor"` where the anchor polity is not live in its own year. Set to `-Inf` to disable and match strictly by data year.

`backcast_polity`
: Which territory a row before `backcast_anchor` is labelled with: `"anchor"` or `"present_day"`. See the section above. `NULL` (default) reads `getOption("whep.backcast_pol "anchor")`.

Value

A tibble with added polity metadata columns.

The status vocabulary

`mapping_status` here is a property of resolving one `(area_code, year)`, **not** the same column as polity_area_crosswalk's, which is a property of a published crosswalk row (whep#637). The resolver carries the selected crosswalk row's own status through, and overwrites it wherever the resolution substituted something for a real period hit:

- `"matched"` / `"manual"`: the year fell inside the polity's period, and the value is the crosswalk row's own provenance, carried through.
- `"backcast_anchor"`: the row is before `backcast_anchor`, so it was resolved at the anchor year (or at 2023 under `backcast_polity = "present_day"`), and the polity live then is **not** live in the row's own year. That polity is still the honest label – the value is a reconstruction on the anchor year's territory – but the row is no evidence the polity existed then, which is exactly what `"matched"` asserts. FAOSTAT area 238 reads `ETH-1952-1993` from 1850, 102 years before that polity began: `"backcast_anchor"` for 1850-1951, `"matched"` from 1952. A pre-anchor row whose anchor polity *does* cover its own year keeps `"matched"`.
- `"out_of_span"`: no mapped period covered even the anchored year, so a nearest-period stand-in was used.
- `"unmapped"`, or `NA`: no polity at all, carried through from the crosswalk or left by an area with no applicable period. `polity_code` is `NA` too.

`"backcast_anchor"` and `"out_of_span"` exist only here, so a tibble carrying either is unambiguously this column and not the crosswalk's. The two still overlap in `"matched"`, `"manual"` and `"unmapped"`, which is whep#637 and is not resolved here.

Which stand-in is picked

A period that has **not started yet** is preferred over one that has already **ended**, and only then is the nearest in years taken. Ranking by distance alone split one reporting area's series between two entities at whichever year the arithmetic flipped: FAOSTAT area 178 Eritrea read `ERI-1889-1952` (the Italian colonial administration) up to 1972 and `ERI-1993-2025` from

1973, and area 273 Montenegro split at 1961 between `MNE-1913-1918` and `MNE-2006-2025` on a one-year margin. Preferring the not-yet-started period keeps each area on one entity and agrees with the back-cast anchor, whose purpose is to avoid resolving back-cast rows onto a larger historical-extent period. Set `options(whep.polity_stand_in = "nearest")` to restore ranking by distance alone; it changes 235 of the crosswalk's 46,640 `(area, year)` pairs over 1850-2025, all of them areas 178 and 273 (whep#705).

Which territory a back-cast row names

A row before `backcast_anchor` has two territorial references, and they differ wherever a territory changed after the anchor (whep#748). Its **level** is the area's value at the anchor, walked backwards by `fill_proxy_growth()`, so it describes the anchor year's territory. Its year-on-year **movement** is a ratio of LUH2 land keyed on present-day ISO3, so it describes today's territory. `polity_anchor_drift()` reports where the two differ. `backcast_polity` chooses which one the label names:

- `"anchor"` (default): the polity live at `backcast_anchor`, the territory of the level. Area 238 Ethiopia reads `ETH-1952-1993`, which includes Eritrea, from 1850 to 1992.
- `"present_day"`: the polity live in 2023, the territory of the movement and the reference year `polity_anchor_drift()` measures against. Area 238 reads `ETH-1993-2025` from 1850 to 1960, `ETH-1952-1993` from 1961 to 1992 and `ETH-1993-2025` after. An area with no polity live in 2023, such as 15 Belgium-Luxembourg or 248 Yugoslav SFR, keeps its anchor polity, because there is no present-day territory to name.

Either way the row's `mapping_status` is `"backcast_anchor"` where the polity named is not live in the row's own year. Neither choice moves a value of the table being labelled or a `polity_area_code` bucket: over the crosswalk's 1850-1960 grid, `"present_day"` relabels 5,106 `(area, year)` pairs of 46 areas and leaves every bucket where it was. A consumer that joins on `reporting_polity_code` follows the label it is given.

Without an explicit argument the session option `options(whep.backcast_polity = )` decides, and it is the same switch every published output's `reporting_polity_code` and `partner_polity_code` follow, so one setting relabels them all consistently.

See Also

`polity_coverage_gaps()`, which reports the `"out_of_span"` and `"backcast_anchor"` rows of an already-built table, whose published columns no longer carry `mapping_status`.

Examples

```
# The same area code resolves to different polities in different years:
# area 16 reports as East Pakistan before 1971 and as Bangladesh after it.
tibble::tibble(area_code = c(16L, 16L), year = c(1965L, 2000L)) |>
  add_polity_code() |>
  dplyr::select(area_code, year, polity_code, polity_name, mapping_status)

# Without a year column the current/default mapping is used.
add_polity_code(tibble::tibble(area_code = 231L), year_column = NULL) |>
  dplyr::select(area_code, polity_code, polity_name)
```

`admin_coverage_prototype`
Zero-row prototype of the run's admin-coverage report

Description

The schema of `admin_coverage.csv`, which `run_spatialize()` writes beside `run_metadata.yaml` whenever a run requests a depth. One row per container x item x year, saying which admin source constrained it and at what grain. The populated table is produced by the T25 resolver; until a run is wired to one, the file is written with this header and no rows, so a reader can tell "no coverage was granted" from "the file was never written".

It IS the resolver's own coverage schema, returned rather than restated: this function had a second, hand-written list of names (`source`, `tier`, `grain`, and a character `not_shipped`) where `resolve_admin_shares()` emits `resolved_source`, `resolved_tier`, `resolved_grain` and a logical, so handing the resolver's table to the writer aborted on three missing columns, and the writer's subset would have dropped the resolution audit trail (`resolved_indicator`, `resolved_nuts_version`, `resolution_rule`) from the file.

Usage

```
admin_coverage_prototype()
```

Value

A zero-row `tibble` with the report's columns.

Examples

```
admin_coverage_prototype()
```

`admin_items_eurostat` *Eurostat crop classes and their WHEP items*

Description

The 79 classes of the `crops` dimension shared by the `apro_cpshr` and `apro_cpnhr_h` dataflows, each mapped to its WHEP `item_prod_code` or recorded as dropped. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_items_eurostat
```

Format

A tibble with one row per Eurostat crop class and the eight `admin_items_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies, crops_eurostat

Examples

```
head(admin_items_eurostat)
```

`admin_items_jrc` *JRC subnational crop classes and their WHEP items*

Description

The nine crop classes of release 2025.01 of the JRC harmonised EU subnational crop statistics. Two of them, `"Total wheat"` and `"Total barley"`, are aggregates of members the same release also publishes, so summing all nine would double-count wheat and barley; the aggregates bind and their members are recorded but never summed alongside them. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_items_jrc
```

Format

A tibble with one row per JRC crop class and the eight `admin_items_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies

Examples

```
head(admin_items_jrc)
```

`admin_items_nass` *USDA NASS crop series and their WHEP items*

Description

The Quick Stats `SHORT_DESC` crop series the subnational spatialization reads, each mapped to its WHEP `item_prod_code`. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_items_nass
```

Format

A tibble with one row per NASS series and the eight `admin_items_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies

Examples

```
head(admin_items_nass)
```

admin_items_sidra *IBGE SIDRA crop classes and their WHEP items*

Description

The 72 categories of classification 782 of SIDRA table 5457 (Producao Agricola Municipal), each mapped to its WHEP `item_prod_code` or recorded as dropped. Category `"0"` is the all-crops total and is dropped as such. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_items_sidra
```

Format

A tibble with one row per SIDRA crop category and the eight `admin_items_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies

Examples

```
head(admin_items_sidra)
```

admin_shares_prototype *The admin-shares zero-row prototype*

Description

The zero-row tibble `admin_shares_schema()` describes: the declared columns, in the declared order, each with the declared type. Built with `empty_table_from_schema()`, so it passes `check_table_schema()` by construction and cannot drift from the schema. See `admin_shares_schema()` for the full column-by-column contract.

Usage

```
admin_shares_prototype()
```

Value

A zero-row tibble with the admin-shares contract's columns, in contract order.

Examples

```
admin_shares_prototype()
nrow(check_table_schema(admin_shares_prototype(), admin_shares_schema()))
```

---

admin_shares_schema *The admin-shares table contract*

Description

Declare, as data, the shared row shape every admin-shares reader emits and every admin-shares consumer reads: one subnational administrative unit's reported value and its share of the FAOSTAT container's total, for one `(item_prod_code, indicator_used, year)`. Readers emit source-native identifiers and never resolve names to WHEP codes here; a later resolution step reconciles those once. Unresolved rows keep `level_polity_code == NA` visibly rather than being dropped.

There are two measurement columns, `value` and `share`, and a row must carry at least one of them. `value` may be missing wherever `share` is present, which is a **first-class case** created by a publication consent and not a gap to be filled; see the section below before writing anything that assumes an absolute value is there.

`admin_shares_schema()` is the `check_table_schema()` contract itself; `admin_shares_prototype()` is the zero-row tibble it implies, and the two cannot drift apart because the prototype is built *from* the schema with `empty_table_from_schema()`.

Usage

```
admin_shares_schema()
```

Value

A schema list, as documented in `check_table_schema()`: closed (`extra_columns = "forbid"`), keyed on `(area_code, level_polity_code, level, item_prod_code, species_group, indicator_used, ye`

Admin-shares table

One row per `(area_code, level_polity_code, level, item_prod_code, species_group, indicator_used,`

- `area_code`: FAOSTAT-style area code of the *container* – the polity the subnational units sum into, not the unit itself.
- `level_polity_code`: the polity code resolved at the row's granted depth (`level`); `NA` when that depth is not yet resolved to a polity.

- `level`: administrative depth granted for this row, a positive integer (`1L` is the container's direct subnational units).
- `item_prod_code`: WHEP production-item code (see `add_item_prod_code()` / `add_item_prod_name()`) on a crop row; `NA` on a livestock row.
- `species_group`: the livestock group a livestock row counts, in the vocabulary `build_gridded_livestock()` allocates in (`"cattle_dairy"`, `"pigs"`, `"sheep_goats"`, ...); `NA` on a crop row. A row names exactly one of `item_prod_code` and `species_group`.
- `indicator_used`: which indicator this row's `value` anchors to, one of `"area_harvested"`, `"area_planted_or_sown"`, `"area_main"`, `"area_cultivated"`, `"production"`, `"yield"` for a crop row, or `"head_count"` (live animals, head) for a livestock row.
- `year`: calendar year of the observation.
- `value`: the unit's own reported value for `indicator_used`, in the source's native unit. `NA` on a row whose source is consented to ship shares only – see *Shares-only rows* below. Never invented.
- `share`: the unit's own value divided by the admin sum across sibling units for the same `(area_code, level, item_prod_code, indicator_used, year)`, taken by the row's **own producer** over its own units. `NA` where the producer ships values and that sum has not been taken. This package does not fill it in at load: a share derived here from the values in the same table would make the seam gate's value-versus-share identity (`R/admin_shares_gate.R`, tier A) true by construction, and an identity that cannot fail detects nothing.
- `source`: dataset label of the row's producer, e.g. `"USDA_NASS"`, `"Eurostat_apro_cpshr"`, `"Eurostat_apro_cpnhr_h"`, `"Eurostat_apro_mt_ls_r"`, `"Eurostat_ef_lsk_poultry"`, `"IBGE_PAM"`, `"IBGE_PPM"`, `"JRC_subnational_crops"`, or a tier-2/3 admin-statistics family label. Documented here, **not** enforced as a closed vocabulary: later tiers add sources this list cannot enumerate in advance, unlike `indicator_used`, `grain` and `treatment_year`, which the contract does close. A caller may therefore put any label here, and `resolve_admin_shares()` will rank it. Where the vocabulary *is* closed is at the pin boundary: every source in the assembled `admin-shares` pin must be declared before `read_admin_shares()` will hand it over, because that artifact carries sources whose values are withheld by a publication consent and an undeclared label is indistinguishable from a new one.
- `tier`: data tier of the source, `1L`-`3L`.
- `grain`: the reporting geography's fineness, one of `"admin1"`, `"admin2"`, `"admin3"`, in that ascending order. Stored as `character` rather than an R `ordered` factor: `check_table_schema()`'s type vocabulary (`R/table_schema.R`) has no factor type, so the ordering is carried by the vocabulary's declared order rather than by the column's R class. A caller needing genuine ordered comparisons can do `factor(grain, levels = c("admin1", "admin2", "admin3"), ordered = TRUE)`.
- `concept_break`: whether the item concept changed where this row's source or grain took over from another.
- `nuts_version`: the NUTS nomenclature version the row's geography was coded under, `NA` outside NUTS geographies.
- `source_native_id`: the row's identifier exactly as the reader found it in the source; `NA` for a row with no native identifier, e.g. one derived rather than read (such as a residual).

- `source_native_name`: the row's name exactly as the reader found it, diagnostic only – never joined or matched on.
- `source_id`: the producer's immutable identifier, in `row_evidence()`'s vocabulary (`R/row_evidence.R`) – the same value space as `source`, so an admin-shares table can be handed to the row-evidence family without renaming.
- `source_version`: version or vintage of that source, `NA` when the producer has none, exactly as `row_evidence()` documents it.
- `recorded_at`: when the row was recorded, as an ISO 8601 UTC string (`"2026-01-01T00:00:00Z"`), the same stamp shape `row_evidence()` writes.
- `treatment_year`: how this row's year was obtained, one of `"observed"`, `"interpolated"`, `"carried"`. Reserved for the gap rule; this contract only names the vocabulary.
- `treatment_value`: how the row's value was obtained, `"observed"` (reported by the unit) or `"reconstructed"` (derived by the source, e.g. from a national total). `NA` when the source does not say, which is every source today; it is never defaulted to `"observed"`.
- `value_flag`: a free-text data-quality flag, `NA` when the row is clean.

Shares-only rows

A row may carry `share` and no `value`. That is not a defect, not a missing observation and not a gap for a later step to fill: it is what a publication consent produces, and the contract admits it on purpose.

The Latin American subnational panel of Infante-Amate, Urrego-Mesa, Badia-Miro and Aguilera ships to WHEP as **derived shares only** – 875,514 rows over 142 first-level units of Argentina, Bolivia, Brazil, Chile, Colombia and Mexico – under the co-author agreement of 2026-09-02 recorded in `inst/extdata/admin_stats_pins_manifest.csv`. Its source values are withheld until that panel's own publication, so WHEP may carry each unit's share of its container's total and nothing else. Five of those six countries have no other subnational evidence in this package. Demanding a `value` would therefore not have improved the data: it would have dropped five countries out of the subnational constraint while every balance and conservation check still passed.

A synthetic value is forbidden, in both the forms that tempt:

- `value = 0` satisfies the contract and then enters the allocation as a reported area of zero – a claim the source never made, and one that a downstream reader cannot tell from a real zero.
- a value back-computed as `share * national total` satisfies it too, and additionally reconstructs the quantity the consent withheld.

Neither is acceptable, and finding either in code is a defect to report rather than a shortcut to reuse. A shares-only row travels as a shares-only row: `resolve_admin_shares()` ranks candidates on indicator, grain, tier and run length, none of which reads `value`, and `allocate_level_crops()` (`R/spatialize_levels.R`) carries a `"share_normalised"` denominator for exactly this case.

What the contract does still refuse is a row carrying **neither** measurement: `ensure_admin_shares()` and `resolve_admin_shares()` abort on one with class `whep_error_admin_no_measure`,

because such a row constrains nothing and would enter an allocation as an invisible abstention.

It equally refuses a **non-finite** `value` or `share`, with class `whep_error_admin_nonfinite`. `NaN` is not a missing measurement: it is what a 0/0 leaves behind, and since `is.na(NaN)` is `TRUE` every `is.na(value)` branch downstream would read it as the consented shares-only case above. `Inf` is refused with it, which no schema bound catches either – `check_table_schema()` guards `min` and `max` with `!is.na(values)`, and `value` has no maximum.

Examples

```
admin_shares_schema()

# A schema-conformant table: two sibling units of one container.
rows <- tibble::tibble(
  area_code = c(840L, 840L),
  level_polity_code = c("USA-IOWA", "USA-ILLINOIS"),
  level = c(1L, 1L),
  item_prod_code = c(44L, 44L),
  indicator_used = c("area_harvested", "area_harvested"),
  year = c(2020L, 2020L),
  value = c(1000000, 800000),
  share = c(0.42, 0.34),
  source = c("USDA_NASS", "USDA_NASS"),
  tier = c(1L, 1L),
  grain = c("admin1", "admin1"),
  concept_break = c(FALSE, FALSE),
  nuts_version = NA_character_,
  source_native_id = c("19", "17"),
  source_native_name = c("Iowa", "Illinois"),
  source_id = c("USDA_NASS", "USDA_NASS"),
  source_version = c("2021-05", "2021-05"),
  recorded_at = "2026-01-01T00:00:00Z",
  treatment_year = c("observed", "observed"),
  value_flag = NA_character_
)
nrow(check_table_schema(rows, admin_shares_schema()))

# A shares-only pair, as a consented source ships it: `share` present,
# `value` absent, and the contract satisfied.
consented <- rows |>
  dplyr::mutate(
    value = NA_real_,
    share = c(0.79, 0.21),
    source = "admin-stats-latam",
    source_id = "admin-stats-latam",
    tier = 3L
  )
nrow(check_table_schema(consented, admin_shares_schema()))
```

`admin_source_vocabularies`
Source-item vocabularies for the subnational admin statistics

Description

Seven tables mapping the crop and livestock classes the subnational admin-statistics readers return – verbatim, in each publisher's own identifiers – onto WHEP's own item and species vocabularies. Four cover crops (`admin_items_*`) and three cover livestock (`admin_species_*`); the JRC dataset publishes no livestock, so it has no species table.

The tables are vocabulary, not data: they say what a source class means in WHEP terms and, where it means nothing exactly, that it is dropped and why. Nothing here converts, sums or reads a value.

Format

The four `admin_items_*` tables share eight columns:

- `source`: the reader's own source label (`"USDA_NASS"`, `"Eurostat"`, `"IBGE_PAM"`, `"JRC_subnational_crops"`).
- `source_table`: the dataflow, dump or release the class belongs to. `"apro_cpshr|apro_cpnhr_h"` where the two Eurostat crop dataflows share one vocabulary.
- `class_key`: the table's unique key, the source class code where the publisher issues one and the class name where it does not.
- `source_class_code`, `source_class_name`: the class exactly as the source publishes it. The code is `NA` for JRC and NASS, neither of which issues one.
- `item_prod_code`: the WHEP items_prod_full key, as character. `NA` on a dropped class.
- `mapping_kind`, `mapping_reason`: as described above.

The three `admin_species_*` tables carry the same columns with `item_prod_code` replaced by two:

- `species_group`: the WHEP livestock functional type of `inst/extdata/livestock_mapping.csv` (`"cattle_dairy"`, `"pigs"`, `"sheep_goats"`, ...). `NA` where the class maps to no single group.
- `constrains`: for an `"aggregate"` class that maps to no single group, the `"+"`-joined set of groups whose **sum** it constrains, for example `"cattle_dairy+cattle_non_dairy"`. `NA` on every other kind, which the builder enforces.

Mapping kinds

`mapping_kind` takes five values, and the distinction between the first four is what stops a source total being counted twice:

- `"exact"`: one source class, one WHEP target.

- `"aggregate"`: a total the publisher ships alongside its own members. The aggregate binds; its members are never summed alongside it.
- `"member"`: one of those members. It is recorded so the class is accounted for rather than silently absent, and is not summed.
- `"sum_member"`: one of several classes that **do** sum to the WHEP target, the publisher shipping no aggregate for them.
- `"dropped"`: no exact WHEP counterpart. The class is never folded into a near neighbour; `mapping_reason` says what it is and why it could not be mapped, so a coverage report can count it.

Every row that is not `"exact"` carries a `mapping_reason`, and the builder aborts on one that does not.

The rules these tables encode

The tables decide nothing. They are the data form of the vocabulary rules fixed for the subnational spatialization, and each row's `mapping_reason` names the rule it applies:

- Crops. Where a publisher ships an aggregate class, that aggregate binds and its members are never summed alongside it. Maize is grain maize only, silage and forage maize excluded. A source rice area is paddy and binds `item_prod_code` 27. A class with no exact WHEP counterpart is dropped and counted, never folded into a near neighbour.
- Livestock. A published dairy series binds dairy cattle, and non-dairy is the total minus the dairy series, the unit-year being refused where that difference is negative. A combined sheep-and-goats class constrains their sum. Published reference dates are accepted as they are, which is why the NASS hog inventory keeps its 1 December date. A combined poultry class constrains the sum of the poultry groups.

Where the class lists come from

Each table enumerates the classes its source actually serves, taken from the source itself rather than from a reader's defaults:

- Eurostat: the `crops` dimension of `apro_cpshr` and `apro_cpnhr_h` (identical 79-class vocabularies), the `animals` dimension of `apro_mt_ls_r` (47 classes) and of `ef_lsk_poultry` (9 classes), read off the dissemination API on 2026-09-04.
- IBGE SIDRA: classification 782 of table 5457 (72 categories, one of them the all-crops total) and classification 79 of table 3939 (10 herd types), from the `servicodados.ibge.gov.br` metadata endpoint on 2026-09-04, plus table 94, whose single series is milked cows.
- JRC: the nine crop classes of release 2025.01 of the harmonised EU subnational crop statistics, as `inst/scripts/prepare_jrc_subnational.R` measured them on the pinned archive.
- USDA NASS: the Quick Stats `SHORT_DESC` series `read_admin_stats_nass()` requests, plus the silage-maize series the maize rule excludes. This is deliberately **not** an inventory of the NASS vocabulary: Quick Stats keys tens of thousands of series on `SHORT_DESC`, the bulk dump is the only complete listing of them, and no offline copy of it ships with this package.

crops_eurostat is the older, label-only Eurostat crop table (13 green fodder and root-crop codes, no WHEP target). It is left as it is; `admin_items_eurostat` is the vocabulary that carries the mapping, and covers those 13 codes among its 79.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

`read_admin_stats_nass()`, `read_admin_stats_eurostat()`, `read_admin_stats_sidra()`, items_prod_full, crops_eurostat.

`admin_species_eurostat`

Eurostat livestock classes and their WHEP species groups

Description

The 47 classes of the `animals` dimension of `apro_mt_ls_r` and the nine of `ef_lsk_poultry`, each mapped to its WHEP `species_group`, to the set of groups whose sum it constrains, or recorded as a member of a binding aggregate. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_species_eurostat
```

Format

A tibble with one row per Eurostat animal class and the nine `admin_species_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies

Examples

```
head(admin_species_eurostat)
```

`admin_species_nass` *USDA NASS livestock series and their WHEP species groups*

Description

The Quick Stats inventory series the subnational spatialization reads, each mapped to its WHEP `species_group` or to the set of groups whose sum it constrains. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_species_nass
```

Format

A tibble with one row per NASS series and the nine `admin_species_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies

Examples

```
head(admin_species_nass)
```

admin_species_sidra *IBGE SIDRA herd types and their WHEP species groups*

Description

The ten herd types of classification 79 of SIDRA table 3939 (Pesquisa da Pecuaria Municipal), plus the milked-cow series of table 94, which is the only dairy split IBGE publishes for that herd. See admin_source_vocabularies for the columns, the mapping kinds and the rules encoded here.

Usage

```
admin_species_sidra
```

Format

A tibble with one row per SIDRA herd type, plus one for table 94, and the nine `admin_species_*` columns of admin_source_vocabularies.

Source

Eurostat dissemination API (https://ec.europa.eu/eurostat/api/dissemination/sdmx/2.1), dataflows `apro_cpshr`, `apro_cpnhr_h`, `apro_mt_ls_r` and `ef_lsk_poultry`; IBGE SIDRA aggregate metadata (https://servicodados.ibge.gov.br/api/v3/agregados), tables 5457, 3939 and 94; Ronchetti, G. et al. (2024). Harmonized European Union subnational crop statistics reveal climate impacts and crop cultivation shifts. *Earth System Science Data*, 16, 1623-1649. doi:10.5194/essd1616232024; USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/).

See Also

admin_source_vocabularies

Examples

```
head(admin_species_sidra)
```

aggregate_grass_to_polity *Aggregate gridded grass availability to polity totals.*

Description

Sums gridded grass availability to polity (country, or subnational where available) totals, splitting each cell's grass by the cell's land-area share in each polity so border cells are attributed proportionally. The polity grass supply ceiling for feed allocation.

Usage

```
aggregate_grass_to_polity(
  grass,
  cell_polity,
  polity_validity = c("keep", "flag", "drop")
)
```

Arguments

`grass`	Gridded grass availability from `build_grass_availability()`, with `lon`, `lat`, `year` and `grass_avail_dm_t`.
`cell_polity`	Cell-to-polity mapping with `lon`, `lat`, `area_code` and `polity_frac` (the cell's land-area fraction in the polity; pass 1 for a majority assignment, e.g. from `country_grid`).
`polity_validity`	What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

Value

A tibble with `area_code`, `year` and `grass_avail_dm_t`, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`. This output carries no reporting-polity columns, so the flag is attached directly rather than derived from them.

Examples

```
grass <- build_grass_availability(method = "lpjml", example = TRUE)
cp <- tibble::tibble(
  lon = grass$lon,
  lat = grass$lat,
  area_code = 1L,
  polity_frac = 1
)
aggregate_grass_to_polity(grass, cp)
```

---

aggregate_unit_extent

Aggregate a gridded land quantity onto administrative units

Description

Sum a per-cell land quantity onto the compartments of a granted-depth country grid, giving the per-unit extent `E_u(t)` the seam back-cast modulates shares with:

$$E_u(t) = \sum_c E_c(t) \times f(c, u)$$

where `f(c, u)` is `cell_area_frac`, the unit's share of the physical cell on the land basis. The gridded tables are the ones already in memory for the spatialization engine, so no LUH2 archive is re-read here: `read_luh2_landuse()` is never called, and the identity between the extent that modulates a share and the extent that weights cells within a unit holds by construction rather than by assertion. What the construction cannot prove is which LUH2 vintage the caller's tables came from; the prep-side provenance stamp on those tables carries that.

Usage

```
aggregate_unit_extent(
  gridded,
  country_grid,
  quantity,
  extent_by = NULL,
  container_frac = NULL,
  residual_code = NULL,
  tolerance = 1e-08
)
```

Arguments

`gridded`	A tibble of per-cell land quantities keyed on `lon`, `lat`, `year`, plus any extra key columns named in `extent_by`, plus the numeric columns named by `quantity`.
`country_grid`	A tibble of compartments: `lon`, `lat`, `area_code`, `level_polity_code`, `level`, `cell_area_frac`. One row per `(lon, lat, area_code, level_polity_code)`; an interval-grained layer must be filtered to one geometry first.
`quantity`	Either a character vector of `gridded` columns summed with weight 1, or a lookup table with a `quantity` column of column names, an optional numeric `weight` column (default 1) and the `extent_by` key columns the lookup is keyed on. Several rows per key sum several columns.
`extent_by`	Character vector of extra key columns of the output. Each must be a column of `gridded` (carried through the aggregation) or a key column of a `quantity` lookup (which selects columns per key), never both.
`container_frac`	Optional tibble `lon`, `lat`, `area_code`, `cell_area_frac` giving the container's own share of each cell on the same land basis. Required by `residual_code`, ignored without it.

`residual_code` Optional `level_polity_code` for a residual pseudo-unit, whose extent is `E_res(t) = E_container(t) - sum_u E_u(t)`. `NULL` (default) emits no residual row, which is the right shape unless the allocation runs under the residual policy.

`tolerance` Relative tolerance below which a negative residual extent counts as floating-point noise and is clamped to zero. A larger negative residual aborts: it means the unit fractions exceed the container's.

Value

A tibble: `area_code`, `level_polity_code`, `level`, the `extent_by` columns, `year`, `extent_ha` and `extent_basis` (the `weight * column` terms behind the row, `+`-joined in a stable order). One row per unit, key and year for which at least one compartment of that unit met a gridded cell.

Mirroring the engine's land quantity

`R/spatialize.R:337-405` builds the crop weight from `cropland_ha = i.cropland_ha * cell_area_frac`; a type-aware run replaces that with `type_ha * cell_area_frac` joined on `(lon, lat, luh2_type)`, zeroes cells with no row for the crop's type, and restores total cropland for a whole `(area_code, item_prod_code)` group whose type potential `sum(harvest_fraction * cropland_ha)` is not positive. `R/spatialize_livestock.R:358-405` builds the livestock weight from `pasture_ha + rangeland_ha` (`pasture`), `rangeland_ha` (`rangeland`), `cropland_ha` (`cropland`) or `0.5 * (pasture_ha + rangeland_ha) + 0.5 * cropland_ha` (`mixed`), also times `cell_area_frac`.

This function mirrors the `cell_area_frac` multiplication and the compartment sum. Which land column stands for a given item or species group is the caller's choice, declared through `quantity`: a character vector applies one weighted column set to every row, a lookup table applies a different set per key. The engine's whole-group fallback to total cropland turns on `harvest_fraction`, a within-unit weight rather than a land extent, so the caller evaluates it and passes the resulting per-item choice in the lookup; the choice comes back on `extent_basis`, so the land quantity behind each row is recorded rather than inferred.

Territory basis

`country_grid` is used exactly as given, unfiltered by edge validity. That is the t0-geometry convention of the plan's seam section: shares and `E_u` are defined on the t0 unit set with t0 geometry held fixed, so a unit whose containment edge starts after a back-cast year still contributes its t0 cells at that year. Pass the layer already filtered to the t0 reference year (`.filter_country_grid_year()`) when the t0 geometry differs from the delivered layer.

Examples

```
grid <- tibble::tibble(
  lon = c(10.25, 10.75, 10.75),
  lat = 40.25,
  area_code = 900L,
  level_polity_code = c("A1", "A1", "A2"),
  level = 1L,
  cell_area_frac = c(1, 0.3, 0.5)
```

```
)
cells <- tibble::tibble(
  lon = c(10.25, 10.75),
  lat = 40.25,
  year = 1900L,
  cropland_ha = c(1000, 1200)
)
# A1 = 1000 * 1 + 1200 * 0.3 = 1360; A2 = 1200 * 0.5 = 600.
aggregate_unit_extent(cells, grid, "cropland_ha")
```

`align_extension` *Align an extension table to input-output sector labels.*

Description

Turn a long-format extension table into the dense per-sector numeric vector that `compute_footprint()` expects, ordered to match a single year's `labels` from `build_io_model()`. Sectors absent from the extension are filled with zero, and extension rows outside the model are dropped. Rows sharing an `(area_code, item_cbs_code)` are summed.

Usage

```
align_extension(extension, labels, year, value_col = "impact_u")
```

Arguments

`extension`	Long-format extension tibble with `year`, `area_code`, `item_cbs_code` and the column named by `value_col`.
`labels`	One year's `labels` tibble from `build_io_model()`, with `area_code`, `item_cbs_code` and `index` columns.
`year`	Year to select from `extension`.
`value_col`	Name of the extension magnitude column, `"impact_u"` by default.

Value

A numeric vector with one entry per row of `labels`, ordered by the label `index`.

Examples

```
extension <- tibble::tibble(
  year = 2000L,
  area_code = 1L,
  item_cbs_code = 10L,
  impact_u = 5
)
labels <- tibble::tibble(
  index = 1:2,
  area_code = c(1L, 1L),
```

```
  item_cbs_code = c(10L, 20L)
)
align_extension(extension, labels, 2000L)
```

`allocate_grazing_to_products` *Allocate grazing land forward to livestock products.*

Description

Push grazing land forward along the feed chain instead of tracing it backward through a Leontief inverse. Grass is not traded, so the multi-regional input-output footprint leaks the grazing land that meat and dairy consumption actually drives (the feed-conversion columns make the system non-productive). This function hands every tonne of grazing land forward in two mass-conserving steps:

1. pool each country's grazing land and split it across the grazing animals that fed there, in proportion to their forage intake;
2. split each animal's share across its output products in proportion to production mass, landing the land on the meat and milk items that are actually traded.

The result is a direct-land extension keyed by livestock output item, ready to route to consumers with `compute_footprint_balance()`. Because each step redistributes a total without creating or destroying it, the country-level land total is conserved whenever an animal fed and produced an output there. Land fed to animals that produced no eligible output item is set aside and reported via a warning rather than silently dropped.

Usage

```
allocate_grazing_to_products(
  grass_land,
  grazer_intake,
  livestock_production,
  products = c("all", "meat_milk")
)
```

Arguments

`grass_land`	Tibble with `year`, `area_code` and `value` (grazing land, e.g. hectares). Multiple rows per country (for example one per grass item) are pooled.
`grazer_intake`	Tibble with `year`, `area_code`, `live_anim_code` and `value` (the intake by which to split the land across animals), as from `get_feed_intake()`. Using grazer forage intake (grass plus roughage residues) rather than the `"grass"` feed type alone keeps extensive-grazing countries, whose forage the feed model classes as residues, from dropping out.
`livestock_production`	Tibble with `year`, `area_code`, `live_anim_code`, `item_cbs_code` and `value` (output production, tonnes).

`products` Which output items absorb the land. `"all"` (default) spreads it across every livestock output by mass; `"meat_milk"` restricts it to meat and dairy items, so all grazing land lands on meat and dairy consumers per the forward-allocation goal.

Value

A tibble with `year`, `area_code`, `item_cbs_code` (a livestock output item), `value` (grazing land allocated to it) and `method_allocation` (the chosen `products`).

Examples

```
grass_land <- tibble::tibble(
  year = 2010L, area_code = 1L, value = 100
)
grazer_intake <- tibble::tibble(
  year = 2010L,
  area_code = 1L,
  live_anim_code = c(960L, 976L),
  value = c(75, 25)
)
livestock_production <- tibble::tibble(
  year = 2010L,
  area_code = 1L,
  live_anim_code = c(960L, 960L, 976L),
  item_cbs_code = c(2848L, 2731L, 2732L),
  value = c(90, 10, 5)
)
allocate_grazing_to_products(
  grass_land, grazer_intake, livestock_production
)
```

`allocate_level_crops` *Allocate national crop areas at a granted containment depth*

Description

The two-level allocation: a national total is split across the administrative units of its container by reported harvested-**area** shares, and each unit's target is then spread across that unit's cells by the gridded pattern the engine already uses, under one capacity ceiling, in ONE pass per country. Items with no statistics travel in the same pass on pattern-implied unit shares, so a constrained country is allocated at one grain and a cell's capacity is shared by everything in it.

Usage

```
allocate_level_crops(
  country_areas,
  crop_patterns,
```

```
  gridded_cropland,
  allocation_layer,
  admin_shares = NULL,
  config = list()
)
```

Arguments

`country_areas`	National crop areas, as `build_gridded_landuse()` takes them: `year`, `area_code`, `item_prod_code`, `harvested_area_ha` and optionally `irrigated_area_ha`. Keyed on the CONTAINER; the unit split is this function's job. A negative `harvested_area_ha` is refused first, as `whep_alloc_negative_national`: the irrigation check would otherwise read it as a row with more irrigated than harvested area and say so.
`crop_patterns`	Per-cell crop pattern, as `build_gridded_landuse()` takes it.
`gridded_cropland`	Per-cell cropland extent, as `build_gridded_landuse()` takes it.
`allocation_layer`	One country grid choosing a depth per country, from `build_allocation_layer()`: granted countries arrive as their units, every other country as its level-0 row.
`admin_shares`	Resolved admin shares – the `shares` element of `resolve_admin_shares()`, optionally completed by `backcast_admin_shares()`. `NULL` allocates every item on pattern-implied unit shares.
`config`	Named list. Every `build_gridded_landuse()` key is forwarded (and validated there); `pattern_extension` is fixed to `"granted_units"` by decision T31(b). This function's own keys are `tolerance_relative` (0.10), `tolerance_absolute` (1000 ha) and `conservation_tolerance` (1e-6, relative).

Value

A list of seven tibbles:

- `allocation`: the gridded rows, at unit grain, exactly the `build_gridded_landuse()` schema plus `level_polity_code`.
- `targets`: one row per `(year, area_code, level_polity_code, item_prod_code)` with `share`, `target_ha`, `irrigated_target_ha`, `rainfed_target_ha`, `irrigation_clipped_ha` and the regime, `method_crop_alloc`.
- `coverage`: one row per `(year, area_code, item_prod_code)` with `n_units`, `n_units_reporting`, `coverage`, `admin_sum`, `residual_target_ha`, `discrepancy_ha` and the denominator `basis`.
- `breach`: the capacity excess per `(year, area_code, level_polity_code, item_prod_code, mc_basis)`.
- `straddle`: per unit and item, `n_cells`, `straddle_sibling`, `straddle_foreign` and `cell_limited`.

- `conservation`: allocated against target, at container and unit grain. The container's target is the NATIONAL total, so hectares that became no unit's target are visible here as well as in `coverage$dropped_ha`; the unit rows are warned about only where their container reconciles, which is the sibling-absorption failure they exist to see.
- `bridges`: per `(area_code, item_prod_code, treatment)`, how many years of the shares were not observed and the longest contiguous run of them, so a 60-year LUH2 bridge is visible rather than merely legal (decision T31(e)).

The composite key and where it must be carried

A granted-depth compartment is identified by `(area_code, level_polity_code)`: the container's integer reporting code, which every `area_code`-typed join, `add_area_name()` and the `area_key` switch keep working on, plus the unit's polity code from the containment edge (`NA` at level 0). The pair travels through the whole chain, and these are the places that had to learn it:

`.compartment_id_cols()` already appends `level_polity_code` (T12), so every capacity, redistribution and CFT grouping keyed on it followed for free.

The engine's national-table join and share denominators `.spatialize_year()` joined `country_areas` to the grid on `(area_code, item_prod_code)` and formed `rf_pot_sum` / `ir_pot_sum` / `rainfed_sum` / `irrigated_sum` on the same pair. Both now use `.alloc_target_cols()`, the grain the NATIONAL TABLE is keyed at. That is the load-bearing distinction: the key comes from the targets, never from the grid, because a unit-keyed grid under a container-keyed national table is the pattern-implied case and must stay one national total.

The LUH2 type-potential fallback `type_pot` decides per group whether a crop has any of its LUH2 type to sit in; grouped on the container it would let one unit's type cropland keep another unit out of the whole-group fallback.

`.redistribute_country_dt()` `.crop_group` was `item_prod_code` alone. It is now the target grain inside the country, so the logit passes and the final rescale conserve each (unit, item). This is the grouping error the machine criterion pins: with the item alone, a unit whose cells are too small has its excess pushed across the border into a sibling, the two units' targets swap, and every national total still reconciles.

`.warn_unallocated_crops()` reports at the target grain, so a unit that cannot place its target is visible instead of being averaged into its container's success.

The capacity ceiling `.capacity_bases()` keys the unit multi-cropping factor on `(area_code, level_polity_c`

Two places do **not** carry it, for stated reasons. `.redistribute_countries_dt()` still iterates `area_code`: it is a memory bound on the per-country working set, and the compartment key is inside each subset, so splitting further would only make the chunks smaller. `.compartment_interval_groups()` stays on `(lon, lat, area_code)` because `polycell_id` – and, at depth, the unit code – changes at a succession, which is exactly what its open-ended-interval test must see across.

What binds, and what is only measured

- The national total binds. Admin statistics set the within-country shape and nothing else; a yield or production row never binds (decision T31(i)) and is dropped, counted, by `build_level_crop_targets()`.

- The reported share binds inside the country. Where a unit's target does not fit its cells, the target wins and the ceiling gives way; the excess is measured per unit, at both multi-cropping bases, and returned in `breach` (decisions T31(c), T31(g)).
- Partial coverage raises a RESIDUAL pseudo-unit carrying `national - admin_sum`, supported by the non-reporting units' cells only. With complete coverage there is no residual: the reported units rescale proportionally and the leftover is a diagnostic, refused only beyond both tolerances (decisions T31(a), T31(d)).

See Also

`build_level_crop_targets()`, `build_allocation_layer()`, `build_gridded_landuse()`.

Examples

```
# Two units of container 1 (Armenia), one cell each, one crop.
layer <- tibble::tribble(
  ~lon,  ~lat, ~area_code, ~level_polity_code, ~level, ~cell_area_frac,
  0.25, 50.25,         1L, "A-A1-1900-2100",       1L,               1,
  0.75, 50.25,         1L, "A-A2-1900-2100",       1L,               1
)
shares <- tibble::tribble(
  ~area_code, ~level_polity_code, ~item_prod_code, ~year, ~value,
          1L, "A-A1-1900-2100",               15L, 2000L,     150,
          1L, "A-A2-1900-2100",               15L, 2000L,     100
) |>
  dplyr::mutate(level = 1L, indicator_used = "area_harvested")
out <- allocate_level_crops(
  country_areas = tibble::tibble(
    year = 2000L, area_code = 1L, item_prod_code = 15L,
    harvested_area_ha = 250
  ),
  crop_patterns = tibble::tibble(
    lon = c(0.25, 0.75), lat = 50.25, item_prod_code = 15L,
    harvest_fraction = c(0.5, 0.5)
  ),
  gridded_cropland = tibble::tibble(
    lon = c(0.25, 0.75), lat = 50.25, year = 2000L,
    cropland_ha = c(1000, 1000)
  ),
  allocation_layer = layer,
  admin_shares = shares
)
out$targets[c("level_polity_code", "target_ha", "method_crop_alloc")]
```

`allocate_manure_to_land` *Allocate field-available manure to cropland and grassland by crop.*

Description

Distributes the manure nitrogen, carbon and volatile solids that survive management (the output of `apply_management_losses()`) across cropland crops and grassland, with a local agronomic cap and an explicit overflow path. The collected/housed manure fills cropland up to each crop's cap (weighted by West-2014 receptivity by default), the surplus spills onto grassland up to the grassland cap, and any residual beyond all reachable sinks follows the `disposal_method`. The in-situ grazing stream is deposited on grassland where it falls, uncapped. Carbon and volatile solids ride along with each stream at its post-storage bundle ratio, so mass is conserved by construction.

Usage

```
allocate_manure_to_land(applied, gridded = list(), options = list())
```

Arguments

`applied`	A tibble from `apply_management_losses()` with at least `year`, `territory` (a stringified `area_code`, see `estimate_n_excretion()`), `sub_territory`, `stream`, `applied_n`, `applied_c` and `applied_vs`.
`gridded`	A named list describing the land surface for each polity: • `crops`: a tibble keyed by `year`, `territory`, `sub_territory`, `crop` (a code, `as.character(item_prod_code)`; this function treats it as an opaque label but the downstream nitrogen balance resolves it through items_prod_full, so a name is deprecated there) with the allocation weight (`manure_n_receptivity` for `"area_x_receptivity"`, `crop_n_demand` for `"crop_n_demand"`) and the cap basis (`crop_n_cap`, t N, for `"potential_uptake"`/`"reali` `crop_area_ha` for `"fixed_ceiling"`). • `grass` (optional): a tibble keyed by `year`, `territory`, `sub_territory` with `grass_n_cap` (t N) or `grass_area_ha`. The grassland cap is scaled by `f_n_tolerance` on the same footing as the uptake-based crop caps (not for `"fixed_ceiling"`). Absent grassland means no grassland sink (cap zero).
`options`	A named list of method options: `method` (`"area_x_receptivity"` default, or `"crop_n_demand"`), `cap_method` (`"potential_uptake"` default, `"realised_removal"`, or `"fixed_ceiling"`), `f_n_tolerance` (default 1.2, applied to the uptake-based caps), `fixed_ceiling_kg_ha` (default 170, EU Nitrates) and `disposal_method` (`"over_apply_local"` default, `"unmanaged_disposal"`, or `"retain_unallocated"`).

Value

A tibble with one row per allocation target: `year`, `territory`, `sub_territory`, `land_use` (`"Cropland"`/`"Grassland"`/`"Disposal"`/ `"Unallocated"`), `crop`, `source_stream` (`"collected"`/`"grazing"`), `manure_type` (`"Excreta"`/`"Solid"`/`"Liquid"`, carried from `applied` when present; each allocated row's N/C/VS is split across the `manure_types` in proportion to their share of the cell's collected N, so the capacity-filling math itself stays pooled), `applied_n`, `applied_c`, `applied_vs`, `over_cap` and the `method_allocation`, `method_cap` and `disposal_method` provenance columns. The N split by `manure_type` is exact (mass-conserving per type);

`applied_c` and `applied_vs` are split by the SAME N-proportional share, not by each `manure_type`'s own C:N/VS:N ratio, so a per-`manure_type` carbon or volatile-solids figure does not reflect that type's actual composition (e.g. solid vs. liquid manure have materially different C:N). Callers needing manure-type-specific carbon/VS (e.g. CH4 potential by system) should not rely on the `manure_type`-broken-out `applied_c`/ `applied_vs` columns without accounting for this.

Examples

```
applied <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~stream, ~manure_type,
  ~applied_n, ~applied_c, ~applied_vs,
  2020L, "203", NA, "collected", "Solid", 80, 800, 40,
  2020L, "203", NA, "grazing", "Excreta", 20, 380, 12
)
crops <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~crop, ~manure_n_receptivity, ~crop_n_cap,
  2020L, "203", NA, "44", 6, 50,
  2020L, "203", NA, "15", 4, 40
)
allocate_manure_to_land(applied, list(crops = crops))
```

---

allocate_manure_transport

Spill surplus manure to neighbouring cells with spare capacity.

Description

Moves the manure nitrogen that a cell could not place locally (its surplus above the local cropland and grassland caps) to nearby same-polity cells that still have room, on the 0.5-degree grid. Each source offers its surplus to its single-ring (king-move) neighbours in proportion to their remaining room; a sink that is over-subscribed by several sources is filled only to its room and the rejected manure returns to the sources as residual. This is the resolution-robust, mass-conserving analogue of the room-weighted first-ring redistribution of an earlier regional historical reconstruction; cross-polity transport is not allowed. Carbon and volatile solids ride along with each nitrogen flow at the source cell's bundle ratio.

Usage

```
allocate_manure_transport(source_cells, sink_cells, options = list())
```

Arguments

`source_cells`	A tibble of cells exporting manure, keyed by `year`, `territory` (a stringified `area_code`, see `estimate_n_excretion()`) and `sub_territory` (a `"lon_lat"` cell id), with `surplus_n`, `surplus_c` and `surplus_vs` (t).
`sink_cells`	A tibble of cells with spare capacity, keyed by `year`, `territory` and `sub_territory`, with `room_n` (remaining cropland plus grassland N capacity, t).

`options` A named list. `n_rings` (default 1) is the neighbourhood radius in grid steps; only the single-ring kernel is shipped.

Value

A tibble with one row per landing site: `year`, `territory`, `sub_territory` (the sink cell for transported manure, the source cell for residual), `applied_n`, `applied_c`, `applied_vs`, `kind` (`"transported"` or `"residual"`) and `method_transport`. The `"residual"` rows are the un-transportable surplus handed back for local disposal.

Examples

```
source_cells <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~surplus_n, ~surplus_c, ~surplus_vs,
  2020L, "203", "1.5_40", 10, 90, 6
)
sink_cells <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~room_n,
  2020L, "203", "1_40", 4,
  2020L, "203", "2_40", 6
)
allocate_manure_transport(source_cells, sink_cells)
```

`amg_h_by_input_type` *AMG humification coefficient by carbon input type.*

Description

Lookup table giving the AMG humification coefficient `h` (the fraction of carbon inputs entering the active pool) as a function of the carbon input type. Rows are matched in ascending `match_order` against the lowercased input type using the `pattern` regular expression; the last row (input type `"default"`, with a missing pattern) is the fallthrough.

Usage

```
amg_h_by_input_type
```

Format

A tibble with columns:

match_order Integer matching priority (lower is tried first).

input_type Canonical input type label (e.g. `"green_manure"`, `"manure"`, `"residue"`, `"default"`).

pattern Regular expression matched against the lowercased input type; `NA` for the fallthrough row.

h Humification coefficient (fraction of input carbon stabilised).

Source

Saffih-Hdadi, K. & Mary, B. (2008). Modeling consequences of straw residues export on soil organic carbon. *Soil Biology and Biochemistry*, 40(3), 594-607. doi:10.1016/j.soilbio.2007.08.022.

Examples

```
amg_h_by_input_type
```

---

animals_codes *Animal codes and classifications*

Description

Maps live animal CBS items to their livestock classifications, process codes, and associated product items used in livestock modeling.

Usage

```
animals_codes
```

Format

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

- `Item_Code`: Numeric FAOSTAT item code for the live animal.
- `item_cbs`: Name of the CBS item (e.g., `"Cattle"`, `"Asses"`).
- `proc_code`: Short process code used internally (e.g., `"p092"`).
- `proc`: Descriptive process name (e.g., `"Asses"`).
- `item_cbs_code`: Numeric CBS item code (often equal to `Item_Code`).
- `Farm_class`: Broad farm classification grouping the animal. One of `"Cattle"`, `"Dairy_cows"`, `"Monogastric"`, `"Sheep_goats"`, `"Bees"`, `"Game"`.
- `Item_product`: Name of the primary product derived from this animal, if applicable (e.g., milk for dairy cows).
- `Item_Code_product`: Numeric FAOSTAT code for the associated product item.
- `Liv_prod_cat`: Livestock product category the animal belongs to.
- `Graniv_grazers`: Broad feeding behaviour classification. One of `"Grazers"`, `"Granivores"`, `"Bees"`, `"Game"`.
- `Livestock_name`: Internal livestock identifier used across datasets (e.g., `"Cattle"`, `"Dairy_cows"`, `"Asses"`).
- `Animal_class`: Fine-grained animal class, including production type distinctions (e.g., `"Broilers"`, `"Hens"`, `"Hogs"`, `"Dairy_cows"`).
- `Item_FAOmanure`: Name of the corresponding FAOSTAT manure management item.

- `Item_Code_FAOmanure`: Numeric code of the FAOSTAT manure management item.
- `Cat_Labour`: Labour category used in labour-related analyses. One of `"Cattle"`, `"Equines"`, `"Dairy_cows"`, `"Birds"`, `"Small_ruminants"`, `"Pigs"`, `"Bees"`.
- `Cat_FAO1`: Top-level FAO category. Currently always `"Animal"`.
- `item_bouwman`: Item name used in Bouwman et al. livestock datasets.

Source

Derived from FAOSTAT data and internal livestock classification work.

Examples

```
head(animals_codes)
```

---

apply_management_losses

Apply IPCC manure-management losses to the collected manure streams.

Description

Nets the nitrogen surviving manure management onto the field, applying the IPCC 2019 management-loss fractions to the collected/housed streams from `split_manure_management()`: `applied_n = n_stream * (1 - FracLossMS)` where `FracLossMS = FracGasMS + FracLeachMS + EF3 + FracN2MS`. The grazing (pasture/range/paddock) stream is deposited in situ and keeps its full nitrogen (its in-situ soil losses belong to the soil stage). Indirect N2O is reported as a labelled sub-flux of the already-removed volatilized and leached nitrogen (the same N is not removed twice). Carbon applied to the field is `c_stream * (1 - c_loss_fraction)`, the stream's carbon less what its management system mineralises in storage, and the volatile solids are scaled by the same ratio. The applied C:N is therefore a RESULT of that loss rather than an imposed cap. Loss fractions and their sources are in `inst/extdata/manure/manure_storage_c_loss.csv`: nothing for grazing or daily spread, which have no storage stage; 0.420 of initial carbon for solid storage and 0.424 for poultry litter (Pardo et al. 2015, doi:10.1111/gcb.12806); 0.110 (cattle) and 0.128 (pigs) for slurry, derived from Kupper et al. 2020 (doi:10.1016/j.agee.2020.106963) using this package's own 0.47 kg C per kg volatile solids.

Until whep#1006 this was a cap, `pmin(c_stream, applied_n * post-storage C:N)`, which produced a loss only as a side effect of holding the applied C:N down. That made the loss depend on the excreted composition rather than on the storage system: when excreted carbon moved, the reported loss for cattle solid storage fell from 40.3% to 6.9% with no coefficient changing.

Usage

```
apply_management_losses(split, options = list())
```

Arguments

`split`	A tibble from `split_manure_management()`, optionally with the `n_bedding` and `c_bedding` columns `add_manure_bedding()` adds.
`options`	A named list. `method` selects the loss method (`"ipcc_2019_tier2"`). `bedding_c_loss` selects how bedding carbon is treated in storage: `"same_as_excreta"` (default) or `"none"`; see the Bedding section. `indirect_n2o_source` selects the edition of indirect_n2o_ef the EF4 and EF5 of the indirect N2O are read from: `"ipcc_2019"` (default; EF5 0.011, Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26 of the 2019 Refinement) or `"ipcc_2006"` (EF5 0.0075, Table 11.3, p. 11.24 of the 2006 Guidelines, the value shipped before whep#1245). EF4 is 0.010 in both, so only the leached share of `n2o_indirect_n` moves, by a factor of 1.467; `applied_n` does not. `ef3_source` selects the edition of ipcc_manure_ef3 the direct N2O of each stored stream is read from: `"ipcc_2019"` (default; Vol 4, Ch 10, Table 10.21 (Updated) of the 2019 Refinement) or `"as_shipped"` (the mixed table read before whep#601). It moves `n2o_direct_n` and `n2_n`, and so `applied_n`, of the collected streams; the grazing stream carries no management N2O either way. `slurry_crust` sets the liquid-slurry crust share under the 2019 edition: `"half"` (default, EF3 0.0025), `"none"` (0) or `"all"` (0.005); see ipcc_manure_ef3. `method_direct_n2o` records the choice (e.g. `"ipcc_2019; slurry_crust_half"`).

Value

The input rows with `manure_type`, `applied_n`, `applied_c`, `applied_vs`, `n_volatilized`, `n_leached`, `n2o_direct_n`, `n2_n`, `n2o_indirect_n`, `c_lost`, `vs_destroyed`, `n_bedding`, `c_bedding`, `method_losses`, `method_bedding_c`, `method_indirect_n2o` and `method_direct_n2o`.

Bedding

When the rows carry `n_bedding` and `c_bedding` (from `add_manure_bedding()`), the straw bedded under housed animals is part of the manure that reaches the field, and the applied C:N is the C:N of the bedded farmyard manure rather than of the excreta alone. Without those columns nothing changes: they default to zero and every number is the excreta-only one.

Nitrogen follows IPCC 2019 Refinement Vol. 4 Ch. 10 Eq. 10.34 (p. 10.94) exactly: `NbeddingMS` sits **outside** the `(1 - FracLossMS)` term, because "mineralization of nitrogen compounds in beddings occurs more slowly compared to manure and the concentration of ammonia fraction in organic beddings is negligible", so "both volatilization and leaching losses during storage of bedding are assumed to be zero" (p. 10.93). Bedding nitrogen therefore raises `applied_n` one-for-one and raises none of the loss side-streams.

Carbon has no IPCC rule at all – manure CO2 is out of scope there – so `bedding_c_loss` selects it. `"same_as_excreta"` (default) applies the stream's own storage carbon-loss fraction to the bedding carbon too: the 0.420 for solid storage comes from Pardo et al. 2015 (doi:10.1111/gcb.12806, Table 2), a systematic review of whole manure heaps, and a heap in solid storage in practice already contains its litter, so the measured loss is a whole-heap loss. `"none"` keeps every gram of bedding carbon, mirroring the zero storage loss IPCC gives bedding nitrogen; it is the upper bound on applied carbon and raises the applied

carbon of a bedded solid-storage stream by `0.420 / (1 - 0.420) = 72%` of the bedding carbon relative to the default.

Volatile solids stay excreta-only under both, so the Tier 2 methane engine in `build_livestock_ghg_extension()` is untouched. IPCC 2019 Ch. 10 does ask for bedding to be combined with volatile solids when estimating manure methane; that is a separate change to a separate engine and is not made here.

Examples

```
excretion <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~livestock_category,
  ~n_excretion, ~c_excretion, ~vs_excretion,
  2020L, "203", NA, "Cattle_milk", 100, 1900, 60
)
apply_management_losses(split_manure_management(excretion))
```

---

assert_footprint_invariants

Assert that footprint conservation invariants hold.

Description

Build-time gate over a footprint result, intended for use in a pipeline or regression test. Aborts when the footprint over-traces any origin sector (embodied pressure exceeds the source, which should never happen) or when the global share of untraced pressure exceeds `max_rel_loss`. A regression that silently loses or inflates pressure then fails loudly instead of shipping.

Usage

```
assert_footprint_invariants(
  footprint,
  extensions,
  labels,
  x_vec,
  max_rel_loss = 0.05
)
```

Arguments

`footprint`	Footprint tibble from `compute_footprint()`, with `origin_area`, `origin_item` and `value` columns.
`extensions`	Numeric vector of environmental extensions per sector, as passed to `compute_footprint()`.
`labels`	Tibble with `area_code` and `item_cbs_code` mapping each sector to its meaning, as passed to `compute_footprint()`.
`x_vec`	Numeric vector of total output per sector.
`max_rel_loss`	Maximum tolerated global relative under-tracing (pressure that never reaches final demand). The engine's negative-zeroing and column capping cause a small amount, so this is non-zero by default.

Value

Invisibly, the one-row summary from `summarise_conservation()`. Called for its side effect of aborting on violation.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
y_mat <- matrix(c(85, 195), ncol = 1)
extensions <- c(50, 30)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(1L, 2L)
)
fp <- compute_footprint(
  x_vec = x_vec, y_mat = y_mat, extensions = extensions,
  labels = labels, z_mat = z_mat
)
assert_footprint_invariants(fp, extensions, labels, x_vec)
```

---

assert_parquet_integrity

Assert that a Parquet file is structurally sound.

Description

Build-time gate over a Parquet artifact. Aborts when `check_parquet_integrity()` reports any anomaly, so a corrupt multi-gigabyte cube fails loudly at the moment it is written instead of being read back truncated for months (whep#531).

Usage

```
assert_parquet_integrity(path, deep = FALSE)
```

Arguments

`path`	Path to a Parquet file.
`deep`	Whether to also read every row group and report the ones that fail to decode. Exact, but reads the entire file.

Value

Invisibly, `path`. Called for its side effect of aborting on violation.

Examples

```
path <- tempfile(fileext = ".parquet")
nanoparquet::write_parquet(data.frame(x = 1:10), path)
assert_parquet_integrity(path)
unlink(path)
```

assert_table_schema *Assert that a table conforms to a declarative schema.*

Description

Build-time gate over a tabular artifact. Aborts when `check_table_schema()` reports any `"error"` diagnostic and warns when it reports only `"warning"` ones, so a table that has silently lost a column, changed type or gained duplicate keys fails at the moment it is produced instead of downstream. Returns its input, so it can sit inside a pipeline without changing the value that flows through.

Usage

```
assert_table_schema(data, schema, arg = "data")
```

Arguments

`data`	Table to validate. Not modified.
`schema`	Declarative schema, as described above.
`arg`	Name of the validated object, used in messages.

Value

Invisibly, `data`, unchanged. Called for its side effect of aborting on violation.

Examples

```
schema <- list(
  columns = list(
    list(name = "year", type = "integer"),
    list(name = "value", type = "double", min = 0)
  ),
  key = "year"
)
data <- tibble::tibble(year = c(2000L, 2001L), value = c(1, 2))
assert_table_schema(data, schema)
```

attach_provenance *Attach a provenance record to a result.*

Description

Store a provenance record (from `record_provenance()`) on an object as an attribute, so the result travels together with its lineage. Retrieve it later with `get_provenance()`.

Usage

```
attach_provenance(x, provenance)
```

Arguments

x	Any R object, typically a result tibble.
provenance	Provenance tibble from record_provenance().

Value

x, unchanged except for an added whep_provenance attribute.

Examples

```
prov <- record_provenance(
  aliases = "bilateral_trade",
  recorded_at = as.POSIXct("2026-01-01", tz = "UTC")
)
result <- attach_provenance(tibble::tibble(value = 1), prov)
get_provenance(result)
```

---

attach_scope *Attach a scope record to a result.*

Description

Store a scope record (from footprint_scope()) on an object as an attribute, so the result carries its goal-and-scope with it.

Usage

```
attach_scope(x, scope)
```

Arguments

x	Any R object, typically a footprint tibble.
scope	Scope tibble from footprint_scope().

Value

x, unchanged except for an added whep_scope attribute.

Examples

```
scope <- footprint_scope("cropland", "ha", "FABIO-MRIO")
result <- attach_scope(tibble::tibble(value = 1), scope)
get_scope(result)
```

`attribute_fallow_to_crops` *Attribute reported fallow land to crops.*

Description

Distribute each country's FAOSTAT-reported fallow area among its crops using a precomputed allocation weight, adding the result to each crop's cropped physical area. The weight is typically `gridded_fallow_weights()`, which puts fallow on rainfed dryland cereals and rainfed monsoon rice and keeps it off irrigated/continuous systems.

The fallow *magnitude* comes from `fallow_total` (FAOSTAT "Land with temporary fallow", item 6640) — reported separately from temporary meadows/pastures, so it isolates real fallow from fodder.

Usage

```
attribute_fallow_to_crops(cropgrids, fallow_total, alloc_weight)
```

Arguments

`cropgrids`	Tibble of national crop areas with columns `area_code`, `item_cbs_code`, `physical_ha` (cropped physical area), `harvested_ha`.
`fallow_total`	Tibble of reported fallow area with columns `area_code` and `fallow_ha`.
`alloc_weight`	Tibble of `area_code`, `item_cbs_code`, `weight` giving the within-country allocation weight, e.g. from `gridded_fallow_weights()`.

Value

A tibble with `area_code`, `item_cbs_code`, `physical_ha` (cropped physical area plus attributed fallow), and `harvested_ha`.

Examples

```
cropgrids <- tibble::tribble(
  ~area_code, ~item_cbs_code, ~physical_ha, ~harvested_ha,
  1L, 2511L, 500, 500,
  1L, 2807L, 400, 400
)
fallow_total <- tibble::tribble(~area_code, ~fallow_ha, 1L, 200)
# all weight on wheat -> the 200 ha reported fallow goes to wheat
alloc_weight <- tibble::tribble(
  ~area_code, ~item_cbs_code, ~weight,
  1L, 2511L, 1,
  1L, 2807L, 0
)
attribute_fallow_to_crops(cropgrids, fallow_total, alloc_weight)
```

`backcast_admin_shares` *Back-cast admin shares across the seam*

Description

Complete an admin-shares table to the full year range of its extent and fill the pre-seam years by modulating each unit's seam-year share with its own land extent, the plan's decision-7 back-cast:

$$\tilde{s}_u(t) = s_u(t_0)\frac{E_u(t)}{E_u(t_0)}, \qquad s_u(t) = \frac{\tilde{s}_u(t)}{\sum_v \tilde{s}_v(t)}$$

The first step is `fill_proxy_growth()` with the extent as proxy, the same machinery `.fill_pre_faostat()` (`R/build_production.R:3008`) uses to back-cast the national totals these shares split, so the two compose. The second is a per-year renormalisation over the `t0` unit set, the residual pseudo-unit included when the shares carry one.

Nothing here is a fallback: a year the rules cannot resolve is refused with a diagnostic and keeps `share = NA`, never a carried value.

Two things the plan's seam section names are deliberately **not** here. Pre-seam irrigation needs no separate treatment: the unit irrigated target is built from the engine's own annual `ir_potential` (`R/spatialize.R:420`), whose LUH2 `irrigated_ha` varies every year, pre-seam years included, so it moves with the same extent that modulates the area share. And the within-unit weight *regime* per `(unit, item, year)` is the allocation's to record, not this function's: `treatment` here says how a share was obtained, while the regime says which weight then spread it inside the unit.

Usage

```
backcast_admin_shares(
  shares,
  extent,
  seam = NULL,
  binding_indicator = "area_harvested",
  settings = list()
)
```

Arguments

`shares`	An admin-shares table (see `admin_shares_schema()`). Minimum columns: `area_code`, `level`, `item_prod_code`, `indicator_used`, `level_polity_code`, `year`, `share`, `treatment_year`. Every other contract column present is carried onto the produced rows from their anchor row.
`extent`	A per-unit extent table from `aggregate_unit_extent()`. Its year range is the range the shares are completed to.

`seam` Optional tibble overriding `t0` per series: `area_code`, `item_prod_code`, `t0`, optionally `level`. An override is the temporal hold-out lever: every year below the given `t0` becomes a back-cast year even where an observation exists, and the superseded observations are counted (`seam_supersedes_observed`).

`binding_indicator`
The indicator that binds, `"area_harvested"` by the plan's decision 8. `t0` is the series' first observed year carrying it; a series with none is refused (`no_binding_anchor`).

`settings` Named list of tuning knobs, each defaulting as below. An unknown key aborts, and what the call ran with comes back in the returned `settings` tibble:

- `max_gap` (`Inf`): passed to `fill_proxy_growth()`, its own default.
- `max_gap_linear` (`0`): passed to `fill_proxy_growth()`; a placeholder pending T31(e), see the interior-gaps section.
- `zero_policy` (`"hold_outside"`): how case (a) is handled. Only `"hold_outside"` is signed off (decision 9); the key exists so a later decision can add a value without changing call sites.
- `repair` (`FALSE`): on a flagged extent jump, `TRUE` applies the unit-keyed analogue of `.fix_luh2_crop_collapse()` and refuses only what stays flagged; `FALSE`, pending T31, refuses the affected series outright. Either way the choice is in `settings` and the outcome in `diagnostics`.
- `ratio_bounds` (`c(0.55, 1.6)`): plausible band for `check_extent_jumps()`.
- `collapse_ratio` (`0.02`): fraction of the adjacent-year mean below which a year counts as an isolated collapse, as in `.fix_luh2_crop_collapse()` (`R/build_production.R:629-630`).
- `min_neighbour_ha` (`100`): both neighbours must exceed this for a collapse to be repaired – that function's `min_neighbor_mha = 0.001` Mha expressed in hectares.
- `tolerance` (`1e-9`): absolute tolerance for the share and extent arithmetic.

Value

A list of four tibbles:

- `shares`: the input rows plus the completed year set for the `t0` unit sets, with two added columns. `treatment` is `"observed"` for a row that arrived in `shares`, `"backcast_t0_geometry"` for a row this function produced, `"luh2_clamped"` where `t0` lies beyond the extent's last year so `E_u(t0)` is the clamped last slice, and `NA` for a completed row nothing filled, whose reason is in `diagnostics`. `gap_rule` records the interior-gap setting. A produced row carries its anchor's provenance columns, `value = NA` (no reported value exists) and `treatment_year = NA` (reserved for the interior-gap rule), so produced rows deliberately sit outside the closed `admin_shares_schema()`; the `"observed"` subset on the contract columns still conforms.
- `diagnostics`: one row per event, with the series and unit keys, `year` where the event is year-specific, `diagnostic` from a closed vocabulary and free-text `detail`.

- `counters`: every vocabulary member with its count, zeros included.
- `settings`: the one-row record of what this call was run with.

Territory basis

Shares and `E_u` are defined on the `t0` unit set with `t0` geometry held fixed, the `read_luh2_landuse()` snapshot convention and the analogue of `mapping_status == "backcast_anchor"`. Every row produced that way carries `treatment = "backcast_t0_geometry"`, so a unit whose containment edge starts after a back-cast year still gets a row at that year, on its `t0` cells.

Translating those rows onto the partition valid at `t` through the containment-edge lineage – an earlier unit that later split taking the sum of its `t0`-descendants' modulated shares – happens before the rows enter the engine and is **not implemented here**. It is a follow-up, because the relation it needs is succession (which unit became which), while `polity_containment` carries containment (`member_code`, `container_code`, `start_year`, `end_year`, `basis`) only: nothing in the delivered data says a unit starting in one year descends from one that ended the year before. Until it lands, a container whose partition changed inside the back-cast window is back-cast on its `t0` partition, which `treatment` states on every row.

Zero cases

Post-processing applied after the fill, which itself leaves such positions `NA` (`.fg_fill_backward_vec()` keeps a fill only where the result is finite and positive). The plan's four cases, with precedence (a) over (b):

- (a) `E_u(t0) == 0` with `s_u(t0) > 0`: hold `s_u(t0)` **outside** the renormalisation, siblings renormalised to `1 - s_u(t0)`. Holding it inside would anti-modulate the unit. Counter `zero_extent_t0_held`.
- (b) `E_u(t) == 0` at a back-cast year of a unit not held by (a): share `0` there, its mass going to the siblings through the renormalisation. Counter `zero_extent_year_zeroed`.
- (c) the renormalisable sum is zero while there is mass to distribute: the year is refused for that series and all its units left `NA`. Counter `zero_sum_year_refused`.
- (d) `E_u(t0) == 0` with no positive `s_u(t0)`: nothing to hold and no mass to move, so the unit contributes `0`. Counter `zero_extent_t0_no_share`.

Case (b) counts every zero-extent back-cast year, including the ones inside a year (c) then refuses: both statements are true of such a year, and (c) is what decides the output.

A zero year also breaks the growth chain: `fill_proxy_growth()` telescopes a product of consecutive growth factors, so it cannot reach the years beyond the break and leaves them `NA` even though the ratio above is perfectly well defined there. Those positions are completed from that ratio directly – the same expression, not a second method – and every one is counted (`direct_ratio_repair`). Where the chain is intact the two agree to machine precision, which the tests assert.

Interior gaps

The fill window ends at `t0`, so no gap between two observed years is ever filled: such rows are completed, left `NA` and counted (`interior_gap_refused`, `trailing_gap_refused`). That is the placeholder default until T31(e) decides the rule; it is recorded on every

row in `gap_rule` and in `settings`. `max_gap` and `max_gap_linear` are passed straight to `fill_proxy_growth()` so that decision needs no API change. Today `max_gap` bounds the back-cast run itself – a longer run is left unfilled and counted (`max_gap_exceeded`), because `fill_proxy_growth()`'s gap test governs the leading run as well as the interior ones – while `max_gap_linear` cannot bite, there being no interior run inside the window.

Examples

```
shares <- tibble::tibble(
  area_code = 900L,
  level = 1L,
  item_prod_code = 15L,
  indicator_used = "area_harvested",
  level_polity_code = c("A1", "A2"),
  year = 1902L,
  share = c(0.6, 0.4),
  treatment_year = "observed"
)
extent <- tibble::tibble(
  area_code = 900L,
  level_polity_code = rep(c("A1", "A2"), each = 3),
  level = 1L,
  year = rep(1900:1902, times = 2),
  extent_ha = c(800, 900, 1000, 600, 550, 500)
)
out <- backcast_admin_shares(shares, extent)
out$shares[, c("level_polity_code", "year", "share", "treatment")]
```

`balance_io_flows` *Balance input-output flows so the footprint conserves.*

Description

RAS-balance an inter-industry flow matrix toward the product-balance margin. The row target is intermediate use, $u = X - \text{rowSums}(Y)$; the column target keeps the original column composition, rescaled so its total matches $\sum u$.

When the balanced system is productive (the spectral radius of A is below 1), $(I - A)X = Y$ holds and the footprint conserves – the case for the productive systems in the examples and tests.

Caveat for physical agriculture: livestock feed conversion makes some columns of A sum well above 1 (many tonnes of feed per tonne of product), so the balanced system is not productive. Balancing to faithful margins does not remove that, and the traced footprint then over-traces rather than conserving (on the real 2010 model, by about 15%). RAS therefore does not, on its own, fix the grassland under-tracing; always verify with `check_footprint_conservation()`.

Pass the result as `z_mat` to `compute_footprint()` with a large `max_column_sum` (e.g. `1e12`) and `conserve_extensions = FALSE`.

Usage

```
balance_io_flows(z_mat, x_vec, y_mat, max_iter = 1000L, tol = 1e-06)
```

Arguments

`z_mat`	Inter-industry flow matrix (dense or `Matrix`).
`x_vec`	Total output per sector.
`y_mat`	Final demand vector, or matrix whose row sums give total final demand per product.
`max_iter`, `tol`	Passed to `balance_ras()`.

Value

The balanced flow matrix.

Examples

```
z_mat <- matrix(c(1, 4, 2, 3), nrow = 2)
x_vec <- c(10, 10)
y_mat <- c(3, 3)
balance_io_flows(z_mat, x_vec, y_mat)
```

`balance_ras` *Balance a matrix to target margins by RAS.*

Description

Reconcile a non-negative matrix to prescribed row and column sums using the RAS (biproportional / iterative proportional fitting) algorithm: alternately rescale rows then columns until both margins are matched, preserving the matrix's structure (zero pattern and relative magnitudes). The row and column targets must have equal totals.

Works on dense and sparse (`Matrix`) inputs, staying in the input representation so it is efficient for both small dense tables and large sparse input-output systems. (The same core also balances the bilateral trade matrices.) For matrices with negative entries the signed GRAS variant is needed and is not implemented here.

Usage

```
balance_ras(x, target_rows, target_cols, max_iter = 1000L, tol = 1e-09)
```

Arguments

`x`	Non-negative matrix (dense or `Matrix`).
`target_rows`	Desired row sums, length `nrow(x)`.
`target_cols`	Desired column sums, length `ncol(x)`.
`max_iter`	Maximum RAS iterations.
`tol`	Convergence tolerance on the largest margin deviation, relative to the margin total (so it is scale-free).

Value

The balanced matrix, in the same representation as `x` (dense in, dense out; sparse in, sparse out).

Examples

```
m <- matrix(c(1, 2, 3, 4), nrow = 2)
balance_ras(m, target_rows = c(10, 20), target_cols = c(12, 18))
```

---

biomass_coefs *Biomass coefficients for crops and livestock products*

Description

Provides dry-matter, nutrient, and energy conversion coefficients for agricultural products and residues. Used to convert fresh-matter production quantities into biomass flows, nutrient budgets, and energy content.

Usage

```
biomass_coefs
```

Format

A tibble where each row corresponds to one product or item. It contains 63 columns:

- `Code`: Item code (character), corresponding to FAOSTAT production codes.
- `Name_biomass`: Item name as used in biomass accounting.
- `Equiv`: Reference equivalence item used when coefficients are borrowed from another similar commodity (e.g., `"Wheat"` for oats).
- `Category`: Broad commodity category (e.g., `"Cereals, other"`, `"Barley"`, `"Vegetables"`).
- `Product_kgDM_kgFM`: Product dry-matter content in kg DM per kg fresh matter.
- `Residue_kgDM_kgFM`: Residue dry-matter content in kg DM per kg fresh matter of product.
- `Conventional_kgDM_ha`: Conventional yield in kg dry matter per hectare.
- `Organic_kgDM_ha`: Organic yield in kg dry matter per hectare.
- `GE_product_edible_portion_MJ_kgFM`: Gross energy of the edible portion in MJ per kg fresh matter.
- `GE_product_residue_MJ_kgFM`: Gross energy of the residue in MJ per kg fresh matter (may be character due to source formatting).
- `GE_product_MJ_kgFM`: Gross energy of the whole product in MJ per kg fresh matter.
- `GE_residue_MJ_kg`: Gross energy of the residue in MJ per kg.
- `kg_product_kg_aerial_biomass`: Fraction of aerial biomass that is product (harvest index, kg/kg).

- `kg_residue_kg_aerial_biomass_FM`: Fraction of aerial biomass that is residue, on fresh matter basis.
- `kg_residue_kg_product_FM`: Ratio of residue to product on fresh matter basis.
- `Carcass_to_LW`: Carcass-to-live-weight ratio (livestock only; logical placeholder for crop items).
- `Edible_portion`: Edible fraction of the product (kg edible / kg fresh matter).
- `N_kgN_kgFM`: Nitrogen content in kg N per kg fresh matter.
- `Lipids_g_kgFM`: Lipid content in g per kg fresh matter.
- `Carbohydrates_g_kgFM`: Carbohydrate content in g per kg fresh matter.
- `Calcium_mg_kgFM`: Calcium content in mg per kg fresh matter.
- `VitaminA_microg_kgFM`: Vitamin A content in micrograms per kg fresh matter.

`N_kgN_kgFM` is the food-composition nitrogen density of the edible part, and is what `build_food_supply()` turns into protein. Its provenance is weak and worth knowing before trusting a row. Upstream it sits under the `NUTRIENTS IN EDIBLE PART` group header with the four columns above, but the workbook's `Sources` sheet has **no column for any of the five**: its columns run from `kg_residue_kg_product_FM` straight to `Product_kgN_kgDM`, so the whole nutrition block is undocumented at source (#500 section 6, #1074). Most rows are a `VLOOKUP` into the workbook's food-composition sheet `Conversores_Dieta`, the same sheet that fills the four columns above; that sheet derives its own nitrogen column as `Proteinas / (6.25 * 1000)`. The derivation is missing for exactly its five **cereal** rows – `Wheat`, `Oats`, `Rye`, `Maize` and `Rice` – which carry a `Proteinas` value but no nitrogen, so for those five the lookup was replaced by the agronomic `Product_kgN_kgDM * Product_kgDM_kgFM`. Other rows, `Vegetables, other`, `Olive` and `Honey` among them, hold a bare literal. Those eight are the ones #1096 tracks; what became of each is below.

`Wheat` was the largest consequence. It carried 0.018951324393104405 kg N per kg, 118.45 g of protein per kg at N x 6.25, which is exactly `Product_kgN_kgDM * Product_kgDM_kgFM` – the agronomic whole-grain nitrogen the `Sources` sheet attributes to FEDNA 2016, a *feed* table. Applied to a commodity-balance `food` quantity that is the wheat-grain equivalent of the milled products people eat, that counts the bran and germ protein milling diverts away from food.

It now carries **0.01488 kg N per kg, i.e. 93 g of protein per kg: the flour basis**. The value is read from the workbook cell, not recalled: `Biomass_coefs.xlsx`, sheet `Conversores_Dieta`, row 4, labelled `Harina de Trigo` (wheat flour) and keyed to `Name_biomass = "Wheat"`, column `Proteinas` = 93 g/kg. That row is where this table already takes the wheat lipids (12 g/kg), carbohydrates (800 g/kg), calcium (150 mg/kg) and vitamin A (0) from – all four ship today – so the wheat nutrition block was flour composition everywhere except protein, and the change makes the row internally consistent rather than importing a foreign number.

Neither 118.45 nor 93 has a `Sources` entry, so 93 is corroborated, not cited upstream. What "flour basis" means numerically is protein per kg of *grain equivalent*, which is flour protein times the milling extraction rate, so the published tables were read on that basis:

- FAO, *Food composition tables for international use*, 2nd ed., Rome, 1953, https://www.fao.org/4/x5557e/x5557e04.htm (FAO flags it as historical). Medium wheat: whole meal 12.2 g/100 g at 100% extraction, flour 11.7 at 85%, 10.9 at 72% – so 99.4 and 78.5 g per kg of grain equivalent, which bracket 93.

- USDA FoodData Central 168894 (SR Legacy), "Wheat flour, white, all-purpose, enriched, bleached": protein 10.33 g/100 g, so 103.3 g per kg of flour, 72-88 g per kg of grain equivalent at 70-85% extraction.
- FAO, *Technical conversion factors for agricultural commodities*, Rome, https://www.fao.org/fileadmin/templates/ess/documents/methodology/tcf.pdf – its per-country `Flour of Wheat` extraction rates run 70-97% with a median of 75%, which is why no single flour figure maps to one grain-equivalent density.

The oracle is FAOSTAT FBS itself, which builds its protein the same way: composition factors on the products eaten, over the standardised primary-equivalent quantity that WHEP's `food_t` also is (FAO, *Food balance sheets: a handbook*, section III, https://www.fao.org/4/X9892E/X9892e03.htm). For item 2511 its **World row (area 5000)** implies 91.71-93.15 g/kg over 2010-2023, 92.99 in 2010; summed instead over the 171 countries reporting both elements it is 94.96-96.90, 96.60 in 2010. So 93 lands within 0.02% of the World row and 3.7% below the country sum, where 118.45 was above both: 27.4% above the World row and 22.6% above the country sum (whep#796).

One global coefficient cannot carry the extraction pattern country by country: FBS's own per-country implied density for 2511 in 2010 spans 41-115 g/kg (q10 69, median 81, q90 90), so 93 is the mass-weighted world point of that distribution and leaves the median country high. The previous value stays selectable as `build_food_supply(protein_basis = "product_nitrogen")`, which reads the agronomic route directly and returns exactly 0.11844577745690253 kg protein per kg; the choice is recorded per row in `method_protein_basis`.

The other seven rows were settled one by one (#1096), by the same test as wheat: does the row's own `Conversores_Dieta` entry, the one that already fills its other nutrients, carry a protein value, and does following it agree with the FBS oracle? The oracle ratios below are WHEP world food protein over the FBS World row (area 5000), 2010, on a real build.

- `Olive` now carries **0.00128** (8 g protein/kg), from row 39 `Aceituna de mesa` (table olive), whose lipids 200, carbohydrates 10, calcium 630, vitamin A 220 and edible portion 0.8 are the values already shipped. The bare literal 0.0038 it replaces put olive protein at 3.23x FBS; at 0.00128 it is 1.09x.
- `Rye` now carries **0.016** (100 g/kg), from row 6 `Harina de Centeno` (rye flour), whose lipids 19.3, carbohydrates 597 and calcium 270 are already shipped – the wheat case exactly, flour basis. The agronomic 0.01367 it replaces was 0.90x FBS; 0.016 is 1.06x.
- `Honey` now carries **0.0008** (5 g/kg), carbohydrates **780 g/kg** and calcium **50 mg/kg**, from row 36 `Miel`. Its block had been a hand-typed placeholder (protein 0, carbohydrate 1000 g/kg, calcium 0) that matched no row and put 1000 g of carbohydrate in 785 g of dry matter; protein and carbohydrate from `Miel` sum to exactly that dry matter. At 5 g/kg honey is 1.69x FBS, where it was zero. Only the protein moves a computed number: nothing in the package reads the carbohydrate or calcium columns.
- `Oats` and `Maize` are **unchanged**, on the agronomic value. Their rows, 5 `Avena grano` 117 g/kg and 7 `Harina de Maiz` 87 g/kg, move *away* from the FBS density (72.6 and 62.2 g/kg, country sum) where the agronomic 83.8 and 78.0 are closer, so following the row would be consistency at the cost of accuracy. Open for an expert.
- `Rice` is **unchanged**: its basis was settled in #751/#755.
- `Vegetables, other` is **unchanged** and moves no food number: no `item_cbs_code` bridges to it (`Vegetables, Other` 2605 reaches `Artichoke` through `items_full`). Its row 113 would give 33.3 g/kg against the literal's 14.0, and neither is checked.

All three new values are the workbook's own composition figures, which, like the rest of this block, have no `Sources` entry: they are corroborated by the FBS oracle, not cited upstream. The workbook copy read was `afsetools/inst/extdata/Biomass_coefs.xlsx`; upstream still carries the old values in `Coefs` and should follow.

The tree-nut rows `Almonds`, `Walnut` and `Hazelnut` pair a **kernel** nitrogen density with an `Edible_portion` that now refers to the nut **in shell** (whep#797). FBS item 2551 `Nuts and products` is a quantity in shell, and it reaches `Almonds` through items_full; with an `Edible_portion` of 1 the shell was counted as food, 200 g of protein per kg against an FBS density of 46.6, 4.3x. The edible fraction is FAO's own food-composition factor for the FBS, in-shell protein over shelled protein per 100 g (FAO, *Food balance sheets: a handbook*, Rome, 2001, Annex I, https://www.fao.org/docrep/pdf/011/x9892e/x9892e00.pdf, PDF page 65): almonds 8.0 / 20.0 = **0.4**, walnuts 6.4 / 14.3 = **0.4476**, hazelnuts 6.0 / 13.0 = **0.4615**. The shelled figures are this table's own kernel densities to within 8% (20.0, 14.0, 14.1 g/100 g). FAO's *Technical conversion factors for agricultural commodities* (https://www.fao.org/fileadmin/templates/ess/documents/methodology/tcf.pdf) corroborates the range: its per-country shelled-from-unshelled extraction rates run 30-65% for almonds (median 35), 35-70% for walnuts (median 40) and 40-50% for hazelnuts (median 50). Nuts now carry 80 g of protein per kg, 1.7x FBS. What remains is the basket: 2551 covers ten species and FAO's in-shell factors for them run from 1.8 (chestnuts) to 10.3 (pistachios), so one representative species cannot match it (whep#1453). Only the protein moves: the energy columns are not scaled by `Edible_portion`.

The ten `Edible_*` and `NonEdible_*` nutrient columns below are **empty in every row**, upstream in the source workbook as well as here, so no edible/non-edible nutrient split can be read from them (#361). Use `Edible_portion` with `N_kgN_kgFM` or `Product_kgN_kgDM` to derive an edible basis instead, as `build_food_supply()` does.

- `Edible_kgDM_kgFM`: Edible dry matter in kg per kg fresh matter. Empty.
- `Edible_kgC_kgFM`: Edible carbon in kg C per kg fresh matter. Empty.
- `Edible_N_kgFM`: Edible nitrogen in kg N per kg fresh matter. Empty.
- `Edible_kgP_kgFM`: Edible phosphorus in kg P per kg fresh matter. Empty.
- `Edible_K_kgFM`: Edible potassium in kg K per kg fresh matter. Empty.
- `NonEdible_kgDM_kgFM`: Non-edible dry matter, kg per kg fresh matter. Empty.
- `NonEdible_kgC_kgFM`: Non-edible carbon in kg C per kg fresh matter. Empty.
- `NonEdible_kgN_kgFM`: Non-edible nitrogen in kg N per kg fresh matter. Empty.
- `NonEdible_kgP_kgFM`: Non-edible phosphorus, kg P per kg fresh matter. Empty.
- `NonEdible_kgK_kgFM`: Non-edible potassium, kg K per kg fresh matter. Empty.
- `Product_kgN_kgDM`: Nitrogen content of product in kg N per kg dry matter.

The synthetic feed-additive rows take this column from the FEDNA feed tables as crude protein divided by 6.25, the same way `Threonine`, `Tryptophan`, `Valine` and `Urea` still do. `Methionine` carried 0.1143 instead, which is 21.7% above the most nitrogen the molecule can hold: methionine is C5H11NO2S with one nitrogen atom, so its mass fraction is 14.007 over 149.208, or 0.0939 kg N per kg. It is now FEDNA's DL-Metionina entry, 58.5% crude protein and better than 99% purity, giving 0.0936 (whep#931; https://fundacionfedna.org/ingredientes-para-piensos). The alternative is the other commercial methionine source, the hydroxy analogue FEDNA lists as HIDROXI-ANAL MET, which is C5H10O3S

and holds no nitrogen at all; it is the value the retired pin carried. That `items_full$FEDNA` names HIDROXI-ANAL MET for this item is not evidence for the analogue: that column is a feed-table stand-in, not a product identity (see items_full, whep#1131).

The upstream workbook records no source for the 28 feed-additive rows, so their provenance is kept in `inst/extdata/harmonization/biomass_coefs_additive_sources.csv`, one row per additive with the entry it was read from in the FEDNA 2019 tables (`Tablas FEDNA 2019.xlsx`, v. 15.01.2021, linked from https://fundacionfedna.org/ingredientes-para-piensos) or an explicit "assumed, unverified" status (whep#1074). `Lysine` is FEDNA's L-Lisina HCl, 94.4% crude protein, 0.1510; it carried 0.2015, above even the free base's 0.1916. `Ammonium chloride` is FEDNA's Cloruro amonico, 163.6% crude protein, 0.2618. `Choline chloride` is FEDNA's Cloruro colina 60, whose 52% choline gives 0.0699 through the choline cation's nitrogen fraction, 14.007 over 104.173; FEDNA books its crude protein as zero because choline nitrogen is not protein, not because it is absent. Both carried 0, so their nitrogen was missing from the balance. `Phytase` and `Enzimes` keep an unsourced 0.116, now marked as assumed; their fresh-matter `N_kgN_kgFM` of 0.16 now follows this column.

- `Product_kgP_kgDM`: Phosphorus content of product in kg P per kg dry matter.
- `Product_kgK_kgDM`: Potassium content of product in kg K per kg dry matter.
- `Product_kgC_kgDM`: Carbon content of product in kg C per kg dry matter.
- `Residue_kgN_kgDM`: Nitrogen content of residue in kg N per kg dry matter.

For the wood and forest rows this column prices two physically different quantities through one cell. `create_n_prov_destiny()` sends the harvested `Wood` item to `Average wood`, and also relabels the residue production of forest and shrubland land as `Firewood`, which resolves to the same row. Harvested wood is stemwood; forest residue is branches, bark and foliage.

The shipped value is assumed, unverified. No value the upstream workbook has carried for these rows has a citation that can be opened: the beech anchor's source cell begins `ChatGPT.`, the conifer anchor's two citations do not resolve as written, and the holm-oak anchor's cell is blank (whep#932). The anchors can still be placed against an independent compilation. Thurner et al. (2025) pool 1048 stem, 599 branch, 267 root and 5944 leaf nitrogen measurements for boreal and temperate trees – the woody ones almost entirely from 192 literature studies – and report medians, in kg N per kg dry matter, of 0.0010 in stem sapwood, 0.0035 in branches, 0.0060 in roots and 0.0167 in leaves. Against those, `Average wood`'s 0.0030 is 0.86 of the branch median and 3.0 times the stem median – a branch concentration – while the retired pin's 0.00095 is 0.95 of the stem median, a stemwood one.

So the gap whep#932 records is not two estimates of one quantity in disagreement; it is the 3.5-fold gap between stem and branch tissue, and whichever single number the cell carries is wrong by about that factor for one of its two uses. The same ordering appears in the trembling-aspen budget of Morrison and Foster (1979, reproduced in Hacker 2005, "Effects of Logging Residue Removal on Forest Sites"), where stemwood holds 84 kg N in 119 t of dry matter, 0.00071 kg N per kg, against 0.0042 for bark, 0.0049 for branches and 0.024 for foliage. So this cell is left as published and `create_n_prov_destiny()` does not read it for wood by default: with `wood_n_method = "tissue"` it prices `Wood` at the stem median 0.0010 and `Firewood` at the branch median 0.0035 (whep#932); `"single"` reads this cell for both. `tests/testthat/test_biomass_coefs_wood_provenance.R` holds the comparison and fails if either side of it drifts.

Thurner, M., Yu, K., Manzoni, S., Prokushkin, A., Thurner, M. A., Wang, Z., and Hickler, T. (2025). Nitrogen concentrations in boreal and temperate tree tissues vary with tree age/size, growth rate, and climate. *Biogeosciences* 22(5), 1475-1493. doi:10.5194/bg22-14752025.

- `Residue_kgP_kgDM`: Phosphorus content of residue in kg P per kg dry matter.
- `Residue_kgK_kgDM`: Potassium content of residue in kg K per kg dry matter.
- `Residue_kgC_kgDM`: Carbon content of residue in kg C per kg dry matter.
- `Residue_humified_kgC_kgC`: Humification coefficient of residue carbon (fraction of residue C stabilised as soil organic matter).
- `MgDM_m3`: Megagrams dry matter per cubic metre (bulk density proxy).
- `Root_humified_kgC_kgC`: Humification coefficient for root carbon.
- `Root_mass_kgC_kgDM`: Root carbon mass in kg C per kg crop dry matter.
- `Residue_C_N`: Carbon-to-nitrogen ratio of the residue.
- `Root_kgN_kgDM`: Nitrogen content of roots in kg N per kg root dry matter.
- `GE_Roots_MJ_kgDM`: Gross energy of roots in MJ per kg dry matter.
- `Fiber_g_kgFM`: Dietary fibre content in g per kg fresh matter.
- `SFA_g_kgFM`: Saturated fatty acid content in g per kg fresh matter.
- `MUFA_g_kgFM`: Monounsaturated fatty acid content in g per kg fresh matter.
- `PUFA_g_kgFM`: Polyunsaturated fatty acid content in g per kg fresh matter.
- `PUFA_n3_g_kgFM`: Omega-3 PUFA content in g per kg fresh matter.
- `Iron_mg_kgFM`: Iron content in mg per kg fresh matter.
- `Zinc_mg_kgFM`: Zinc content in mg per kg fresh matter.
- `Magnesium_mg_kgFM`: Magnesium content in mg per kg fresh matter.
- `Cadmium_microg_kgFM`: Cadmium content in micrograms per kg fresh matter.
- `VitaminB12_microg_kgFM`: Vitamin B12 content in micrograms per kg fresh matter.
- `VitaminD_microg_kgFM`: Vitamin D content in micrograms per kg fresh matter.
- `Folate_microg_kgFM`: Folate content in micrograms per kg fresh matter.
- `VitaminC_mg_kgFM`: Vitamin C content in mg per kg fresh matter.
- `VitaminE_mg_kgFM`: Vitamin E content in mg per kg fresh matter.
- `Flavonoids_mg_kgFM`: Flavonoid content in mg per kg fresh matter.
- `Carotenoids_mg_kgFM`: Carotenoid content in mg per kg fresh matter.

Details

This is the **single** source of biomass coefficients in the package. Until #489 a `biomass_coefs` pin frozen at `20250728T082553Z` was also readable through `whep_read_file()`; it was a narrowed 2025 export that disagreed with this table on 12 of their 36 shared columns, so the same commodity carried different nitrogen coefficients depending on which path read it. The pin has been retired and all callers now read this dataset.

Five runtime-dead below-ground fields were retired from this legacy table: `BG_Biomass_kgDM_ha`, `Root_Shoot_ratio`, `Root_kgC_kgDM`, `Rhizodeposits_mass_kgC_kgDM`, and `Rhizodeposits_N_kgN_kgRootN`. Their related item-keyed fields in `bio_coefs` are, respectively, `bg_biomass_dm_kg_ha`, `root_shoot_ratio`, `root_c_kgdm`, `rhizodeposit_mass_c_kgdm`, and `rhizodeposit_n_kgn_krootn`. This is not a universal one-to-one row mapping. `calculate_crop_roots()` uses `ipcc_root_coefs$bg_ref_dm_t_h` and `ipcc_root_coefs$rs_default` when they are available, with the corresponding `bio_coefs` fields as fallbacks. `root_c_kgdm` and `rhizodeposit_n_kgn_krootn` are direct calculation inputs. `rhizodeposit_mass_c_kgdm` is an integrity and documentation component that is already included in `root_c_kgdm`, rather than a separate runtime input.

Three all-caps rows of the source spreadsheet are section headers rather than commodities and are dropped at ingestion (#752): `TRANSFORMED PRODUCTS` and `AGRO-INDUSTRY BYPRODUCTS` are empty, and `ANIMAL PRODUCTS` holds the VLOOKUP column-index vector the upstream `Coefs` sheet addresses by absolute position, which read as data claims an `Edible_portion` of 4.

The `Aquatic Plants` row is WHEP's own, not the upstream workbook's, which has no seaweed row: items_full bridged CBS item 2775 to a placeholder name that matched nothing, so its feed mass dropped out of feed availability and its food mass carried no protein or energy (whep#970). Two sources fill it, one per use:

- **Food** (`N_kgN_kgFM`, `GE_product_MJ_kgFM`, `Lipids_g_kgFM`, `Edible_portion`): FAO's own composition factor for the item, read from FAO, *Food balance sheets: a handbook*, Rome, 2001, Annex I "Food composition in terms of retail weight", row `AQUATIC PLANTS`, PDF page 68 (printed 64), https://www.fao.org/docrep/pdf/011/x9892e/x9892e00.pdf: 54 kcal, 2.8 g protein and 0.6 g fat per 100 g of the commodity as recorded, i.e. fresh. So nitrogen is 2.8 / 100 / 6.25 = 0.00448 kg per kg, the energy cell holds 540 kcal, 2.25936 MJ, per kg, and `Edible_portion` is 1 because FAO applies the factor to the recorded quantity directly. The energy is FAO's dietary energy, not bomb-calorimeter gross energy: it is what `build_food_supply()` turns into `energy_kcal_cap_day`, against FBS element 664. Oracle: FAOSTAT FBS item 2775, World (area 5000), 2010, implies 26.5 g protein per kg (element 671 over element 5142), 5.5% below the 2001 factor; and 372 kcal per kg, so current FBS uses a lower energy factor than the 2001 table.
- **Feed** (`Product_kgDM_kgFM`, `Product_kgN_kgDM`): Heuze, V., Tran, G., Giger-Reverdin, S., Lessire, M. and Lebas, F. (2017), *Seaweeds (marine macroalgae)*, Feedipedia, INRAE, CIRAD, AFZ and FAO, https://feedipedia.org/node/78, table "Laminaria and Saccharina seaweeds" (fresh): dry matter 15.4% as fed (SD 3.6, range 9.7-22.5, n = 56), crude protein 9.8% of DM (n = 50), so 0.01568 kg N per kg DM. It is the only fresh-seaweed table on that page; the others are dried meals at 85-93% DM, which would put fresh mass at nearly its own dry weight.

The two disagree on nitrogen per kg of fresh matter – 0.00448 against 0.154 x 0.01568 = 0.00241 – because FAO's factor covers the whole aquatic-plant basket and Feedipedia's table covers kelps only. Protein differs widely across that basket: McHugh, D. J. (2003), *A guide to the seaweed industry*, FAO Fisheries Technical Paper 441, chapter 8, https://www.fao.org/4/y4765e/y4765e0b.htm, puts nori (*Porphyra*) at 30-50% protein and *Laminaria* at about 10%. Each value is kept on the use it was published for. `Product_kgC_kgDM` and the remaining columns are left empty: no source was read for them.

Source

Compiled from multiple sources including FAO food composition data, crop physiology literature, and IPCC Tier 1 coefficients.

Examples

```
head(biomass_coefs)
```

`build_ag_land_support` *Build the gridded agricultural land support.*

Description

Assembles the physical agricultural land support that `build_n_inputs()` allocates its non-crop-specific nitrogen terms over: per grid cell, polity, year and CBS item, the hectares of agricultural land available to receive nitrogen. Cropland hectares come from the LUH2-derived `type_cropland` surface, split among crops by the static `crop_patterns` composition (normalised within each cell, so the cell's physical cropland area is apportioned rather than inflated by multicropping). Grassland hectares come from `read_luh2_landuse()`'s gridded grassland class and are all carried on CBS 3000, with no intensive/extensive split inferred. Both sides are split across border polities by the same `cell_polity` crosswalk.

Years with cropland but no grassland coverage (a grassland source that runs short of the cropland surface, as `"luh2"` does after 2015) keep their cropland support and raise a warning naming the affected years; supply `data$grassland_ha` to cover them.

Usage

```
build_ag_land_support(
  years = NULL,
  grassland = c("gridded_pasture", "luh2", "none"),
  polity_validity = c("keep", "flag", "drop"),
  data = list(),
  example = FALSE
)
```

Arguments

`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the cropland surface covers.
`grassland`	Grassland-support source. `"gridded_pasture"` (default) is the prepared per-cell `pasture_ha` + `rangeland_ha` surface, which shares the cropland surface's grid and 1851-2023 span. `"luh2"` reads the same LUH2 classes through `read_luh2_landuse()` and agrees with it where they overlap, but stops at 2015. `"none"` returns cropland-only support, an explicit choice rather than a silent gap.

`polity_validity`
: What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`data`
: Optional named list of pre-loaded inputs to avoid remote reads: `cell_polity` (the `build_cell_polity()` crosswalk), `type_cropland` (`lon`, `lat`, `year`, `luh2_type`, `type_ha`), `crop_patterns` (`lon`, `lat`, `item_prod_code`, `harvest_fraction`), `gridded_pasture` (`lon`, `lat`, `year`, `pasture_ha`, `rangeland_ha`), `states` (`read_luh2_landuse()`'s raw LUH2 states) and `grassland_ha` (`lon`, `lat`, `area_code`, `year`, `area_ha`, bypassing the grassland read entirely). Each falls back to its reader when absent.

`example`
: If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `area_code`, `item_cbs_code`, `year`, `land_use` (`"cropland"` or `"grassland"`) and positive `area_ha`, plus the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells –

add_polity_code() says so as mapping_status == "backcast_anchor", and polity_coverage_gaps() reports it as gap_kind == "backcast_anchor". These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so reporting_polity_code can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after SUD-1956-2011 ends, and its post-2011 rows carry that code. These columns do not say so: add_polity_code() reports such a row as mapping_status == "out_of_span", and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. polity_coverage_gaps() reports the stand-in rows of a built table, and options(whep.polity_mapping_status = "flag") (or "status") carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_ag_land_support(example = TRUE)
```

---

build_allocation_layer

Choose an allocation depth per country and assert no cell is claimed twice

---

Description

Decision 10's allocation layer: ONE country grid in which each country is represented at exactly one depth. A container listed in granted arrives as its granted-depth units and as nothing else; every other country arrives as its level-0 row. The engines see a single grid and need no depth logic.

Usage

```
build_allocation_layer(grid0, grid_deep, granted)
```

Arguments

grid0	The level-0 country grid, e.g. read_level_country_grid(0L).
grid_deep	A granted-depth country grid, e.g. read_level_country_grid(1L). May also carry level-0 rows for countries that are not granted; they are ignored here and taken from grid0.
granted	A tibble with area_code (integer container code) and level (integer depth), one row per container – the granted rows of the T25 coverage report. NULL or a zero-row tibble returns grid0.

Value

grid0-shaped tibble with level_polity_code and level, carrying the ragged-coverage diagnostic in its "ragged_coverage" attribute and the unclaimed-land report in its "unclaimed_land" attribute (each a zero-row tibble when nothing was found).

The two-part assertion

(a), a one-sided abort The shares of one physical cell may never EXCEED 1, within `1e-8`, **at any one year**: the sum is taken over the rows valid together, once per epoch the layer carries, and a layer with no validity intervals is one epoch and therefore the unconditional sum T37 agreed on. Summing ACROSS epochs was the defect: two successive polities each hold their cell whole and never coexist, so every succession of the year-aware level-0 read – the USSR, Yugoslavia, Sudan, Czechoslovakia – summed to 2 and no year-aware grid could be combined with a granted depth. A sum above 1 is ground claimed twice – a container kept beside its own units – and is invisible downstream because every national total still reconciles, so it aborts with class `whep_alloc_layer_not_partition`. A sum BELOW 1 is not an error: it is land inside the cell that no reporting polity claims, the same unclaimed ground the year-aware level-0 read leaves out, and it is reported rather than refused (see the unclaimed-land section). A compartment holding two different shares of one cell IN ONE EPOCH is a contradiction rather than a share that moves, and is refused with class `whep_alloc_layer_varying_share`; a share that differs between two DISJOINT intervals of one compartment is now read at each of them, which is what the year-aware level-0 read produces on real data.

(b), a diagnostic For every granted country: no container-keyed row survives anywhere, and per cell AND EPOCH the unit shares reproduce the country's level-0 share **in that same epoch**. Both sides are swept together per (cell, container), so neither is ever compared against a total taken across every epoch of the other. The epochs read are those of the container's own depth in the cell – from its first unit row to its last, gaps inside it included – because outside that span the layer holds no units for it at all and the 2015 snapshot claims every year, which is the vintage question open at T31(j) rather than anything the units did; land the layer then leaves unclaimed is reported by the other attribute. A container with no unit row in a cell at all is reported over its level-0 span. Rows that fail are RETURNED, in the `"ragged_coverage"` attribute, and are never back-filled with the container – back-filling would restore exactly the fold this feature removes. (b) failing is not by itself a defect: level 0 is a fixed 2015 snapshot while granted depths are year-filtered by edge validity, and how the two coexist is open at T31(j).

Unclaimed land

A cell whose shares fall short of 1 holds land that no reporting polity claims. That is a real state of the grid and not corruption: on the `20260825T102349Z-1a0eb` polycell support, the level-0 grid at the 2015 snapshot has 37 such cells out of 66,709 – the worst claimed to 0.0116 at (20.75, 42.75) – and the year-aware read leaves whole cells out on top of that, which is what `validation/spatialize_grid_vintage.R` sizes per year. Aborting on the shortfall would stop every granted-depth run on real data; ignoring it would hide land that no national total accounts for. So it is measured and returned, ONE ROW PER (CELL, EPOCH) that falls short, in the `"unclaimed_land"` attribute: `lon`, `lat`, `start_year`, `end_year` (`NA` for a bound the layer never gave), `claimed_share`, `unclaimed_share`, and `unclaimed_ha`. A caller reads that attribute to learn how much land is unclaimed, where, AND WHEN.

The epoch is in the report because a cell is not one claim: it is short in the years it is short. Judging each cell at its fullest epoch instead – the moment the double-claim check has to

read – forgives a cell short only before one of its holders existed, but by the same line of code it also hides a cell that goes short in a LATER epoch, which is what a granted depth ending before its container produces, and it turned an abort into silence. So every epoch is read.

WHICH YEARS ARE READ is a second question, and reading each cell between its own first claim and its last answered it wrongly. A depth stops covering its container either by holding a smaller share or by holding NO ROW – which is what a real deep grid produces, since a unit has no rows outside its validity – and in the second case the cell's last claim IS the depth's end, so the window closed exactly where the silence began. On the `20260825T102349Z-1a0eb` support, a depth for `area_code` 33 that reproduces its level-0 shares exactly but stops in 1990 left 7,810 of the 8,109 cells it still holds after 1990 in neither report – 826.5 Mha of land, each cell counted once – and 0 ragged rows. It is now 0 cells.

The window is therefore the union of the layer's span in the cell and the level-0 span of the granted containers holding it: the same rule at both ends, so a depth that starts late is caught like one that ends early, and a cell the depth never reaches at all is read over the level-0 span alone. It widens only to years level 0 itself states – a row carrying no validity interval (the 2015 snapshot, which has no time dimension) gives no bound to fall short of, and the window then stops at the layer's own.

The ragged-coverage report does NOT widen with it, and the asymmetry is deliberate. This report asks what the LAYER claims, where absence is a fact whatever `grid0`'s vintage; assertion (b) COMPARES two grids, and outside a container's depth span it would be comparing year-scoped units against a level 0 that may claim every year by having no time dimension at all – the open vintage question at T31(j), not a defect of the units. So the land a stopped depth leaves is reported here, once, rather than twice or not at all.

`unclaimed_ha` is `NA` where the layer carries no `land_area_ha` at all – never 0, which would deny the shortfall. For an epoch no compartment claims, the shortfall is instead read from `base`: `grid0`'s own row for the granted container, in that same cell and on its own interval, which `.level_cell_segments(shares, base)` already receives `base` for and which carries a measurement rather than an absence. `unclaimed_ha` is `NA` there only when `base` itself gives no matching land for the epoch – no `base` was supplied (a direct caller of the internal `.level_unclaimed_ha()`), or the container's own row carries no `land_area_ha` either. Every hectare figure this function reports assumes `land_area_ha` on each row already equals `cell_area_frac` times the cell's whole land; that is a contract on the CALLER's `grid0` and `grid_deep`, not something re-derived or checked here, so a layer whose two disagree reports the column's stated value. The attribute is always present; it is a zero-row tibble when every cell is fully claimed in every epoch, and also on the no-grant path, which returns `grid0` unexamined.

See Also

`read_level_country_grid()`.

Examples

```
grid0 <- tibble::tribble(
  ~lon,  ~lat, ~area_code, ~cell_area_frac,
  10.25, 40.25,       900L,             0.6,
  10.25, 40.25,       901L,             0.4
```

```
)
grid_deep <- tibble::tribble(
  ~lon,  ~lat, ~area_code, ~level_polity_code, ~level, ~cell_area_frac,
  10.25, 40.25,       900L, "A-A1-1900-2100",      1L,             0.25,
  10.25, 40.25,       900L, "A-A2-1900-2100",      1L,             0.35,
  10.25, 40.25,       901L, NA_character_,         0L,             0.4
)
granted <- tibble::tibble(area_code = 900L, level = 1L)
build_allocation_layer(grid0, grid_deep, granted)
```

`build_carbon_balance` *Build the historical gridded soil-organic-carbon balance.*

Description

Reconstruct per-cell soil-organic-carbon stock trajectories: run the selected turnover model to equilibrium under the earliest per-land-use carbon inputs, open each land-use class at the stock `init` selects, march forward on yearly per-cell per-land-use areas applying the model annual update plus a carbon-conserving land-use-change transfer, and derive the soil-organic-nitrogen change from the carbon rate via asymmetric soil carbon-to-nitrogen ratios.

Usage

```
build_carbon_balance(
  model = c("hsoc", "rothc", "icbm", "amg", "century", "lpjml"),
  init = c("own_equilibrium", "cell_average"),
  resolution = c("grid", "polity"),
  polity_validity = c("keep", "flag", "drop"),
  data = list(),
  years = NULL,
  crop_groups = list(),
  class_water = c("cell", "regime"),
  density_basis = c("renormalised", "static"),
  method_grazing = c("whep", "lpjml"),
  method_som_cn = c("justes_2009", "nicolardot_2001", "century"),
  method_input_cn = c("known_crops", "require_all"),
  block_years = 10L,
  example = FALSE
)
```

Arguments

`model`	Turnover model: one of `"hsoc"` (default), `"rothc"`, `"icbm"`, `"amg"`, `"century"` or `"lpjml"`. The choice sets the equilibrium target, and through the time constant `soc_eq / c_input` the speed the stock relaxes toward it; the transient itself is a single exponential for every model.

`init` How each land-use class's opening stock is set. Neither option is the physical one and the default is **not** settled evidence; the measurements behind both sit on `.cb_init_density()` in the source. `"own_equilibrium"` (default) starts every class at the stock its own carbon input and climate support: it removes the opening transient the balance would otherwise report as soil nitrogen mineralization, at the cost of opening cropland at roughly a third of the carbon measured in those soils. `"cell_average"` starts every class in a cell at the fraction-weighted mean of the classes sharing it, the behaviour of the earlier regional historical reconstruction this balance is ported from: it opens cropland near its observed stock, as a proxy for the legacy carbon of the vegetation it replaced, at the cost of then draining that stock toward an equilibrium whep#799 puts several-fold too low. Recorded in `method_soc_init`.

`resolution` `"grid"` (default, per cell and land-use class) or `"polity"` (aggregated to `area_code` conserving carbon mass).

`polity_validity`
What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`data` Named list of pre-loaded inputs. Every entry named below falls back to its reader when absent EXCEPT the two grazing entries at the end, which have no reader at all and must be supplied under the default `method_grazing = "whep"`; the entries that do fall back are `c_inputs` (per cell, land-use class and year, with `c_input_mgc_ha_yr` and `humified_fraction`); `land_use` (yearly per-cell per-class `lon`, `lat`, `area_code`, `year`, `land_use`, `area_ha`); `climate` (either a precomputed per cell-year `climate_modifier`, applied to every land-use class alike, or the raw monthly drivers `temp_c` and `water_minus_pet_mm` keyed by `lon`, `lat`, `area_code`, `year`, `month`, from which the selected model's native modifier is computed internally per land-use class: for the RothC/HSOC cover term the monthly vegetated soil-cover fraction is taken from the generic land-use curve `soc_soil_cover_curve` (a crop growth-stage canopy for cropland, sustained perennial cover for grassland/natural), so any `soil_cover` column supplied on the raw drivers is ignored); `clay` (per cell `clay_pct`); and an `natural_cover` (per cell and year, with `natural_cover`, the vegetated fraction of the natural stand, from `read_lpjml_natural_cover`); when supplied it replaces `soc_soil_cover_curve`'s constant for the NATURAL class only – managed grassland has no measured cover to use and stays on the curve – and when absent every class stays on the curve, which is the previous behaviour; `cropland_cover` (per cell, year and MONTH, from `read_lpjml_crop_cover`), which replaces the curve for the CROPLAND class with the cover its own crop calendar implies. The curve already gives cropland a season, but an-

chors it to the cell-year's warmest month: measured at 2010 the real crop mid-season falls there in only 5.2% of cropland cells and three or more months away in 51.0%, so the correction is one of timing rather than of annual mean (0.254 on the curve against 0.343 on the calendar); and an optional `equilibrium_climate` (the pre-industrial climatological normal, one representative monthly cycle per cell, used only for the equilibrium spin-up modifier while the forward march uses the year-specific drivers). Two further entries are forwarded to `build_carbon_inputs` for the grassland grazing terms, and read only when `c_inputs` is not supplied: `livestock_intake` (the `redistribute_feed` result) and `excreta` (the `applied` stream of `build_livestock_nutrient_flows`). These are the two with no reader behind them – `excreta` is a livestock-pipeline output, not a readable input – so under the default `method_grazing = "whep"` a call that omits either is refused straight away, before any input is read. Pass `method_grazing = "lpjml"` to charge the grassland the model's own grazing and need neither. Two optional entries are forwarded to `build_carbon_inputs` for the rainfed/irrigated yield split of crop groups, also only when `c_inputs` is not supplied: `regime_ratio` (the `build_regime_yield_ratio` table; pass the one `build_nitrogen_balance` used so both chains apply one ratio) and `regime_production` (the `get_primary_production` table the national yields and the yield bounds come from). Each is built when absent.

`years` Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the inputs cover, but reading the full LUH2 range (850-2015) is infeasible turnkey, so a subset is strongly recommended when the default readers are used. Threaded into every default reader (`read_luh2_landuse`, `get_soc_climate_drivers` and `build_carbon_inputs`); ignored for inputs supplied via `data`.

`crop_groups` How cropland is resolved into land-use classes; see `build_carbon_inputs()`. `list()` (default) marches crop GROUPS – herbaceous crops pooled per irrigation regime, woody crops per species, rainfed and irrigated separate. Each cell-year's LUH2 cropland area is split over the groups in proportion to their crop-pattern area, so LUH2's total is kept (verified on a 2009-2010 run: 1442.8 Mha either way, with the global stock moving 0.01%). Herbaceous groups follow the annual crop cover (and the crop calendar); woody groups take a perennial cover of 0.85, an ASSUMED value with no sourced constant behind it yet. Soil cover is computed once per cover profile and joined to the classes, so the class count does not multiply the monthly climate table. `list(method = "none")` keeps the single `cropland` class the package used before. The default `regime = "yield_split"` builds the irrigated:rainfed yield ratio when `data$regime_ratio` is not supplied (and `c_inputs` is not), which needs SPAM2010 (downloaded into the user cache on first use, so the first run needs network), the `lpjml-crop-regime-yield` pin (or `WHEP_LPJML_RUN_DIR`) and the FAOSTAT production, fertiliser and land-use pins plus `luh2-areas`; see `build_carbon_inputs()`. `list(regime = "area_split")` needs none of them.

`class_water` How a cell's applied irrigation is shared among its land-use classes in the

moisture term. "cell" (default) gives every class except natural land the cell-level water surplus, irrigation included, as before. "regime" concentrates the irrigation on the irrigated crop groups in proportion to their share of the cell and runs every other class on rain alone; the area-weighted mean over classes is the cell value either way. Needs crop_groups, because only groups carry a regime. Recorded in method_class_water.

density_basis Which crop area weights the per-crop carbon densities when they collapse to a class; see build_carbon_inputs(). "renormalised" (default) uses the yearly FAOSTAT-renormalised cell area the densities were computed on; "static" keeps the crop-pattern weights the package used before. Only read when the carbon inputs are built here rather than supplied.

method_grazing
Whose grazing removes carbon from grassland and returns it as excreta; see build_grass_natural_carbon_inputs(). "whep" (default) charges the class WHEP's own grass intake and applied excreta, so it needs data$livestock_intake and data$excreta and aborts without them; "lpjml" uses the model's own livestock module and needs neither. Only read when the carbon inputs are built here rather than supplied through data$c_inputs.

method_som_cn Which published parameterisation sets the C:N at which soil organic matter forms from the carbon input that formed it, CN_new = a - b / CN_input, floored and then bounded by the IPCC 2019 land-use range. "justes_2009" (default, a = 15.4, b = 76) is the refit on the larger combined dataset; "nicolardot_2001" (16.1, 123) is the original fit, and the citation HSOCN's own nitrogen submodel follows, so it is the parameterisation that puts WHEP and HSOCN on one basis; "century" (16, 120) is CENTURY/DayCent's shipped parameterisation. All three, each with its source, sample and floor, are in inst/extdata/balances/som_marginal_cn.csv. The choice sets the nitrogen of a carbon change and never the carbon: at an input C:N of 40 the three give a marginal C:N of 13.50, 13.03 and 13.00, so son_change_kgn_ha spans 3.8%; at a narrow (manure-like) input C:N of 12 they give 9.07, 8.00 and 10.00, a 25% spread, and the widest anywhere is 29% at an input C:N of 15; above an input C:N of 70 they never differ by more than 1.9%. Recorded per row in method_som_cn, which instead reads "land_use_default" on a row whose input C:N is unknown and "directional_ipcc_range" when no input C:N is carried at all.

method_input_cn
What a crop of unknown input C:N does to the C:N of the cropland class it sits in, and so to the C:N of the organic matter that input forms; see build_carbon_inputs(). "known_crops" (default) forms the class ratio from the crops that have one; "require_all" leaves it NA, which sends the class to the land-use default C:N. Only read when the carbon inputs are built here rather than supplied through data$c_inputs.

block_years How many consecutive years are built at a time. The climate drivers and the land-use areas are read, the class table and its equilibria built, the stocks marched and the output tail run one block of years at a time, and each cell's soil state is carried across a block boundary exactly as

the march carries it from one year to the next. The result is therefore identical, bit for bit, to building the whole span in one pass, whatever the value: only memory and run time move. Memory then grows with the block rather than with the span; see the section below. `Inf` builds the whole span as one block, which is the fastest and the largest. A block is cut from `years` when the land use is read, and from the years of `data$land_use` when it is supplied; with neither, the span is unknown before reading and the build runs as one block.

`example` If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Details

`polity_validity` governs this function's own output. The internal `get_soc_climate_drivers` read it falls back on always keeps its rows: the march needs a climate modifier for every cell-year it steps through, so dropping driver rows for an anachronistic polity label would break the trajectory rather than relabel it. The driver read therefore warns on its own key space (whep#462) while this argument decides the fate of the balance rows.

Value

A tibble keyed by `(lon, lat, area_code, land_use, year)` at `"grid"` resolution (or `(area_code, year)` at `"polity"`), with `stock_mgc_ha`, `mineralization_mgc_ha`, `c_input_mgc_ha`, `luc_transfer_mgc_ha`, `luc_transfer_mgc`, `rate_mgc_ha`, `son_change_kgn_ha`, `area_ha`, and one column per method choice that moves a number: `method_soc`, `method_soc_init`, `method_class_water`, `method_area_basis`, `method_grazing`, `method_som_cn` and `method_input_cn` and `method_crop_groups`. All of them survive the `"polity"` roll-up, which additionally carries `input_land_ha` (the land the land-use input gave that polity-year) and `modelled_land_frac` (`area_ha / input_land_ha`, the share of it the densities in the same row are a mean over); see the coverage section below. Plus the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`.

Memory, and why the span is built in blocks

Built in one pass, every stage holds the whole span at once, and two of them hold many times the table they return: the monthly climate read peaks at about 15 times its output and the output tail at 5 to 6 times its own (whep#1287). `block_years` bounds both by the block instead. Measured on the real grid as the process's peak committed memory, with the carbon inputs supplied, and the output identical in every case:

span	one pass	10-year blocks	3-year blocks	2-year blocks
1961-1966	13.8 GB	(one block)	12.4 GB	11.6 GB
1961-1972	18.5 GB	17.1 GB	13.6 GB	

The single pass grows by about 0.8 GB per year of span. A blocked build grows only by what it must keep: the output (about 0.12 GB per year at `"grid"`) and the carbon inputs (about 0.08 GB per year), which are read once over the whole span because a year-scoped production and commodity build returns different rows for the same years (whep#833,

whep#834). From 1961-1966 to 1961-1972 the 3-year build rose 1.20 GB, against 1.17 GB for those two terms. Each year added to the block costs about 0.5 GB.

Extrapolated linearly from these points to 1931-2023 (not measured there), one pass would need about 80 GB, 10-year blocks about 33 GB and 3-year blocks about 29 GB, while the span-wide carbon-input read that precedes the blocks is extrapolated in whep#1287 to 45-55 GB. The default of 10 years keeps the blocks below that read, which is then the build's peak; every block re-reads the static cell support, clay and climate series, at 3 to 4 minutes a block, so 3-year blocks would cost about an hour more over 1931-2023 to save about 4 GB that the carbon-input read has already spent.

Nothing about the carbon itself is decided per block. The balance opens from the first year it marches, and the year-less equilibrium-climate normal is read only then; the check that every cell carries every year runs across block boundaries; and each warning and message the class table and the output tail raise is gathered over the blocks and raised once for the whole span, as a single pass raises it. Messages the land-use and climate readers raise about their own read are raised by each block's read.

The land-use-change ledger closes on mass, not on density

`luc_transfer_mgc_ha` is the carbon a class received (positive) or gave up (negative) through land-use change, per hectare of the class's CURRENT area. A class whose area falls to zero still gives up its whole stock – the balance carries the row at zero area and moves the carbon into the growing classes – but at zero hectares that outflow has no per-hectare expression, so it is reported as 0 and `sum(luc_transfer_mgc_ha * area_ha)` over a cell-year is then positive by exactly the vanished stock. `luc_transfer_mgc` is the same transfer as a signed mass in Mg C, on every row including the vanished one, and sums to zero within every cell-year (and, at `"polity"` resolution, is the summed mass). Check conservation on the mass column.

Soil depth

Every carbon and nitrogen density this function reports – `stock_mgc_ha`, `mineralization_mgc_ha`, `c_input_mgc_ha`, `luc_transfer_mgc_ha`, `rate_mgc_ha` and `son_change_kgn_ha` – is a **0-30 cm topsoil** quantity, not a whole-profile one. The depth is a property of the model family, not a free choice: HSOC comes from Aguilera et al. (2018), which states that "the model was applied to the 0-30 cm layer of the soil"; the humification fractions in residue_humification are that paper's Table 2; and the RothC/HSOC climate modifier rescales RothC's own 0-23 cm maximum topsoil-moisture-deficit expression to 30 cm (`soc_rate_modifier_rothc(soil_depth_m = 0.3)`).

Comparing this output against a whole-profile soil-carbon product is therefore a category error. LPJmL's `soilc`, in particular, reports carbon over its top 3 m and is roughly three times a topsoil stock; global 0-30 cm references such as GSOCmap are the valid comparators. Stating this is not pedantry – an unstated depth convention is what made a chain of contradictory diagnoses possible (whep#799).

Spatial support

Every default reader on the carbon path – the land-use areas, the carbon inputs, the climate drivers and the clay – resolves its cell-to-polity table through one polycell support

(`read_polycell_support()`), read at a static reference year. A cell shared between polities therefore delivers to each only the land it holds there, and no reader can be left on a different crosswalk: half the path on one footprint and half on another would surface as an ordinary climate-coverage warning from the modifier join, not as an error. Land the reporting vocabulary cannot key (no `area_code`) is reported and dropped, never folded into another polity's.

A polity total covers the modelled land, not the polity

The climate drivers are on the LPJmL run's grid, a coarser land mask than LUH2's, so land outside it has no climate modifier and leaves the balance. A `"polity"` row is therefore an area-weighted mean over the hectares that survived, and `modelled_land_frac` is the share of the polity-year's land that did: read it before quoting a national stock or trend. It is **not** a quality flag on the hectares that are there – they are modelled exactly as before – and no value in this output changes because of it.

Measured at 2010 against the pinned `lpjml-soc-hydrology` grid, which drops 296.5 Mha of LUH2 land over 7,070 cell-polity compartments: global coverage is **0.977**, 149 of 193 polities are below 1, 51 are below 0.9, and **12 carry no row at all** – Antigua and Barbuda, Bahrain, Barbados, Dominica, Grenada, Macao, Malta, Mauritius, Saint Kitts and Nevis, Saint Lucia, Singapore and Tonga, whose land is entirely outside the grid. Worst of those that survive: Cabo Verde 0.22, Rest of World 0.31, Vanuatu 0.32, Bahamas 0.32, Cyprus 0.35, Somalia 0.72 (18.0 Mha), the Philippines 0.73 (7.7 Mha), Greece 0.76 (3.1 Mha). A warning names the polities that vanish, because no column on the output can: they have no row to carry one.

The land is reported rather than gap-filled, and that is a decision with evidence behind it (whep#1166). The run grid holds 58,795 cells against CRU TS's 67,420 land cells, and what it excludes is fractional-land coastal and island cells: median land fraction 0.14, against 0.998 inside the grid. CRU still carries temperature, PET and precipitation for 5,981 of the 6,847 cells that go – 291.5 of the 296.5 Mha – so a climate *could* be assembled there without LPJmL. The carbon input could not: 261.4 of those 296.5 Mha are natural (214.8) and grassland (46.6), whose input is the LPJmL net carbon flux, and this package holds no second source for it. Filling the climate alone would march 261.4 Mha on a zero-filled carbon input, draining its whole opening stock – the failure whep#1146 named, which today reaches 0.212 ha globally. Filling the input as well means asserting a productivity for land the model was never run on. Only a rerun on a land mask matching LUH2's puts real carbon on those hectares.

What this balance does not cover

The balance runs on the LPJmL grid, because the climate drivers do. That grid is a coarser land mask than LUH2's, so **not every LUH2 hectare is in the output**. On the default readers at 2010, 7,079 cell-polity compartments carrying 296.8 Mha of LUH2 land – 30.4 Mha cropland, 46.6 Mha grassland, 214.9 Mha natural, 4.8 Mha urban – have no climate driver and are dropped, with a warning naming the hectares and the worst-hit polities. The loss is essentially a coastline, so it is negligible globally (1.4% of grassland, 2.6% of natural land) and large for maritime polities: 31.6% of Greece's grassland, 25.3% of the Philippines', 22.7% of Somalia's, 17.8% of Indonesia's, 12.8% of the United Kingdom's, 12.5% of Italy's. A national SOC total from this function is a total over the modelled hectares, not over the polity.

Dropped is not the same as marched at zero. A class with LUH2 area but no carbon-input row IS kept, at zero input (that is what makes `urban` dilute rather than deflate the cell), and whep#1146 read the coverage gap as that case. It is not: measured against the `lpjml-grass-natural-net-c` pin and the real climate table, every one of those 46.6 Mha of grassland is dropped for want of a climate driver, and the grassland actually marching on a zero-filled input is 0.212 ha globally at 2010. Both quantities are reported at run time rather than left to be re-derived.

The support is keyed on the polity_area_crosswalk `polity_area_code` bucket, because that is what every national table the path joins to carries: `.aggregate_to_polities()` groups on that column and renames it `area_code`. Sudan and South Sudan therefore report together under `206`, as they do in the production and CBS series. The fold is read through the live crosswalk, so `options(whep.unfold_rest_of_world)` and any future un-fold of `206` (whep#680) move the grid and the national tables together rather than separating them (whep#1168).

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

Aguilera, E., Guzman, G. I., Alvaro-Fuentes, J., Infante-Amate, J., Garcia-Ruiz, R., Carranza-Gallego, G., Soto, D. & Gonzalez de Molina, M. (2018). A historical perspective on soil organic carbon in Mediterranean cropland (Spain, 1900-2008). *Science of the Total Environment*, 621, 634-648. doi:10.1016/j.scitotenv.2017.11.243; land-use-change carbon transfer ported from an earlier regional historical reconstruction.

Examples

```
build_carbon_balance(example = TRUE)
```

---

build_carbon_inputs *Assemble the per-land-use-class soil carbon inputs.*

Description

Build the carbon-input layer `build_carbon_balance()` consumes, keyed by `(lon, lat, area_code, year, land`. The cropland class aggregates the per-crop cropland inputs from `build_soil_carbon_inputs()` to the class grain: the class carbon density is the harvested-area-weighted mean of the per-crop densities, and the humification fraction is the carbon-mass-weighted mean of the per-crop fractions (mass = density times area). The grassland and natural classes come from `build_grass_natural_carbon_inputs()` unchanged.

Usage

```
build_carbon_inputs(
  resolution = c("grid", "polity"),
  data = list(),
  years = NULL,
  crop_groups = list(),
  density_basis = c("renormalised", "static"),
  method_grazing = c("whep", "lpjml"),
  method_unspatialized = c("fodder_pattern", "reallocate", "drop"),
  method_input_cn = c("known_crops", "require_all"),
  method_crop_weights = c("spatialized", "static"),
  method_manure_placement = c("livestock", "crop_area"),
  example = FALSE
)
```

Arguments

`resolution`	`"grid"` (default, per cell and class) or `"polity"` (aggregated to `area_code`, area-weighting the cropland density by the polity crop area).
`data`	Named list of pre-loaded inputs, each falling back to its builder when absent: `cropland` (the `build_soil_carbon_inputs()` output, per cell, crop and year, with `total_c_input_mgc_ha_yr` and `humified_fraction`); `crop_area` (per cell, crop and year harvested area with columns `lon`, `lat`, `area_code`,

item_prod_code, year, crop_area_ha, used to area-weight the crop densities); grass_natural (the build_grass_natural_carbon_inputs() output at the class grain); and optional land_use (per-cell class area_ha, used to area-weight grassland/natural polity output); and country_grid, the polycell support resolved to one row per cell and area_code, from which crop_area is derived when absent. It is the same support build_soil_carbon_inputs() reads, so the weights and the carbon they weight can never come from two different crosswalks. When cropland or grass_natural are absent the respective builder is called with the remaining members of data. The yield split of crop_groups$regime reads three more, each built when absent: regime_ratio (build_regime_yield_ratio() output; built on the crop rows that carry a share and a national yield, so pass the table build_nitrogen_balance() used to make both chains read one ratio); regime_production (get_primary_production() output, for each crop's national yield and the yield bounds; read for years and for 1961-2023 when absent); and residue_humification (defaults to residue_humification), whose residue and root coefficients re-weight each part's humified fraction.

`years` Optional integer vector of calendar years to keep. NULL (default) keeps every year the inputs cover. Threaded into the default build_soil_carbon_inputs() and build_grass_natural_carbon_inputs() builders so their readers slice to the requested years; ignored for inputs supplied via data.

`crop_groups` How cropland is resolved into land-use classes, a named list validated element-wise. method: "rotation_groups" (default) resolves cropland into crop GROUPS – herbaceous crops pooled per irrigation regime (they rotate, so nothing inside the pool is a land-use change), woody crops per species, rainfed and irrigated separate – labelled by soc_crop_group(); "none" keeps the single cropland class the package used before, for comparison and for a caller that wants one cropland number. The former name of "rotation_groups" is still accepted as a deprecated alias: it resolves to "rotation_groups", with identical results, and warns once per session. irrigation: where each crop's irrigated share of its cell area comes from. "spatialized" (default) uses build_gridded_landuse() on the pinned spatialization inputs, crop-specific and yearly; "none" puts every crop in its rainfed group. Recorded in method_c_input. A prebuilt share layer can be supplied as data$crop_regime_share (lon, lat, area_code, item_prod_code, year, irrigated_share). regime: how a crop's carbon per hectare differs between its rainfed and irrigated parts. "yield_split" (default) re-evaluates each part's residue and root at the part's own yield. The parts' yields keep the crop's production, with the irrigated:rainfed yield ratio R of the cell-crop-year that build_nitrogen_balance() splits synthetic nitrogen with (the bounded ratio of split_regime_yield() on build_regime_yield_ratio()): with irrigated share s and mean yield Y, Y_r = Y / (1 + s (R - 1)) and Y_i = R Y_r. Each part carries the crop's residue and root carbon per hectare times the response of calculate_crop_residues() and calculate_crop_roots() from the crop's mean yield to the part's yield, evaluated in the part's water regime, so the model's irrigation terms (a lower residue ratio for the crop groups that respond, a lower root:shoot ratio) apply to the irrigated part. Residue

and root are affine in yield, so a high-yield hectare carries less than its yield share, and with the irrigation terms the crop's residue and root carbon fall on irrigated land rather than being kept. A crop wholly in one regime is re-evaluated at its mean yield in that regime. Weed and manure carbon keep the crop's density on both parts. `"area_split"` keeps the crop's densities on both parts, the rule before the yield split. Recorded per class in `method_regime`.

Inputs the default needs. Unless `data$regime_ratio` and `data$regime_production` are supplied, `"yield_split"` builds the ratio with `build_regime_yield_ratio()` and the national yields with `get_primary_production()`, which read: SPAM2010 (`read_spam_yields()`, downloaded from Harvard Dataverse into the user cache on first use, so the first run needs network, unless `WHEP_SPAM_DIR` points at a copy); the LPJmL crop regime yields (the `lpjml-crop-regime-yield` pin, or the run in `WHEP_LPJML_RUN_DIR` when set); and the FAOSTAT pins for production, fertiliser nutrients and land use, plus the `luh2-areas` pin. `"area_split"` builds no ratio and reads none of them.

`density_basis` Which crop area weights the per-crop densities when they collapse to a class. `"renormalised"` (default) uses the yearly cell crop area the densities were computed on – the FAOSTAT-renormalised area `build_soil_carbon_inputs()` returns as `crop_area_ha` – so the class carbon mass equals the sum of the crop masses that were spatialized. `"static"` uses the time-invariant crop-pattern area split by the polycell's share of the cell, which the package used before. The two differ wherever the spatialized cell areas of a polity-crop-year do not sum to its FAOSTAT harvested area: measured on the 2010 pins over 40,065 cropland cells the per-cell class density ratio renormalised/static has an area-weighted median of 0.988 (p5-p95 0.922-1.043). Recorded in `method_area_basis` on cropland rows.

`method_grazing`

Whose grazing removes carbon from grassland and returns it as excreta; see `build_grass_natural_carbon_inputs()`. `"whep"` (default) charges the class WHEP's own grass intake and applied excreta, and so needs `data$livestock_intake` and `data$excreta`; `"lpjml"` uses the model's livestock module instead and needs neither.

`method_unspatialized`

What happens to a polity-crop whose crop has no hectares in the (time-invariant) `crop_patterns` layer. `"fodder_pattern"` (default) first places the fodder crops (FAOSTAT items 636-649, 651 and 655) on the pooled 16 forage layers of Monfreda, Ramankutty and Foley (2008, doi:10.1029/2007GB002947), which the crop-pattern pin omits, gridded exactly as a crop-pattern crop is. The layers are pooled because the per-item ones follow the circa-2000 reporting vocabulary, not where each item is grown later. Whatever the layer still cannot place is reallocated as under `"reallocate"`. The layer is read from the `spatialize-fodder-patterns` pin (built by `inst/scripts/prepare_fodder_patterns.R`); setting `WHEP_MONFREDA_DIR` to the `HarvestedAreaYield175Crops_Geotiff` folder of the Monfreda archive (`inst/scripts/download/download_monfreda.R`) rebuilds it from the rasters instead, and `data$fodder_patterns` replaces it. It aborts

rather than falling back to uniform reallocation when the layer is empty or a set `WHEP_MONFREDA_DIR` is wrong. The layer codes are matched on the layer names, as the archive's metadata gives no FAO name for them (assumed, unverified against a Monfreda table). At 2010 it moves 83.6 Mt C between cells relative to `"reallocate"`, with polity totals unchanged. `"reallocate"` spreads that carbon over the polity's crop-pattern cropland cells in proportion to each cell's total cropland area, and puts it on the crop's FAOSTAT national harvested area, so the polity's gridded carbon mass equals its national mass; this is the rule the sibling nitrogen spatialization already applies to the same gap (`spatialize_country_n_to_crops()`), and was the default before whep#1118. `"drop"` discards it, which is what the package did before and what a caller who prefers a hole to a smear should ask for. Either way the mass and the affected crops are reported, and the choice is recorded in `method_unspatialized`. Reallocation needs a national area to put the carbon on, so a polity-crop with no `harvested_area` row – and a polity with no cell at all in the support, which no rule here can reach (see whep#1002) – is dropped under all three methods, reported separately. Under either `method_crop_weights`, the fodder layer only fills the polity-crops that year's crop weights do not place, and the 83.6 Mt C figure above was measured on the static weights.

`method_input_cn`

What a crop of unknown input C:N does to the C:N of the class it sits in. `"known_crops"` (default) forms the class ratio from the crops that have one, weighted by the carbon that came with them – the rule `build_soil_carbon_inputs()` already applies one level down, where the input nitrogen totals only the components that carry a nitrogen and the carbon paired with them. `"require_all"` leaves the class ratio `NA` as soon as any one crop's is unknown, so the class takes the land-use default C:N instead of the ratio its other crops measured. Recorded in `method_input_cn` on cropland rows.

`method_crop_weights`

Where each crop's within-polity cell weights come from. `"spatialized"` (default) takes them from the spatialization engine's crop-level output, `build_gridded_landuse()` run on the pinned spatialization inputs and the same cell support, year by year: the crop's placement follows that year's cropland, its irrigated share and the engine's per-cell capacity ceiling, so the carbon lands where the gridded land use puts the crop. `data$gridded_crops` supplies that output pre-built. `"static"` uses the time-invariant `crop_patterns` layer (`harvest_fraction` times each cell's cropland averaged over every year of the gridded-cropland pin), one map for every year, which is what the package did before (whep#1002). Measured on the pins, the L1 distance between the two cell-share vectors of a polity-crop (0 = same placement, 2 = disjoint) has an area-weighted mean of 0.30 in 1961, 0.44 in 2010 and 0.47 in 2020. Polity totals are the same under both; only where the carbon sits moves. Recorded in `method_crop_weights`.

`method_manure_placement`

Where the cropland manure carbon is put. `"livestock"` (default) places

it where the animals are, the grain the gridded nitrogen balance uses for the same manure (whep#1300): the feed intake is spread to cells by gridded livestock heads on `country_grid` (the local grain of `build_feed_intake_local()`), and `build_livestock_nutrient_flows()` runs at `"subnational"`, so each cell's collected manure goes to that cell's crops up to the 170 kg N/ha ceiling and the excess is trucked to neighbouring cells with room. Its crop layer is that year's cell-crop area, the hectares the carbon is divided by. Manure with no crop – trucked in, or over-applied above the cap – is spread over its cell's crops by area, as the nitrogen balance spreads it, and manure on a cell with no crop over its polity's cropland; only a polity with no cropland cell loses it, with a warning. Measured at 2010, it places 322.1 Tg C of cropland manure where `"crop_area"` places 346.9 Tg C, because the cell-grain intake caps grazing at each cell's grass; 41.5% of the manure carbon sits on a different cell. `"crop_area"` allocates the polity's manure to its crops by harvested area and fans each polity-crop out to cells by crop area, as residue and roots are, which is what the package did before (whep#1307); it drops the over-cap manure the national allocation leaves with no crop (whep#805). A supplied `data$manure` must be keyed on cells (`sub_territory` a `"lon_lat"` cell id) under `"livestock"`; a national stream aborts. Recorded in `method_manure_placement`.

`example` If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble keyed by `(lon, lat, area_code, year, land_use)` at `"grid"` resolution (or `(area_code, year, land_use)` at `"polity"`), with `c_input_mgc_ha_yr`, `humified_fraction`, `method_c_input`, and `method_unspatialized`, `method_input_cn`, `method_crop_weights` and `method_manure_placement` (all `NA` on the grassland and natural classes, which are neither spatialized from polity-crop totals nor collapsed from crops), for `land_use` in `"cropland"`, `"grassland"` and `"natural"`, plus the polity columns below. Cropland groups also carry the regime split, `NA` on the single `"cropland"` class and on grassland and natural:

- `method_regime`: the `crop_groups$regime` rule, `"yield_split"` or `"area_split"`.
- `method_regime_split`: the split rules of the class's crop parts, sorted and joined by `"|"`: `"yield_ratio"` (part yields from the regime yield ratio), `"trivial_rainfed_only"` and `"trivial_irrigated_only"` (the crop has one regime in the cell), `"no_ratio"` (no ratio for the cell-crop-year), `"area_no_production"` (no national yield), `"area_no_components"` (the per-crop layer has no residue and root densities), `"area_split"` (asked for) and `"no_regime_share"` (no irrigated share; booked rainfed). The last five keep the crop's own densities on both parts.
- `method_residue_regime`: how the parts' residue and root were set, joined the same way: `"part_yield"` (re-evaluated at a rainfed part's yield), `"part_yield_irrigation_terms"` (re-evaluated at an irrigated part's yield, with the model's irrigation terms) or `"crop_density"` (the crop's own density).
- `yield_split_c_share`: the share of the class's carbon input from crop parts re-evaluated at the part yields of the regime yield ratio (rule `"yield_ratio"`).

- `method_regime_cn`: how the parts' input C:N was formed, joined the same way: `"part_input_cn"` (from the part's own components' carbon and nitrogen) or `"crop_input_cn"` (the crop's, where the per-crop layer carries no component nitrogen).

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

Cropland inputs from `build_soil_carbon_inputs()`; grassland and natural inputs from `build_grass_natural_carbon_inputs()`; assembled per the WHEP historical carbon-balance design.

Examples

```
build_carbon_inputs(example = TRUE)
```

`build_cbs_prices` *Build CBS item prices*

Description

Compute prices for all commodity balance sheet items, including processed products and crop residues. Prices are derived from trade data, with special handling for items without direct trade prices (palm kernels, soy hulls, brans, etc.). Crop residue prices are estimated as a fraction of the product price: only primary crops and grassland bear residues, routed to straw (cereals and pulses), firewood (woody crops) or other crop residues. Crops whose herbaceous/woody habit is unknown take the herbaceous default and are reported in a warning.

Usage

```
build_cbs_prices(
  cbs,
  trade_prices = NULL,
  residue_price_factor = 0.1,
  example = FALSE
)
```

Arguments

`cbs`	A tibble of commodity balance sheets, as returned by `build_commodity_balances()` or `get_wide_cbs()`.
`trade_prices`	A tibble as returned by `build_trade_prices()`. If `NULL`, it is computed internally.
`residue_price_factor`	Numeric. Relative price of crop residues compared to the product. Default `0.1`.
`example`	Logical. If `TRUE`, return a small example tibble. Default `FALSE`.

Value

A tibble with columns:

- `year`: Integer year.
- `element`: `"import"` or `"export"`.
- `item_cbs_code`: Numeric CBS item code.
- `price`: Price in KDollars per tonne.

Examples

```
build_cbs_prices(example = TRUE)
```

`build_cell_climate_zone` *Build the per-cell IPCC climate zone from CRU mean annual temperature.*

Description

Reduce CRU TS monthly near-surface temperature to a mean annual temperature (MAT) per 0.5-degree land cell and year, and classify each cell into the IPCC manure-management climate zone (`"Cool"`, `"Temperate"`, `"Warm"`) that climate_mcf is keyed on. This is the per-cell replacement for the single hardcoded `"Temperate"` zone the manure CH4 model falls back to when no zone is supplied.

CRU TS covers 1901 onwards, so years outside its span cannot be measured. They are not dropped and not silently given a measured year's value: the 30-year climatology at the nearest end of the record is held constant across them and stamped in `method_climate_zone` with its own label (for example `"climatology_1901_1930"`), so a reader can tell a backcast zone from a measured one without reading this code. Holding an early-record climatology constant backwards is an assumption, not an observation.

Usage

```
build_cell_climate_zone(
  years = NULL,
  cru_dir = NULL,
  data = NULL,
  example = FALSE
)
```

Arguments

`years`	Integer vector of calendar years to return. `NULL` (default) returns every year the CRU record covers. Years before the record are backcast and years after it are forward-held, both from the nearest 30-year climatology.
`cru_dir`	Path to the CRU TS NetCDF directory. Defaults to `Sys.getenv("WHEP_CRU_DIR")`; aborts when neither is set. Ignored when `data` is supplied.
`data`	Optional pre-read tibble of annual means (`lon`, `lat`, `year`, `mean_annual_temp_c`) used in place of reading CRU, for testing. The years it carries are treated as the measured record, so the backcast and forward-hold logic still applies to `years` outside them.
`example`	If `TRUE`, return a small fixture instead of reading CRU data. Defaults to `FALSE`.

Value

A tibble with one row per cell and year:

- `lon`, `lat`: Cell centre coordinates (0.5-degree grid).

- `year`: Calendar year.
- `mean_annual_temp_c`: Mean annual near-surface air temperature (degrees Celsius), the mean of the twelve monthly CRU values.
- `climate_zone`: IPCC zone, `"Cool"`, `"Temperate"` or `"Warm"`.
- `method_climate_zone`: `"cru_ts_annual"` for a measured year, or `"climatology_<first>_<last>"` for a held climatology.

Source

CRU TS (Climatic Research Unit, University of East Anglia; Harris, Osborn & Jones 2020, Scientific Data, doi:10.1038/s4159702004533). Zone cuts: IPCC 2006 Guidelines Vol.4 Ch.3, adopted by the 2019 Refinement Vol.4 Ch.10.

Examples

```
build_cell_climate_zone(example = TRUE)
```

---

build_cell_moisture_regime

Build the IPCC 2019 wet/dry moisture regime of WHEP's crop land.

Description

Classify each cell, polity, commodity balance item, water regime and year of WHEP's own crop land use as IPCC 2019 wet or dry, the classes the Tier 1 managed-soil factors are keyed on.

A zone is tropical where the mean annual temperature exceeds 18 degC and there are at most 7 frost days per year (IPCC 2019 Refinement, Volume 4, Chapter 3, Figure 3A.5.2, p. 3.48). A tropical row is wet where its annual water exceeds 1000 mm; any other row is wet where its annual water divided by potential evapotranspiration (PET) exceeds 1; every other row is dry. Both tests are strict, so a value of exactly 1 or exactly 1000 mm is dry (Volume 4, Chapter 11, Table 11.1, footnote 4, p. 11.13). Footnote 4 bases the disaggregation on long-term averages, so by default the inputs are averaged over the 30 years ending in each year before they are classified.

The water term is precipitation plus, on irrigated rows, the irrigation depth of the row's crops (`water = "precip_irrig"`, the default), or precipitation alone (`water = "precip"`). Rainfed rows get no irrigation.

Usage

```
build_cell_moisture_regime(
  years = NULL,
  window = c("trailing_30", "annual", "normal"),
  water = c("precip_irrig", "precip"),
  data = NULL,
```

```
  config = list(),
  example = FALSE
)
```

Arguments

`years`	Integer vector of the years to classify.
`window`	How the inputs are averaged before they are classified: `"trailing_30"` (default, the 30 years ending in each year), `"annual"` or `"normal"` (one fixed period, `config$normal_period`).
`water`	The water term: `"precip_irrig"` (default, precipitation plus the irrigation depth of the row's crops and regime) or `"precip"`.
`data`	Optional list of inputs used in place of reading them. Each element is optional: • `climate`: `lon`, `lat`, `year`, `mean_annual_temp_c`, `annual_precip_mm`, `annual_pet_mm`, `annual_frost_days`; its years are the climate record. • `crop_depth`: the LPJmL irrigation depth per cell, crop and year (`lon`, `lat`, `year`, `lpjml_crop`, `irrig_mm_irrigated`, `irrig_mm_rainfed`, `stand_frac_irrigated`, `stand_frac_rainfed`, `lpjml_version`, `lpjml_git_hash`, `lpjml_run`). • `lpjml_years`: the years the LPJmL run covers (default: the years of `crop_depth`). • `lpjml_grid`: the cells (`lon`, `lat`) the LPJmL run's output covers. Needed with `crop_depth` when an irrigated row's crop has no irrigated LPJmL stand in its cell; otherwise read from the run. • `regime_areas`: WHEP's rainfed and irrigated hectares (`lon`, `lat`, `area_code`, `item_prod_code`, `year`, `rainfed_ha`, `irrigated_ha`). • `country_grid`: the cell support the regime areas are built on when they are not supplied.
`config`	Optional list: `cru_dir` (default `WHEP_CRU_DIR`), `run_dir` (default `WHEP_LPJML_RUN_DIR`, else the pinned depth layer), `normal_period` (default `1991:2020`) and `max_rows` (default `2e6`, the classifier input rows per block of cells, about 1.3 KB each at peak).
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Details

Inputs:

- Climate comes from CRU TS: annual mean temperature, precipitation, PET and ground-frost days. CRU ships PET as a mean daily rate in mm per day, which is multiplied by the days of each month and summed over the year. CRU's frost count is synthetic: CRU derives it from the gridded minimum temperature. Years before the CRU record (1901) hold the 1901-1930 climatology, and years after it hold the climatology of its last 30 years. A cell of WHEP's crop land that holds no value of a CRU variable in any year (all four for a cell off CRU's land mask, mostly coastal and island cells; fewer for a cell CRU maps for some variables only) takes that variable's

whole record from its nearest CRU land cell by great-circle distance between cell centres, among the cells that hold it in every year read. One donor supplies every variable the cell lacks, and the cell keeps the variables it has; a variable with gaps is not lent but left to the classifier, which reports the years without it. The distance is `climate_fill_km`, and a message of class `whep_moisture_regime_nearest_climate` counts the cells by the variables they took. The cell's own water term, irrigation included, is then applied.

- Irrigation depth is LPJmL's applied irrigation per crop stand, pooled to the commodity balance item by WHEP's irrigated hectares. LPJmL's paddy water on its rainfed rice stand is not counted: rainfed rows get none.

Value

A tibble with one row per `lon`, `lat`, `area_code`, `item_cbs_code`, `water_regime` and `year` where WHEP has hectares of that regime, carrying the window-mean `mat_c`, `frost_days`, `prec_mm`, `pet_mm` and `irrig_mm`, `water_mm`, `moisture_ratio`, `tropical`, `moisture_regime` (`"wet"`, `"dry"` or `NA`), `method_moisture_regime`, `climate_fill_km` (the distance to the CRU cell whose climate variables a cell lacking some took, `NA` for a cell with all of its own), `method_irrig_depth`, `lpjml_crop` and the LPJmL vintage columns `lpjml_version`, `lpjml_git_hash` and `lpjml_run`, plus the polity columns below.

An irrigated row's irrigation is averaged over the window years in which WHEP irrigates that land, its climate over the whole window; a row irrigated in fewer than all of them carries `irrig_years_<n>of<m>` in `method_moisture_regime`. Where WHEP irrigates a crop the LPJmL run has no irrigated stand of in the cell, in a year the run covers, the depth is the crop's LPJmL depth averaged over the polity's cells that have the stand, weighted by WHEP's irrigated hectares, and `method_irrig_depth` says why: `lpjml_landuse_vintage` for a cell on the LPJmL grid (the run's land-use forcing differs from WHEP's there) or `no_lpjml_cell` for one outside it, with the suffix `_world_mean` where the polity has no such stand and the world's mean is used. A message of class `whep_moisture_regime_stand_gap` counts these rows and hectares by cause; its `diagnosis` field lists every case.

Irrigation depth stamps

`method_irrig_depth` holds the least direct method over the window years of the row's irrigation record, one of:

- `lpjml_airrig_stand`: the item's own LPJmL crop has an irrigated stand in the cell.
- `lpjml_crop_proxy`: another named LPJmL crop stands in for the item, as mapped in `regime_yield_crop_mapping` (fodder crops, for example).
- `lpjml_others_proxy`: the item sits on LPJmL's composite "others" stand of minor crops.
- `lpjml_landuse_vintage`: WHEP irrigates the crop in a cell of the LPJmL grid where the run has no irrigated stand of it, because the run's land-use forcing predates WHEP's current land use. The depth is the crop's mean over the polity's cells that have the stand, weighted by WHEP's irrigated hectares.
- `no_lpjml_cell`: the same fill for a cell outside the LPJmL grid.

- `lpjml_landuse_vintage_world_mean` and `no_lpjml_cell_world_mean`: the two fills above where the polity has no such stand that year, so the world's mean over the cells that have it is used.
- `rainfed_regime`: a rainfed row, zero by definition.
- `NA`: `water = "precip"`, where there is no irrigation term.

A stamp carries two optional additions. The suffix `_recycled_climate` follows any of the seven LPJmL-derived methods above when a window year precedes 1901, because LPJmL recycles the 1901-1930 weather before its forcing starts. The field `;outside_run_<n>of<m>_<fill>` follows it when `n` of the window's `m` years lie outside the LPJmL run: those years take the mean of the series' own window years inside the run (`<fill>` is `series_mean`) or, where it has none, its country's mean for the balance item (`country_mean`).

Moisture regime stamps

`method_moisture_regime` joins these fields with `;`, in this order:

- `ipcc2019_wet_dry`: the rule.
- The window: `annual`, `trailing_30` or `normal_<first>_<last>`.
- The water term: `precip` or `precip_irrig`.
- Optional `climatology_<first>_<last>_<n>of<m>`: `n` of the window's `m` years lie outside the CRU record and took the held climatology of the years `<first>` to `<last>`.
- Optional `irrig_years_<n>of<m>`: an irrigated row whose land is irrigated in `n` of the window's `m` years. Its irrigation is the mean of those `n` years and its climate the mean of all `m`.
- Optional `nearest_cru_cell=<col>[+<col>...]`: the cell lacks these CRU climate variables (`mat_c`, `frost_days`, `prec_mm`, `pet_mm`) and took them from its nearest CRU land cell, `climate_fill_km` away; its other variables are its own.
- Optional, on an unclassified row (`moisture_regime` is `NA`): `missing=<col>[+<col>...]`, the inputs whose window mean is unavailable, and `pet_not_positive`, a window-mean PET of zero. Nothing is zero-filled.

Errors

The function aborts, with these condition classes, when:

- `whep_moisture_regime_years`: `years` is `NULL`, empty or has a missing value.
- `whep_moisture_regime_config`: `config` has an unknown entry, or `max_rows` is not one positive number.
- `whep_moisture_regime_period`: `normal_period` is not a run of consecutive whole years.
- `whep_moisture_regime_columns`, `whep_moisture_regime_empty`, `whep_moisture_regime_key`, `whep_moisture_regime_climate`: an input table lacks a required column, has no rows, has a duplicated or missing key, or carries more than one climate for a cell-year; or no cell of the climate holds, in every year read, a variable that a cell lacks.

- `whep_moisture_regime_domain`: an input holds a negative precipitation, PET, frost count or irrigation depth, more than 366 frost days, a `water_regime` other than `"rainfed"` or `"irrigated"`, or irrigation on a rainfed row.
- `whep_moisture_regime_lpjml_grid`: `data$crop_depth` has stand gaps but no `data$lpjml_grid` is supplied.
- `whep_moisture_regime_stand_mapping`: an irrigated production item has no LPJmL crop band.
- `whep_moisture_regime_stand_no_mean`: a crop has no irrigated LPJmL stand anywhere that year, so there is no depth to fill from.
- `whep_moisture_regime_outside_run`: an irrigated series-year outside the LPJmL run has no depth of its own or of its country to fill from.
- `whep_absent_input`: precipitation, PET, the irrigated hectares or the irrigation depth of the irrigated rows is absent, missing or all zero.
- `whep_lpjml_irrig_depth_*`: the LPJmL irrigation depth layer rejects the regime areas, the run or its output cubes, or the pinned layer.

Messages of class `whep_moisture_regime_unclassified`, `whep_moisture_regime_area_years` and `whep_moisture_regime_nearest_climate` report rows left without a regime, window years the regime areas do not cover and cells that took climate variables from their nearest CRU cell.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's

year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

IPCC (2019), 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Volume 4: Chapter 3, Figure 3A.5.2 (p. 3.48); Chapter 11, Table 11.1, footnote 4 (p. 11.13).

CRU TS version 4.09 (files `cru_ts4.09.1901.2024.<var>.dat.nc`), Climatic Research Unit, University of East Anglia.

Examples

```
build_cell_moisture_regime(example = TRUE)
```

---

build_cell_polity *Assemble WHEP's cell-polity crosswalk with true grid-cell area.*

Description

Reads the cell-polity fraction table (`lon`, `lat`, `area_code`, `polity_frac`) and adds `cell_area_ha`, computed from latitude with the same 0.5-degree cell-area formula used across the package (see `build_grass_availability_lpjml()`). This assembles the `data$cell_polity` contract that every Module C function (e.g. `build_n_deposition()`, `build_human_n()`, `get_soc_climate_drivers()`) expects as a required input.

Usage

```
build_cell_polity(
  polity_fraction_path = NULL,
  area_key = c("grid", "polity_area"),
  version = NULL,
  example = FALSE,
  year = NULL,
  reporting_areas = NULL
)
```

Arguments

`polity_fraction_path`
: Optional path to a local parquet, overriding `Sys.getenv("WHEP_POLITY_FRACTION_PATH")` and the pin. Not used with `year`.

`area_key` Which area code the output is keyed on: `"grid"` (default, the table's own reporting-area codes) or `"polity_area"` (the polity_area_crosswalk bucket national tables are aggregated on).

`version` Pin version, passed to `whep_read_file()`. `NULL` takes the version frozen in whep_inputs. With `year`, the version of the `polycell_support` pin, passed to `read_polycell_support()`.

`example` If `TRUE`, return a small fixture instead of reading the pin, so the example runs offline.

`year` `NULL` (default) returns the year-invariant crosswalk. One year returns the year-aware support described in *Year-aware support*.

`reporting_areas` Required with `year`: the area codes carrying national data in `year`, in the code space `area_key` selects.

Value

A tibble with `lon`, `lat`, `area_code`, `polity_frac` and `cell_area_ha`. With `year` it also carries `cell_area_frac`, `polity_rule` and `method_cell_polity` (`"year_aware"`).

Which area code the grid is keyed on

The parquet is rasterized from present-day polygons through `inst/extdata/regions.csv`, so its `area_code` is a raw reporting-area code and **not** necessarily a polity_area_crosswalk `polity_area_code`: the bucket every polity-keyed national table in whep is aggregated on. Grid codes that are not a bucket cannot join to national data at all: the join is silently empty on both sides.

`area_key` selects which of the two the output carries. It is not a fallback: `"grid"` is the default, reproduces the parquet's own codes bit-for-bit, and warns naming the codes that cannot resolve; `"polity_area"` resolves each code to its bucket through polity_area_crosswalk and re-sums `polity_frac` within `(lon, lat, area_code)`, so a cell straddling two areas of the same bucket stays one row per bucket and each cell's fractions still sum to 1. It respects `options(whep.unfold_rest_of_world = TRUE)` (see `folded_reporting_areas()`), so the grid and the national tables agree about where a Rest-of-World member's rows belong.

Under `"polity_area"` the raw reporting code is **carried, not replaced**: the output gains `grid_area_code` holding the parquet's own code, joined with `+` where a cell's areas collapse into one bucket. So the fold this performs is recoverable at the join rather than baked into the grid — a derived key silently overwriting the raw one it came from is what whep#582 reports from the output side, and the same fold is what dropped Sudan's 40.8 M goats and doubled its sugar cane in the published production series (whep#563).

The output deliberately does **not** gain `polity_code` / `reporting_polity_*`. A bucket is not a polity: `999` holds up to 17 territories at once and `206` holds Sudan and South Sudan together, so no polity code string is recoverable from this year-less grid. Carrying that identity needs the cell x polity x validity-interval unit tracked by epic whep#458, not a column added here.

Where the table comes from

WHEP produces this crosswalk itself, in section 1b of `inst/scripts/prepare_spatialize_all.R`, from Natural Earth polygons and `inst/extdata/regions.csv`. It is therefore published as the `spatialize-cell-polity-fraction` pin alongside the nine other artefacts of that script, and the pin is what this function reads by default: no user has to run the producer, or hold the Natural Earth shapefile, to get the table WHEP's own runs use. `WHEP_POLITY_FRACTION_PATH` and `polity_fraction_path` are **overrides** for a local development build, in the shape `read_polycell_support()` already uses.

Vintage of the area vocabulary

The table's `area_code` values must all exist in the `regions.csv` the installed package carries, because that is the table its producer rasterizes through. A copy built through an older vintage keyed Ethiopia `62` and Sudan (former) `206` where today's lookup uses `238` and `276`, so adopting it deleted both countries from every consumer (whep#694). Such a table is **refused** with class `whep_stale_cell_polity_grid` rather than read, and the message names the producer re-run that rebuilds it. The check guards the override, which is the only route that can now go stale, but it runs on the pin too so that pinning an older `version` cannot reintroduce the deletion either.

Year-aware support

With `year` set, the support is not this crosswalk at all: it is the polycell support (`read_polycell_support()`) read at the validity interval covering `year`, so each cell carries the polity the grid holds that year. In 1961 the former-USSR cells carry the USSR (`228`), Czechoslovakia's carry `51` and Yugoslavia's `248`; FAOSTAT reports their national totals under those codes, which the year-invariant crosswalk has no cell for (whep#1196). Each code is resolved from `polity_code` through polity_area_crosswalk, and, under `area_key = "polity_area"`, folded onto its bucket.

Where the polity grid and FAOSTAT's reporting units disagree, a recorded mapping (`inst/extdata/polity_cell_` keys the cells, and every mapped row is labelled in `polity_rule`:

- `"aggregate_member"`: an aggregate reported by FAOSTAT whose members the grid carries separately takes the union of its members' cells in its reporting years (Belgium-Luxembourg `15` to 1999, Viet Nam `237` to 1974, Yemen `249` in 1961).
- `"contained_fold"`: a polity with no FAOSTAT row in the year folds into the reporting unit containing it (the Baltic and Azerbaijan SSRs into the USSR to 1990, the fifteen USSR successors into it in 1991, North Macedonia into `248` in 1991).
- `"constant_territory"`: Yemen `249` in 1962-1989, for which the grid carries no complete predecessor, is placed on modern Yemen's cells.

Some cells carry an aggregate over its members, or two claimants, so their polities' territory adds up to more than the cell. In such an overlapping cell, a polity with no area code, none of `reporting_areas`, or duplicating the container it folds into, is removed from the share denominator. The removed polycells are returned in the `"deduplicated"` attribute, and the overlapping cells where two reporting polities still share are returned in the `"overlap_kept"` attribute.

`polity_frac` is then the polity's share of the cell's measured **land**, the same basis as the carbon path's `cell_area_frac` (which is carried too, with the same value), rather than the crosswalk's subcell count. The land of a polity with no area code stays in the denominator, so it is never handed to a neighbour. `cell_area_ha` keeps the latitude formula of the year-invariant path.

Examples

```
build_cell_polity(example = TRUE)
```

`build_commodity_balances`
Build commodity balance sheets

Description

Construct commodity balance sheets (CBS) from raw FAOSTAT data. This is a convenience wrapper that chains the three pipeline steps:

1. `.read_cbs()` — read & reformat FAOSTAT CBS data.
2. `.fix_cbs()` — processing calibration, trade imputation, destiny filling, and final balancing.
3. `.qc_cbs()` — flag data-quality anomalies.

Usage

```
build_commodity_balances(
  primary_all,
  start_year = 1850,
  end_year = 2023,
  smooth_carry_forward = FALSE,
  example = FALSE,
  historical_data = NULL,
  format = c("long", "wide"),
  trade_recovery = .cbs_trade_recovery_choices(),
  trade_zero = .cbs_trade_zero_choices(),
  share_overflow = .cbs_share_overflow_choices(),
  negative_supply = .cbs_negative_supply_choices(),
  hist_trade_scale = .hist_trade_scale_choices(),
  export_share_overflow = .cbs_export_overflow_choices(),
  export_share_basis = .cbs_export_basis_choices(),
  seed_backcast = .cbs_seed_backcast_choices(),
  aggregate_proxy = .cbs_aggregate_proxy_choices(),
  unmatched_processing = .cbs_unmatched_proc_choices(),
  silk_basis = .silk_basis_choices(),
  silk_duplicate_use = .silk_duplicate_use_choices(),
  tobacco_leaf_use = .tobacco_leaf_use_choices(),
```

```
  nut_basis = .nut_basis_choices(),
  residue_backcast = .cbs_residue_backcast_choices(),
  .fixed_data = NULL
)
```

Arguments

primary_all	A tibble of primary production, as returned by build_primary_production().
start_year	Integer. First year to include. Default 1850.
end_year	Integer. Last year to include. Default 2023.
smooth_carry_forward	Logical. If TRUE, carry-forward tails are replaced with a linear trend. Default FALSE.
example	Logical. If TRUE, return a small hardcoded example tibble instead of reading remote data. Default FALSE.
historical_data	Optional harmonized historical CBS or production rows to add before the CBS historical extension. May be a data frame or a path to a parquet/csv file. CBS-shaped rows should provide year, value, one of area_code or polity_area_code, one of item_cbs_code or item_prod_code, and preferably element. Production-shaped rows without element are accepted as production when their unit is tonnes. **Rice supplied here is assumed to be on a paddy (rough-rice) basis** and is multiplied by the paddy-to-milled extraction rate, matching build_primary_production(); pre-divide by that rate if the series is already milled. Default NULL.
format	One of "long" (default) or "wide". "long" returns one row per element. "wide" pivots the elements into columns, adds the live-animal rows that the FAO sheet omits, and checks the supply-use identity. Both are the same dataset; "wide" is what the IO model and the extensions consume.
trade_recovery	One of "byproduct_residual" (default), "pass_through", "net_import" or "none", selecting what happens to a traded item the CBS has no row for. The trade record is joined onto the CBS, so it can only fill a row that already exists; "none" keeps that, the behaviour before whep#762, and an area whose balance sheet omits an item it imports loses that supply outright. The others first create the missing rows from the trade record, restricted to tonnes-denominated items (live-animal trade is in heads and arrives through get_livestock_cbs()) and to areas the CBS already covers in that year. A created row has no production and no stock change, so its domestic supply is its net import, split over the destinies by the item's world-average shares – the fallback every CBS key with no destiny of its own takes. They differ only on a net exporter, whose export exceeds its import and so has nothing to come from: "pass_through" creates the row with the export capped at the import (the rule FABIO applies to the rows it creates from trade), and "net_import" does not create it. "byproduct_residual" is "pass_through" except for Meat Meal (2112), a rendering by-product WHEP has no production source

for once FAOSTAT's balance for it ends in 2013: its full export is kept, export-only keys included, and the balance imputes production as export minus import, the residual rule FAOSTAT-era Meat Meal rows already get (whep#1450). Measured on a 2011-2015 build, that adds 2.29 Mt of Meat Meal production and export at 2014 (43 areas) and 2.51 Mt at 2015 against `"pass_through"`, and moves no other key. **Recovery moves published values.** Measured on a 2005-2015 build at 2010, `"net_import"` adds 1,038 keys and raises world imports by 8.72 Mt (0.57%), exports by 1.83 Mt and food by 3.07 Mt (0.07%). Protein supply rises from 28 to 63 g/cap/day in Singapore, 31 to 72 in Bahrain and 69 to 116 in Qatar, the first two leaving the "Under" nourishment class and Qatar moving to "Over"; FAOSTAT's own balance sheets, which cover Bahrain and Qatar only from 2019, give 105-112 and 111-114. Spain and the USA do not move. `"pass_through"` adds the net exporters on top: 68 more keys at 2010, raising world imports and exports by a further 1.04 Mt each, while the 2.16 Mt the record exports beyond those imports (1.76 Mt of it Meat Meal, whep#1450) is dropped. It moves no food supply: the largest change in protein supply against `"net_import"` is 0.0003 g/cap/day. `get_wide_cbs()`, `get_processing_coefs()` and `build_io_model()` take the same argument and pass it into the shared build chain, each method under its own cache slot, so a downstream build can be run either way.

`trade_zero` One of `"prefer_record"` (default) or `"keep"`, selecting what happens when the CBS carries a **zero** import or export and the trade record for the same `(year, area_code, item_cbs_code)` carries a positive quantity. The trade record is filled in with `dplyr::coalesce()`, which replaces only a missing value, so the zero used to stand and the trade was discarded. `"prefer_record"` takes the positive trade quantity instead, because such a zero is not an observation: measured at 2010, every one of them carries FAO flag `"I"` (imputed) or the legacy `"S"` (standardized), and neither food-balance vintage carries a single official `"A"` on a trade row. A non-zero CBS value is never overwritten and a zero trade record never overwrites anything, so the fill can only add trade. `"keep"` restores the pre-whep#866 behaviour. **The default moves published values** — at 2010 it raises 4,493 import keys by 9.70 Mt and 3,771 export keys by 10.27 Mt, moving 26,538 published rows over 180 areas; see `NEWS.md`. The conflict count is reported by every build under either setting.

`share_overflow`

One of `"report"` (default), `"clamp"`, `"drop"` or `"abort"`, selecting what happens when a pre-1962 destiny share exceeds 1 — a destiny larger than the `domestic_supply` it is apportioned from (whep#980). Measured on a real 1950–1965 build, 108 of 207,816 rows do: `other_uses` 70, `processing_primary` 19, `food` 15, `feed` 4. They are 1.03% of the 1961 `other_uses` mass and 0.17% of the `food` mass. The cause is not this arithmetic: 89 of them are FAOSTAT's own 1961 balances not closing (the non-food Commodity Balances, which carry tobacco, hides and skins, silk, wool and fibres as `other_uses` and which no better-ranked source overwrites), and the other 19 are hops in net-exporting years,

whose `processing_primary` is the whole production by construction — a trade ratio rather than an apportionment, which no setting can move, so the warning names that subset separately. `"report"` therefore keeps every value as measured and only warns, so it **moves no published value**; it names the count, the split by destiny and the three largest. `"clamp"` caps the share at 1, `"drop"` sets it to `NA` so the key is filled from a neighbouring year instead (and booked as 0 where the violating year is the only observation), and `"abort"` refuses to build. The share is an intermediate, not a published number: `.cbs_fill_destinies()` later rederives each destiny as `domestic_supply` times a share normalised to sum to one. Re-measured under the current defaults (`negative_supply = "floor"`, `hist_trade_scale = "correct"`), no published row carries a destiny above its supply or a negative use under any setting, and every difference between the settings sits at 1960 or earlier. Against `"report"`, `"clamp"` moves 766 rows (`other_uses` −0.91 Mt, `food` −1.00 Mt, `domestic_supply` −1.90 Mt) and `"drop"` moves 557 (`other_uses` −3.15 Mt, `domestic_supply` −3.18 Mt), against 15,898 Mt of 1950–1960 `other_uses` and 15,411 Mt of `food`: at most 0.02%. The +23 Mt and +825 Mt once recorded here were whep#1065's negative supplies, which the `"floor"` default now removes. The reporting default is the one that invents nothing.

`negative_supply`

One of `"floor"` (default), `"report"` or `"abort"`, selecting what happens when a pre-1962 row has no observed `domestic_supply` and the `production + import - export` reconstruction that replaces it comes out below zero (whep#1065). Every destiny of such a row is apportioned from that supply, so a negative one makes every destiny negative – including `other_uses`, which is not a quantity that can be negative.

Measured on a real 1950–1965 build, 151 rows reconstruct a negative supply totalling −1,015.70 Mt, all in 1950–1960. The cause is upstream: US tobacco 1951 carries a 117,504,000 t export against 728,949 t of production, 115,136,000 t of it `historical-trade-exports` item 831 (whep#1085). With `hist_trade_scale = "drop"` the count falls to 83 rows worth −6.87 Mt, some of which a stock draw can describe.

`"floor"` clamps the reconstruction at zero, the treatment an observed negative supply already receives, and so books the exported mass the supply cannot source as a stock withdrawal: `stock_variation` is recomputed downstream as `production + import - export - domestic_supply`. It leaves no negative mass anywhere in the published balance. `"report"` keeps the negative supply and its destinies as computed, which is what every build before this default did; against `"floor"` it moves 3,911 rows, all in 1950–1960, and the published balance then carries 1,155 negative masses worth −1,164.2 Mt, 147 of them `other_uses` worth −883.94 Mt (−3.9% of the 1950–1965 net `other_uses` total of 22,749 Mt), with `stock_variation` 916 Mt smaller in magnitude. `"abort"` refuses to build any range starting before 1961.

Neither `"floor"` nor `"report"` makes a 97× export physical; `"floor"` is the default because it keeps every published use non-negative and puts the

imbalance in the balancing item. Both stay visible: the build warns with the count, the total and the largest rows; the values WHEP estimated for such a key carry the `source "historical_fill_floored_supply"` (or `"historical_fill_negative_supply"` under `"report"`); and any negative mass that reaches the published balance, from this or any other cause, is reported by a separate warning of class `whep_negative_cbs_value`.

`hist_trade_scale`

One of `"correct"` (default), `"report"`, `"drop"` or `"abort"`, selecting what happens when a pre-1961 row of the `historical-trade-*` pins carries a quantity no mass unit can express (whep#1085). The screen bounds a single reporter's flow by the largest **world** flow FAOSTAT records for the same trade item, summed over reporters, over the FAOSTAT years the build already reads — a measured bound, not a chosen cap, because world trade in these commodities grew through the twentieth century. Re-measured on the real pins at 1850–2023 for whep#1117, 1,693 rows exceed it, carrying 3,874.4 Mt, 21.0% of the pins' whole 18,455.4 Mt; a further 10,243 rows have no FAOSTAT reference and go unchecked. **97.2% of the flagged mass is the USA**, whose block over roughly 1900–1960 is inflated by a factor of ten, interleaved with correct values in the same series (cotton lint 767 and raw sugar imports 162 alternate correct and ten-fold values year to year; whep#1117), and by far more on item 831, "Tobacco products nes", published at 115.1 Mt for 1951 — 159x the largest world flow of that item FAOSTAT has ever recorded and 32x the entire 1961 world tobacco crop. `"report"` keeps every value and only warns, so it **moves no published value** (verified: the screened read is `identical()` to the unscreened one); it names the count, the mass, the reporters and the three largest. `"drop"` removes the flagged rows, taking 3,874.2 Mt out of the historical trade input and with it the impossible pre-1962 exports behind whep#1065's negative `domestic_supply`. `"abort"` refuses to build. There is deliberately no clamp: the defect is in the pin's producer and no conversion factor recovers the true value, so a clamped tonnage would be a fabricated one.

`"correct"`, the default (a maintainer decision), judges each flagged row on its own evidence from the two pins and repairs only a proven ten-fold slip (whep#1117): a row is divided by 10 when that brings it within the world bound, when as published it exceeds the whole partner side of the pins (the opposite flow of every other reporter, same item and year) but divided by 10 does not, and when it sits one power of ten above the clean neighbours of its own series within 5 years (the window is assumed, unverified). A row no partner books, or still above the partner side after dividing by 10, is dropped as not a mass (item 831's signature); every other flagged row is dropped as unexplained. No factor other than 10 is ever applied. Re-measured on current `main` against FAOSTAT 1961–2023 (the bound a 1850–2023 build uses), 1,659 rows carrying 3,958.8 Mt are flagged: 175 rows (1,002.0 Mt as published, 100.2 Mt used) are divided by 10, 1,126 (2,662.3 Mt) are dropped as not a mass and 358 (294.5 Mt) as unexplained; 99.9% of the corrected mass is the USA. On a real 1950–1960 build it leaves the same 83 negative reconstructed supplies (−6.87 Mt)

as `"drop"`, against 151 (−1,015.70 Mt) under `"report"`, while keeping 103.4 Mt of repaired trade that `"drop"` discards. The output of every setting that builds carries a `hist_trade_scale_log` attribute (absent with `.fixed_data`): one row per flagged pin row, with the published and the used value, the evidence (`world_max`, `mirror`, `neighbour`), the class and the action.

The warning also carries a second, informational class: rows larger than any flow FAOSTAT records for the **same reporter**, item and element. That bound is far tighter — 19,375 pre-1961 rows carrying 9,154.0 Mt, against the world bound's 1,693 — and it is what makes whep#1117's ten-fold block visible at all. USA raw sugar imports (item 162) are the clearest case: 37 rows, 35–43 Mt over 1955–1959 against the pins' own 4.7 Mt at 1960 and FAOSTAT's 3.7 Mt at 1961, and **not one of them is above the world bound**, so today nothing reports them. It is **never dropped and never aborts**, under any setting, because it provably mixes two populations that no offline anchor separates: the pins' 1961 layer is FAOSTAT verbatim (all 18,531 overlapping rows agree to machine precision), FAOSTAT itself starts in 1961, so no pre-1961 row has a per-row anchor, and nineteenth-century Britain genuinely imported more flax fibre, cotton lint, linseed and cheese than modern Britain does. A ÷10 repair is a change to the pin's producer, not to this reader.

`export_share_overflow`

One of `"report"` (default), `"drop"` or `"abort"`, selecting what happens when the global export share the second processed-products round apportions a new product with exceeds 1 (whep#1086). The share is world `export / (production + import)` for the `(year, item_cbs)` key, multiplied by a country's newly created processed production, so a share above 1 books more export than that country produced. Measured on a real 1950–1965 build, 77 of 2,012 keys exceed 1 and the largest is 443 (Soyabean Cake 1956) — but **none of them is applied**: every one is pre-1961 and the round emits rows from 1961 on only, so `"report"` and `"drop"` give identical output on that range. **That does not hold after 2013** (whep#1177). On a real 2011–2023 build, 60 keys above 1 are applied, every one of them in 2014–2023 (oilseed cakes and molasses, up to 15.7 for Sesameseed Cake 2016): the round books 2,297 rows with a negative `domestic_supply`, −18.33 Mt in total, and the finished balance under `"report"` carries 2.4–4.4 Mt more `export` a year than under `"drop"`, 33.6 Mt over 2014–2023, which `"drop"` books mostly as `feed`. Of the 77, 50 have no world production in the denominator at all (the oils and cakes, whose production is what this round is about to create) and the other 27 are the `historical-trade-exports` defect of whep#1085. `"report"` keeps every share as measured and names the count, the largest, and how many are actually applied. `"drop"` sets a violating share to `NA`, which is booked as no export at all. `"abort"` refuses to build. There is deliberately no clamp, unlike `share_overflow`: a destiny cannot exceed the supply it is apportioned from, so 1 is a true bound there, while here the denominator is incomplete and capping at 1 would book a country's whole processed output as export. Those figures are for the step-4

denominator, `export_share_basis = "snapshot"`; on a 2014–2023 build the default `"current"` leaves 2 shares above 1 (Abaca 2021, Fish, Liver Oil 2017) and applies neither.

`export_share_basis`

One of `"current"` (default), `"processed"` or `"snapshot"`, selecting which world balance the second processed-products round reads its export share `export / (production + import)` off (whep#1143). The share is multiplied by a country's newly created processed production, so its denominator must contain that kind of production. `"current"` reads the balance the round runs on and adds its rows to: processed production from the first round, trade after imputation. It is also the self-consistent choice, since a share `E / S` leaves the world share unchanged once the round's own production and export are added. `"processed"` reads the same production with trade as read, before imputation. `"snapshot"` reads the balance before any processed production exists, which is what every build before whep#1143 did: wherever FAOSTAT reports no production of a processed product — all of them before 1961, and the new food balance sheets' oilseed cakes and molasses from 2014 — its denominator is world import alone.

The default moves published values, at 2014–2023 almost entirely. Measured on real 2014–2023 and 1961–1965 builds of main after whep#1242 against `"snapshot"`: at 2014–2023, 16,526 rows change, world `export` falls 36.1 Mt summed over the ten years (42.8 Mt gross) and `domestic_supply` rises by the same, landing 29.3 Mt on `feed`; applied shares above 1 fall from 60 to 0 (largest 15.7, Sesameseed Cake 2016). World Sesameseed Cake export at 2020 goes from 466 kt to 1.6 kt against 0.4 kt of world import, Oilseed Cakes, Other from 3.17 Mt to 2.13 Mt against 1.85 Mt. The step-4 sheet carries no cake or molasses production there at all. At 1961–1965 the change is 13 kt gross. `"processed"` moves the same cakes (export −39.3 Mt at 2014–2023) and differs from `"current"` only on items whose trade step 7 alone supplies: DDGS (+2.37 Mt of export under `"current"`) and Sugarbeet pulp (+0.60 Mt), for which `"processed"` books no export at all. Production is identical under all three.

`seed_backcast`

One of `"area_rate"` (default) or `"production_share"`, selecting what the pre-1962 seed back-cast reads its rate off and spends it on (whep#699). The fill carries a rate along the year axis: a rate is read from the years that report `seed`, interpolated and extrapolated into the years that do not, and multiplied back out. It yields tonnes only if the quantity it is spent on is the quantity it was divided by.

`"area_rate"` is tonnes of seed per hectare harvested, `seed / area_ha` spent as `area_ha * seed_rate`, and is the default: a seeding rate is an agronomic quantity that holds while yields change, so it is the ratio worth carrying across decades. `"production_share"` is tonnes of seed per tonne of output, `seed / production` spent as `production * seed_rate`; it moves with yield, but reaches keys the production build gives no harvested area.

The default moves published values, because neither is what shipped: a rate defined per hectare used to be spent on production, giving `t x t/ha`.

Measured on a real 1850-2023 build, pre-1962 `seed` falls from 38.62 Gt to 7.13 Gt while the number of keys carrying one rises from 87,882 to 118,332, total tonnage moves -0.934% and pre-1962 tonnage -2.716%, and no 1962-or-later value changes at all. World seed at 1960 goes from 680.3 Mt to 77.7 Mt against the 126.1 Mt FAOSTAT reports for 1961, and `check_series_jumps()` on `seed` over 1950-1970 falls from 582 jumps to 176, of which the 1960-1961 seam holds 3 rather than 295. Under `"production_share"` pre-1962 `seed` is 5.22 Gt, total tonnage moves -1.023%, and the seam holds 5 jumps.

`aggregate_proxy`

One of `"member_sum"` (default) or `"none"`, selecting what Rest of World (area code 999) grows on in the pre-1962 back-cast (whep#724). Every other area grows its `food`, `other_uses` and `processing` on its own population and its `feed` on its own agricultural land. No ISO3 code names Rest of World, so no proxy reaches it directly. Its rows are crop residues from the `crop_residues` pin, which books all of FABIO's Rest-of-World members on one label.

`"member_sum"` grows it on the summed population and agricultural land of those members, the territory its rows describe, and grows its residue along the summed crops of those members (see `residue_backcast`). This holds whatever `options(whep.unfold_rest_of_world)` is set to. `"none"` gives it no proxy, so its series starts in 1961.

The default moves published values, for area 999 before 1961 only. Measured on a real 1955-1965 build with the default `residue_backcast`, 73 rows differ from `"none"` and no other value changes: 75.0 Mt of residue `production` and `domestic_supply` over 1955-1960 (11.7 Mt in 1955, 13.3 Mt in 1960, then 21.2 Mt in 1961), of which 44.1 Mt is `other_uses` and 31.0 Mt `feed`. That raises world residue use by 0.44-0.46% in those years. The step at 1961 is the members' crops: six of the eleven report no production before 1961. Under `"none"` area 999 carries no rows before 1961.

`unmatched_processing`

One of `"other_uses"` (default), `"processing"` or `"redistribute"`, selecting where a `processing` destiny goes when its item has no pathway in cb_processing, so no processed product exists for the mass to become (whep#781). Measured on a real 2010 build this is 15.92 Mt over 25 items, 10.10 Mt of it raw sugar.

`"other_uses"` books it on `other_uses`, the destiny the processing shortfall of an item that *has* a pathway already goes to, and leaves `food`, `feed` and `export` as FAOSTAT reported them. `"processing"` keeps FAOSTAT's row as a terminal destiny; the balance still closes, but `build_io_model()` folds processing that no product consumes into `food`. `"redistribute"` is the behaviour before whep#781: the mass is split pro rata over `food`, `feed`, `other_uses` and `export`.

The default moves published values. Against `"redistribute"` at 2010, world `food` falls 10.82 Mt, `feed` 0.56 Mt and `export` 3.05 Mt, while `other_uses` rises 14.43 Mt and `domestic_supply` 3.05 Mt, because the export share had moved domestic processing out of the country. Food then sits within 0.6% of FAOSTAT's for every affected item but coconut

oil, against 6.0% for raw sugar, 14.4% for cottonseed oil and 40.9% for ricebran oil before. Which destiny is right is open — see whep#781 — and a sourced pathway per item would supersede all three.

`silk_basis` One of `"cocoon"` (default), `"raw_silk"` or `"mixed"`, selecting the mass basis of the Silk balance from 2014 on (whep#1251). Silk is a FAOSTAT chain – reelable cocoons (1185) reeled into raw silk (1186), plus silk waste (1187) – that the non-food Commodity Balances report link by link, each in its own mass, and that WHEP maps onto one item. Summed unconverted, with the cocoons sent to reeling dropped as a chain transfer, the reeled cocoons were left as stock build-up: 267 kt of `stock_variation` against 536 kt of production at 2020, 391 kt against 517 kt at 2021, measured on a real 2019-2021 build.

`"cocoon"` counts production once, as cocoons, books the cocoons reeled at FAO's own cocoon mass, and converts only the raw silk that crossed a border or a stock, dividing by a raw-silk extraction rate of 0.16 – the midpoint of the 12-20% of fresh cocoon weight in Lee (1999), *Silk reeling and testing manual*, FAO Agricultural Services Bulletin 136; FAO's Technical Conversion Factors carry no silk entry. `"raw_silk"` is the same balance multiplied by that rate, so every Silk quantity depends on it. `"mixed"` keeps each link's own mass and books the reeled cocoons as `other_uses`, the convention of FAO's pre-2014 aggregate item 2747: the balance closes but raw silk is counted twice, once as the cocoons it was reeled from. Silk waste keeps its own mass under every setting.

Measured at 2020 on that build, production is 443 / 71 / 536 kt (cocoon / raw_silk / mixed) and `stock_variation` -170 / -32 / -164 kt, of which -157 kt was one FAOSTAT record under every setting: China mainland's 2020 cocoons are booked both as `Processed` and as `Other uses` (FAOSTAT's own `Residuals` is -156,938 t). Under every setting a cocoon `Other uses` that equals the row's `Processed`, where FAOSTAT's `Residuals` is minus that amount, is now dropped as a second booking (whep#1281); world Silk `stock_variation` at 2020 moves from -170 kt to -13 kt under `"cocoon"`. No non-Silk row moves. Years before 2014 come from the aggregated old Commodity Balances, which carry no link breakdown, and are unchanged, so under `"cocoon"` the 2013-2014 seam steps by roughly the raw silk production.

`silk_duplicate_use`

One of `"residual"` (default) or `"equal_only"`, selecting which silk cocoon uses from 2014 on are read as a second booking of the same cocoons (whep#1456). FAOSTAT's own balance of a cocoon record ends in a `Residuals` element; where it is minus one of the two uses, `Processed` or `Other uses`, FAOSTAT's balance closes on the other use alone. `"residual"` drops the use the residual cancels and keeps the one that closes the balance. `"equal_only"` drops it only when the two uses are equal (whep#1281) and otherwise leaves the record as published, so the cancelled use stays in Silk `other_uses` and the balance closes through a Silk stock withdrawal.

In the pinned `faostat-cbs-new` the two differ for Viet Nam 2014-2015, whose 45,073 and 43,620 t processed exceed its 6,761 and 6,543 t of production, Indonesia 2015-2018 and Italy 2019 (also more processed than

supplied), Afghanistan 2015-2018 (`Processed` dropped) and China mainland 2022-2023 (18,721 t of `Other uses` dropped on top of processing that already equals supply). Only Silk rows of those country-years move.

`tobacco_leaf_use`

One of `"as_published"` (default) or `"one_to_one"`, selecting how the Tobacco balance from 2014 on treats leaf manufactured into products (whep#1390). The non-food Commodity Balances book the leaf (826) that goes into a factory as leaf `other_uses`, and the cigarettes, cigars and other manufactured tobacco (828, 829, 831) made from it are used or exported again. Their production is not booked as supply (whep#1276), so the summed uses exceed supply by about the manufactured output, and the balance closes through a stock withdrawal with no stock behind it – about 54 kt a year for the Netherlands and Ukraine over 2019-2021.

`"as_published"` keeps FAOSTAT's leaf `other_uses`, phantom withdrawal included. `"one_to_one"` subtracts the products' production from the leaf `other_uses`, floored at zero, assuming one tonne of leaf per tonne of product – assumed, unverified: no sourced leaf content per tonne of product was found, and a cigarette also holds paper and filter. Where the products outweigh the leaf use (Ukraine 2019: 56 kt against 29.7 kt) the remainder stays as stock change. Only Tobacco rows from 2014 on move.

`nut_basis`

One of `"rebased"` (default) or `"as_published"`, selecting how the trade of FBS item 2551 `Nuts and products` is taken from FAOSTAT's new food balance sheets, 2010 on (whep#1495). Their nut trade is not one mass basis: at 2010 world imports are 13.57 Mt against 7.69 Mt of world exports (1.76x; 1.65x at 2022), and the excess goes on into food, 19.47 Mt against 14.29 Mt of nuts the world grew in shell, with about the old vintage's protein. Regressed on species-level trade, importers convert kernels and prepared nuts to in-shell equivalent with factors 2.5-6x their exporters' (cashew kernels 9.8 vs 3.8), so one flow lands on two bases. The old vintage closes (6.45 vs 6.51 Mt).

`"rebased"` multiplies each area's import and export by its median old/new ratio over 2010-2013, clamped to `[0.2, 1]` – the old vintage's level with the new one's change over time, as the palm-kernel fix already joins them – and takes the net trade removed out of food and domestic supply, so every row keeps its supply-use identity and production and stock change do not move. A row whose net cut is not positive or exceeds its food is left as published. The moved rows are labelled `"FAOSTAT_FBS_New_rebased"`, and the ratio is read from the pins whatever years are built. `"as_published"` keeps FAOSTAT's figures, the behaviour before whep#1495; the old vintage is scaled to whichever level is chosen, so 1961-2009 follows it. **The default moves published values**, all of them in item 2551. Measured on a real 2009-2016 build at 2010, world imports fall from 13.85 to 7.16 Mt and exports from 7.83 to 6.51 Mt (imports/exports 1.77x to 1.10x; 1.61x to 1.02x at 2016), and food from 19.79 to 14.43 Mt against 14.60 Mt of production; the 5.37 Mt drop is 0.11% of the 4,692 Mt of all-item food. Production and stock change do not move from 2010 on. FBS's own 2551 protein over the pin's food rises from 45.6 to 62.9 g/kg, against FAO's in-shell basket of 64.7 that `build_food_supply()` applies.

`residue_backcast`

One of `"own_method"` (default), `"production_ratio"` or `"area_rate"`, selecting what crop residue (Straw, Other crop residues, Firewood) grows back along before the first year of the `crop_residues` pin, 1961 (whep#1508). Each (area, crop, residue item) keeps the ratio of its 1961 gross residue to a basis taken from the crop's primary production, and is then recovered and split into `production`, `feed` and `other_uses` as every 1961+ row is, so production equals feed plus other uses in every year. Rows carry the source `"historical_residue_<method>"`.

`"own_method"` grows each row along the basis its 1961 value was built on: residue computed from yield (cereals, and the non-cereals with an IPCC Table 11.2 line, whep#1561) along `calculate_crop_residues()` applied to the year's production and harvested area, the estimator that computes it from 1961 on, and every other row along its crop's production, since the pin's remaining residue is a residue:product ratio. `"production_ratio"` grows every row along production, holding residue per tonne of crop; it puts 1955-1960 residue computed from yield 1.5-5.7% below the estimator. `"area_rate"` grows every row along harvested area, holding residue per hectare; it puts it 0.3-2.6% above.

The default moves published values, for residue items before 1961 only. Before it, the extension grew residue uses along population and agricultural land and gave the residue no production, so the whole use was booked as a stock draw. Measured on a real 1955-1965 build, world residue production in 1955 / 1960 goes from none to 2,301 / 2,486 Mt (`"production_ratio"` 2,209 / 2,454, `"area_rate"` 2,392 / 2,506), against 3,038 Mt in 1961, and residue `stock_variation` from -2,681 / -2,906 Mt to -375 / -407 Mt under all three. What remains is the Soviet Union, Yugoslavia, Czechoslovakia and Belgium-Luxembourg, whose crop production does not reach before 1961 either (whep#102): their residue keeps the population and land fill, like their crops. World `stock_variation` over all items moves from -3,057 / -3,222 Mt to -751 / -723 Mt. No 1961-or-later value and no other item moves.

`.fixed_data` Optional tibble with the same structure as the output of the internal `.read_cbs() |> .fix_cbs()` steps. When supplied, `primary_all` is ignored and the pipeline skips directly to `.qc_cbs()`. Default `NULL`.

Value

For `format = "long"`, a tibble with columns: `year`, legacy numeric `area_code`, numeric `polity_area_code`, `reporting_polity_code`, `reporting_polity_name`, `reporting_polity_has_geometry`, `item_cbs_code`, `element` (e.g. `"production"`, `"import"`, `"food"`), `value`, `source`, and `fao_flag`. For `format = "wide"`, the elements become one column each, `stock_variation` is split into the non-negative `stock_addition` and `stock_withdrawal`, and `domestic_supply` is total use excluding `export`. A `unit` column says each row's denomination: `"tonnes"`, or `"heads"` for the live-animal rows (see `get_wide_cbs()`).

`fao_flag` is FAOSTAT's own observation-status code for the value, taken from the source that `source` names (`"A"` official, `"E"` estimated, `"I"` imputed, `"S"` standardized, `"SD"`, `"X"`). It is `NA` wherever the number is not one FAOSTAT published under a flag: a WHEP-

derived row (the processing pathway, the destiny gap-fills, the pre-1961 historical extension), a row whose source carries no flag, and a row summed or averaged from parts whose flags disagree. The flag is a claim about the value, so it is dropped rather than guessed when the parts do not agree. Rows sourced from `"FAOSTAT_prod"` are `NA` today because `build_primary_production()` does not carry the flag out of the production pin.

Examples

```
build_commodity_balances(example = TRUE)
```

`build_constant_territory_series` *Build a constant-territory time series for a reference year's boundaries*

Description

Estimates a time series of a quantity over a **fixed** set of territorial boundaries — the polities active in `ref_year` — from data reported under the *changing* historical boundaries of each data year.

Country borders change over time, so there is no raw constant-territory series: a 1900 figure for "Austria-Hungary" is not a figure for present-day Austria. This function estimates one by spatial reallocation (dasymetric areal interpolation):

1. For each data `year`, the value reported by each source polity is spread over **that polity's own extent for that year** across a regular grid, weighted by a `covariate` density (e.g. gridded cropland or population; uniform = plain areal weighting).
2. The grid is then re-aggregated to the `ref_year` target boundaries: a target's estimate is the sum of grid mass falling inside it.
3. Target territory **not covered** by any source with data in that year is *imputed* — its grid cells still carry covariate mass, so they are filled at a donor intensity (value per unit covariate) rather than left at zero. The fraction of a target's covariate mass that had to be imputed is reported as `imputed_share`, an honest confidence signal.

The estimate is only as good as the `covariate`: supply the same gridded surface used elsewhere in WHEP spatialization (cropland for crop output, population for demographic series, livestock density for animals). With `covariate = NULL` the method reduces to area-weighted areal interpolation.

Usage

```
build_constant_territory_series(
  data,
  ref_year,
  polities = NULL,
  covariate = NULL,
  resolution = 25000,
```

```
  donor = c("regional", "none"),
  crs_equal_area = 6933,
  max_cells = 2e+06,
  verbose = TRUE
)
```

Arguments

`data`	A data frame of reported values with columns: • `year`: integer data year. • `polity_code`: the source polity that reported the value (must be active in `year` and carry a polygon). • `value`: numeric value (summed if a polity appears more than once).
`ref_year`	Integer. Target boundaries are the polities active in this year, under the validity convention described for `polities`: a polity does not answer for the year its successor takes over. The same reading selects the sources of each data year.
`polities`	An `sf` of polity polygons with `polity_code`, `start_year`, `end_year` and geometry, plus an **optional** `wiki_status` whose effect is described below. Defaults to `get_polity_geometries()`, which supplies it. `start_year` is inclusive; `end_year` is **exclusive at a succession** and **inclusive at the open end**. So 2014 resolves to `"RUS-2014-2025"`, never to `"RUS-1991-2014"`, while 2025 still resolves to `"RUS-2014-2025"` because nothing succeeds it. An interval counts as open when it ends on the last year the supplied table covers and no later-starting interval of the same polity follows it. Where the table still carries overlapping intervals for one polity, the interval starting on the resolved year wins, then the latest-starting one, and finally – **only when `wiki_status` is supplied** – a live interval beats a `"retired"` or `"superseded"` one. That last key is the one behavioural difference between the two call styles, and it is stated here rather than left implicit. **Supplying only the three required columns is legal and keeps the older, status-blind ordering**, under which a shared start year is decided by nothing at all and an interval upstream has already replaced can take the year from the one that replaced it. On the shipped polities snapshot that is 1,344 polity-years across 10 polities, of which one moves territory rather than only a label: 1913 and 1914 resolve to `"MNE-1913-1915"` (0.9923 Mha) instead of `"MNE-1913-1918"` (1.5893 Mha), a Montenegro 37.6% too small. Pass `wiki_status` – as the default does – to get the live interval. `wiki_status` never selects which years an interval covers; it breaks a tie between two intervals that both cover the resolved year.
`covariate`	`NULL` (uniform density, i.e. area weighting) or a function `function(centroids_sf, year) -> nu` returning a non-negative density per grid-cell centroid (centroids are supplied in `crs_equal_area`).
`resolution`	Grid cell size, in metres of `crs_equal_area`. Default 25000 (25 km). Smaller is more accurate but slower.

`donor` Gap-imputation rule: `"regional"` (default) fills uncovered target cells at the region-wide value-per-covariate intensity of the sources with data that year; `"none"` leaves them at zero (covered-only).

`crs_equal_area` EPSG code of an equal-area CRS used for gridding and areas. Default 6933 (NSIDC EASE-Grid 2.0 Global).

`max_cells` Safety cap on grid cells per year (default 2e6). Aborts if the source/target extent would exceed it (usually a stray continent-scale target); restrict `polities`, coarsen `resolution`, or raise this.

`verbose` Logical; emit progress/warnings.

Value

A tibble, one row per (`ref_year`-target, data `year`):

- `target_polity_code`, `year`
- `value`: constant-territory estimate (`covered + imputed`)
- `covered`: mass from cells overlapping a source with data
- `imputed`: mass added for uncovered cells
- `imputed_share`: covariate fraction imputed (0 = fully observed)
- `n_sources`: number of source polities contributing that year
- `unallocated`: total reported value from sources that could not be placed on the grid that year (constant within a year; 0 when all were placed). Kept out of `covered`/`imputed` so it is neither smeared nor lost.

Examples

```
# Self-contained toy: two adjacent square polities. Only "P1" reports a
# value in 1900, so when the series is rebuilt onto the boundaries active in
# `ref_year` 2000 (both polities), "P2" is imputed from "P1"'s intensity.
# `sf` is a suggested dependency, so the example runs only when it is there.
if (requireNamespace("sf", quietly = TRUE)) {
  make_square <- function(xmin, ymin, side) {
    sf::st_polygon(list(rbind(
      c(xmin, ymin),
      c(xmin + side, ymin),
      c(xmin + side, ymin + side),
      c(xmin, ymin + side),
      c(xmin, ymin)
    )))
  }
  polities <- sf::st_sf(
    polity_code = c("P1", "P2"),
    start_year = c(1800L, 1800L),
    end_year = c(2025L, 2025L),
    geometry = sf::st_sfc(
      make_square(0, 0, 2),
      make_square(2, 0, 2),
      crs = 4326
```

```
  )
)
reported <- tibble::tibble(
  year = 1900L,
  polity_code = "P1",
  value = 100
)
build_constant_territory_series(
  reported,
  ref_year = 2000,
  polities = polities,
  resolution = 50000,
  verbose = FALSE
)
}
```

`build_critical_n_binding` *Derive the binding critical-nitrogen threshold per cell.*

Description

Identifies, for every 0.5-degree cell, the impact threshold whose critical nitrogen surplus is lowest among the three threshold-specific surfaces of Schulte-Uebbing et al. (2022): atmospheric deposition (`"de"`), groundwater nitrate (`"gw"`) and surface-water eutrophication (`"sw"`). That threshold is the one that binds. The archive's own threshold-exceedance map (`read_critical_n()` with `var = "threshold_exceedance"`) is a different quantity: it records which thresholds the 2010 surplus exceeds.

`binding_threshold` is `"deposition"`, `"groundwater"`, `"surface_water"`, a two-way tie, `"yield_potential_cap"` or `"non_agricultural_floor"`. A two-way tie is recorded explicitly by joining both tied thresholds with `+` in that fixed order, for example `"groundwater+surface_water"`. Ties are exact equalities of the deposited values: on the real archive no surface lies within 1e-6 kg N/ha of the cell minimum without equalling it, so a tolerance would change nothing. A cell missing from any of the three surfaces gets `NA`.

A cell where all three critical surpluses are equal has no environmental threshold that binds on its own. Schulte-Uebbing et al. (2022, Nature 610, Methods) set such a threshold-independent value by one of two rules, and the label names which one:

- `"non_agricultural_floor"`: all three thresholds are exceeded (each threshold-specific exceedance of the critical surplus is positive) or the tied critical surplus is negative. The source's "Aggregation to regional and planetary boundaries" section states: "Where N losses from non-agricultural sources alone exceeded thresholds, critical N inputs from fertilizer and manure were set to zero." The critical input is then the fixation and deposition left over, the same for every threshold.
- `"yield_potential_cap"`: every other tie. Step 4 of the source's Methods reads: "for areas with no threshold exceedance, cut off critical inputs and surplus at a maximum value, set to the input level required to obtain crop yield potentials", with `Nin(crit,max) = Nup(Yp) / NUE(act)`, where `Nup(Yp)` is crop nitrogen uptake at potential yield and `NUE(act)` the current nitrogen use efficiency, "capped ... at 0.8".

A tie whose critical surplus is not negative and whose exceedance is missing on any threshold cannot be assigned to either rule and gets `NA`. On the real archive the three critical inputs are identical in every tied cell. The two labels count 293 floor and 8,895 cap cells of 28,573 for `"ara"`, 293 and 9,138 of 28,881 for `"all"`, and 0 and 1,727 of 11,740 for `"igl"`. The split matches the archive's threshold-exceedance map exactly: every floor cell carries code 8 and every cap cell code 1. The 293 floor cells include 3 with a negative critical surplus (minimum -21.78 kg N/ha), and their median critical input is 38.7 kg N/ha against 90.7 for the `"all"` cap cells.

When the deposited minimum-of-all-media surface (`"mi"`) is supplied, `binding_matches_mi` reports whether it equals the lowest of the three threshold-specific surpluses. Where it does not, the relation between `"mi"` and the three surfaces is undetermined: `"mi"` differs from `min(de, gw, sw)` there and the source text does not explain why. On the real archive this happens in 1,623 of 28,573 cells for `"ara"` (maximum gap 159.5 kg N/ha), 1,540 of 28,881 for `"all"` (159.5 kg N/ha) and 1,480 of 11,740 for `"igl"` (479.1 kg N/ha), with `"mi"` above the minimum in some cells and below it in others. `binding_threshold` still names the argmin of the three surfaces in those cells.

Usage

```
build_critical_n_binding(
  critical = NULL,
  exceedance = NULL,
  land_use = c("all", "ara", "igl"),
  dir = NULL,
  example = FALSE
)
```

Arguments

`critical`	Optional named list of `read_critical_n()` critical-surplus layers (`var = "critical_n_surplus"`) with elements `de`, `gw` and `sw`, and optionally `mi`, each stamped with its own threshold and with `land_use`. When `NULL` (default) the four layers are read from the archive with `read_critical_n()`, together with `exceedance`.
`exceedance`	Named list of `read_critical_n()` exceedance layers (`var = "exceedance"`) with elements `de`, `gw` and `sw`, used only to tell the two kinds of three-way tie apart. Required when `critical` is supplied; read from the archive when both are `NULL` (default).
`land_use`	Land-use scope: `"all"`, `"ara"` or `"igl"`, as in `read_critical_n()`. Supplied layers must carry this scope.
`dir`	Optional archive directory passed to `read_critical_n()` when `critical` is `NULL`.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with one row per cell: `cell_id`, `lon`, `lat`, `critical_land_use`, the three threshold-specific critical surpluses `critical_de_kgn_ha`, `critical_gw_kgn_ha`, `critical_sw_kgn_ha`

(kg N per hectare per year), their exceedances `exceedance_de_kgn_ha`, `exceedance_gw_kgn_ha`, `exceedance_sw_kgn_ha`, the minimum critical surplus `binding_critical_kgn_ha`, the `binding_threshold` label, the deposited `critical_mi_kgn_ha` and the logical `binding_matches_mi` (both `NA` when `mi` is not supplied).

Examples

```
build_critical_n_binding(example = TRUE)
```

---

build_crop_land_extension

Build per-crop physical cropland extension.

Description

Convert gridded crop *harvested* area into per-crop *physical* land area and aggregate it to commodity-balance items, producing a land extension keyed by `(year, area_code, item_cbs_code)` for the FABIO footprint model.

The gridded land-use pipeline (`build_gridded_landuse()`) distributes FAOSTAT harvested area across grid cells; per-crop totals therefore conserve to harvested area, which over-counts multi-cropped land and under-counts fallow. This function turns that harvested area into physical occupied land:

- `"cropland_apportion"` (default): within each cell, the cell's physical cropland (`cropland_ha`, from LUH2) is split across crops in proportion to their share of the cell's harvested area. Per-crop physical area then conserves to physical cropland rather than to harvested area, capturing both double-cropping (scaled down) and fallow (resting land charged to the crops the rotation supports) at the resolution of the grid.
- `"intensity_divide"`: each crop's harvested area is divided by the cell multi-cropping intensity (`mc_rainfed`, `mc_irrigated`). Requires `multicropping`.

Unlike a single country-level cropping-intensity factor applied uniformly to every crop, both methods distribute physical cropland by the actual spatial pattern of each crop.

Coverage note: with `"cropland_apportion"` the per-country crop total is bounded by the LUH2 `cropland` layer, which can under-represent perennial or plantation crops (e.g. oil palm, rubber) classified outside cropland; such crops may receive less land than a harvested-area baseline implies.

Usage

```
build_crop_land_extension(
  gridded_crops,
  gridded_cropland,
  items_prod_full = whep::items_prod_full,
  method = c("cropland_apportion", "intensity_divide"),
  multicropping = NULL
)
```

Arguments

`gridded_crops`	Tibble of gridded crop harvested area, the crop-level output of `build_gridded_landuse()` (built without CFT aggregation). Must have columns `lon`, `lat`, `year`, `area_code`, `item_prod_code`, `rainfed_ha`, `irrigated_ha`.
`gridded_cropland`	Tibble of physical cropland per cell. Must have columns `lon`, `lat`, `year`, `cropland_ha`.
`items_prod_full`	Crosswalk from production items to commodity-balance items. Defaults to items_prod_full. Must have columns `item_prod_code` and `item_cbs_code`.
`method`	Physical-area conversion method. One of `"cropland_apportion"` (default) or `"intensity_divide"`.
`multicropping`	Tibble of per-cell multi-cropping intensity, required for `method = "intensity_divide"`. Must have columns `lon`, `lat`, `mc_rainfed`, `mc_irrigated` (and optionally `year`).

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (physical land area in hectares), and `method_land` (the chosen method), plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's

year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
gridded_crops <- tibble::tribble(
  ~lon, ~lat, ~year, ~area_code, ~item_prod_code, ~rainfed_ha, ~irrigated_ha,
  0.25, 50.25, 2000L, 1L, 15L, 600, 0,
  0.25, 50.25, 2000L, 1L, 27L, 200, 0,
  0.75, 50.25, 2000L, 1L, 15L, 400, 0
)
gridded_cropland <- tibble::tribble(
  ~lon, ~lat, ~year, ~cropland_ha,
  0.25, 50.25, 2000L, 1000,
  0.75, 50.25, 2000L, 500
)
items <- tibble::tribble(
  ~item_prod_code, ~item_cbs_code,
  15L, 2511L,
  27L, 2805L
)
build_crop_land_extension(gridded_crops, gridded_cropland, items_prod_full = items)
```

---

build_crop_soil_n2o_extension

Build the crop/soil N2O extension.

Description

Estimate IPCC 2019 Tier 1 nitrous-oxide emissions from nitrogen applied to managed soils, as a footprint extension keyed by `(year, area_code, item_cbs_code)` in kilograms of carbon-dioxide equivalent (CO2e). This is the soil-N2O analogue of `build_livestock_ghg_extension()` and feeds `build_footprint()` / `compute_footprint()` the same way.

Three nitrogen inputs to soil are included:

- **Synthetic fertiliser** (F_SN): FAOSTAT reports it only as a country total (tonnes N per `area_code` per year), so it is allocated to crops by the Coello 2025 rate-weighted, FAOSTAT-conserving crop share (default; the national total is preserved), or by harvested-area share when `synthetic_method = "area_share"`.
- **Applied manure** (F_ON): FAOSTAT "Manure applied to soils (N content)" country total, allocated to crops by harvested area (Coello is a synthetic-N rate basis only).
- **Crop residues** (F_CR): the dry matter of above-ground residues returned to soil (`value_dm` from `get_primary_residues()`, net of the removed fraction) times the crop's residue nitrogen content (IPCC 2019 Table 11.1a).

Both country totals are read under raw FAOSTAT `Area Code` values and harmonised to the whep polity `area_code` through polity_area_crosswalk before they are split to crops, so reporting units that FABIO folds into one bucket are summed rather than dropped (Sudan 276 + South Sudan 277 to 206, Ethiopia PDR 62 to 238). The fold state is the one the rest of the pipeline resolves through, so under the default `options(whep.unfold_rest_of_world = "all")` the Rest-of-World members keep their own code rather than being summed into bucket 999. FAOSTAT rows that are not territories carry no polity and are dropped: 5000 "World", the continent, region, EU-27, OECD and income-group rollups, and the "China" aggregate 351, which overlaps 41/96/128/214. This is a documented exception rather than a coverage gap, since a rollup has no crop shares of its own and would double count its members.

N2O is then estimated with IPCC 2019 Refinement (Vol 4, Ch 11) Tier 1 factors (climate-aggregated): direct `EF1 = 0.010`; indirect via volatilisation `EF4 = 0.010` applied to the volatilised fraction (`FracGASF = 0.11` for synthetic, `FracGASM = 0.21` for manure; crop residues do not volatilise, Eq 11.9); indirect via leaching `FracLEACH = 0.24` times `EF5 = 0.011`. N2O-N is converted to N2O by 44/28 and to CO2e with the chosen GWP100.

Manure deposited by grazing animals (F_PRP) is left out on purpose, not pending: build_livestock_ghg_extens already prices it, direct at EF3PRP and indirect at FracGASM and FracLEACH-(H), and the two extensions are summed in the GHG footprint, so adding it here would count it twice (whep#1524). The livestock chain reports that part apart as `manure_n2o_direct_pasture` and `manure_n2o_indirect_pasture` (see `calculate_manure_emissions()`). Below-ground residue N is a further Tier 1 input not yet included.

Usage

```
build_crop_soil_n2o_extension(
  gwp = c("ar6", "ar5", "ar4"),
  residue_removed_frac = 0.45,
  synthetic_method = NULL,
  data = list(),
  example = FALSE
)
```

Arguments

`gwp`	100-year global warming potential standard for N2O, `"ar6"` (default, 273), `"ar5"` (265) or `"ar4"` (298).
`residue_removed_frac`	Fraction of above-ground crop residue removed from the field (for feed, fuel or construction) and therefore not returned to soil. Defaults to `0.45`, a global mid-range value; country-specific removal (`gleam_fracremove`) is a future refinement.
`synthetic_method`	Synthetic-N crop allocation method, `"coello"` or `"area_share"`. When `NULL` (default), uses `data$synthetic_method %||% "coello"` for backwards compatibility.
`data`	Optional named list of pre-loaded inputs to avoid remote reads: `primary_prod` (`get_primary_production()`, for harvested area), `fertilizer` (the `faostat-fertilizer-nutri`

pin), `manure` (the `faostat-emissions-livestock` pin) and `primary_residues` (`get_primary_residues()`). Each falls back to its reader when absent. `synthetic_method` selects the synthetic-N crop split, `"coello"` (default; Coello 2025 rate-weighted, FAOSTAT-conserving) or `"area_share"`; `coello_rates` overrides the rate table (shaped like coello_synthetic_n), defaulting to `whep::coello_synthetic_n`.

`example` — If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (soil N2O in kilograms CO2e) and `method_soil_n2o`, plus `method_synthetic`, `method_residue_n` (which residue-N source fed the row: `ipcc_2019_table_11_1a`, or `generic_default_0.008` for crops the table does not list) and the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_crop_soil_n2o_extension(example = TRUE)
```

`build_crop_water_consumption`
Build per-crop consumptive water from an LPJmL run, one row per band.

Description

Returns the LPJmL per-crop-functional-type (CFT) consumptive blue and green water and net irrigation requirement **per band**, rather than summed to the cell as `build_water_balance()` reports them. Each value is the band's water expressed as a depth over the whole cell (mm/yr): the per-stand density LPJmL writes, multiplied by the band's stand fraction. That is the same weighting `build_water_balance()` applies before summing, so summing this output over bands per cell-year returns its `blue_consump_mm`, `green_consump_mm` and `cft_nir_mm` exactly.

It is the annual, consumptive counterpart of `build_crop_water_use()`, which keeps the *applied* irrigation (`cft_airrig_month`) per crop and month. The two read different LPJmL outputs and answer different questions (water evaporated by a crop versus water delivered to it), and share the band vocabulary and the `crop_group` column.

The crop dimension is LPJmL's, not WHEP's: a run distinguishes about 16 CFTs (each rainfed and irrigated), and many WHEP items share one CFT – wheat and barley are both `temperate cereals`, and most fruit, vegetable, fibre and stimulant crops fall into `others`. Water is therefore resolved **per CFT band**, never per WHEP item. Mapping a band to items (for example through `cft_mapping`) and deciding how to split it among the items that share it are left to the caller; neither is done here.

Two cautions carried over from the reader. On an LPJmL 6.x run without the green/blue fix (lbm364dl/LPJmL#3) the blue/green split of the consumptive cubes is unusable; the same data check `build_water_balance()` runs warns when rainfed bands carry blue water (class `whep_rainfed_blue_water`), while `blue + green` stays valid. And `cft_nir_mm` is the net irrigation *requirement*, not the gross water applied; which of the two a footprint should charge is a methodological choice.

Usage

```
build_crop_water_consumption(
  resolution = c("grid", "polity"),
  years = NULL,
  bands = NULL,
  data = list(),
  example = FALSE
)
```

Arguments

`resolution`	`"grid"` (per cell and band, depths in mm/yr, default) or `"polity"` (per `area_code`, `year` and band, volumes in m3/yr).
`years`	Optional integer vector of calendar years to read. `NULL` reads every year the files carry; each per-CFT cube is about 3 GB for a full run, so restrict it when reading from disk.
`bands`	Optional character vector of band names (e.g. `"rainfed grassland"`) to keep. `NULL` keeps every band. Matched on the `band_name` the file carries, as in `build_water_balance()`; an unknown name aborts.
`data`	Optional named list of pre-loaded inputs, each falling back to `read_lpjml_hydrology()` (restricted to `years`) when absent: `cft_consump_water_b`, `cft_consump_water_g` and `cft_nir` (each `lon`, `lat`, `year`, `band` and/or `band_name`, `value` mm/yr per stand), `stand_frac` (the same key, `value` the stand fraction of the cell) and, for `resolution = "polity"` only, the required `cell_polity` crosswalk (`lon`, `lat`, `area_code`, `polity_frac`, `cell_area_ha`).
`example`	If `TRUE`, run on a small built-in fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble. For `resolution = "grid"`: `lon`, `lat`, `year`, `band` (when the input carries it), `band_name`, `crop_group` (the soil-carbon crop group of the band, as in `build_crop_water_use()`: `NA` for the `others`, grassland and bioenergy bands), `stand_frac`, and `blue_consump_mm`, `green_consump_mm` and `cft_nir_mm` (mm/yr over the whole cell; `NA` where that cube does not carry the band or was not supplied). Divide by `stand_frac` for a per-hectare-of-crop intensity. Band rows NA in every term (cells LPJmL did not simulate) are dropped; they carry no water, so no band sum changes. For `resolution = "polity"`: `year`, `area_code`, the band columns, `stand_area_ha`, and `blue_consump_m3`, `green_consump_m3` and `cft_nir_m3`, each the depth times `polity_frac * cell_area_ha` (the weight `build_water_balance()` uses for its polity means) summed over the polity's cells, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_crop_water_consumption(example = TRUE)
build_crop_water_consumption(resolution = "polity", example = TRUE)
```

`build_crop_water_use` *Build per-crop monthly applied irrigation from an LPJmL run.*

Description

Applied irrigation per cell, crop and month, from `cft_airrig_month.nc` (first written by the 2026-09-01 v2 run), joined with each crop's stand area fraction from `cftfrac.nc`. This is the disaggregated layer a per-crop water footprint needs; the cell-level water budget stays with `build_water_balance()`, which reports whole-cell terms.

Both unit conventions are returned side by side, because both are needed and confusing them is the characteristic error of per-CFT data: `airrig_stand_mm` is the depth applied per square metre of THE CROP'S OWN STAND (an irrigation intensity), and `airrig_cell_mm` is the same water as a whole-cell depth (`airrig_stand_mm * stand_frac`), which is what sums – over crops it reproduces the crop-less `irrig` cube to 0.999.

Rows are crop stands present in the cell (`stand_frac > 0`), rainfed bands included: LPJmL books paddy water on *rainfed rice* (36% of the stand-weighted applied total at July 2010), so dropping rainfed bands drops real water.

Usage

```
build_crop_water_use(
  resolution = c("grid", "polity"),
  years = NULL,
```

```
  run_dir = NULL,
  data = list(),
  example = FALSE
)
```

Arguments

`resolution`	`"grid"` (default, per cell) or `"polity"` (aggregated to `area_code`, `airrig_stand_mm` area-weighted by stand area, `airrig_cell_mm` by cell area).
`years`	Integer vector of calendar years to keep. Required when the monthly per-CFT cube is read from a run: it is ~7.7e7 long-form rows per year, so `NULL` (every year the run covers) is a request no machine can serve and aborts rather than exhausting memory. `NULL` is fine when `data$airrig_month` is supplied, since the years are then taken from the cube itself.
`run_dir`	Path to the LPJmL run output directory. `NULL` (default) uses `WHEP_LPJML_RUN_DIR`.
`data`	Named list of pre-loaded inputs, each falling back to its reader when absent: `airrig_month` (per cell, band and month, `read_lpjml_hydrology("cft_airrig_month")` output), `stand_frac` (per cell, band and year, `read_lpjml_hydrology("stand_frac", monthly = FALSE)` output) and `country_grid` (the cell-polity support).
`example`	If `TRUE`, return a small fixture instead of reading a run. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `area_code`, `cell_area_frac`, `year`, `month`, `band`, `band_name`, `crop_group`, `stand_frac`, `airrig_stand_mm` and `airrig_cell_mm` at `"grid"` resolution (aggregated over cells at `"polity"`), plus the polity columns below.

`crop_group` is the soil-carbon crop group the band's water belongs to, in the vocabulary of `soc_crop_group()` so the water and carbon ledgers share it: LPJmL's twelve named crops are all herbaceous, so a band is `cropland_rainfed_herbaceous` or `cropland_irrigated_herbaceous` by its regime. The `others` bands pool woody crops with minor herbaceous ones and cannot be resolved to a species, and the grassland and bioenergy bands are not cropland; those carry `NA` rather than a guess.

A border cell shared by several polities appears ONCE PER POLITY, with `cell_area_frac` carrying that polity's share of the cell. The mm columns are densities and do not split; the AREA does. Aggregating over cells therefore weights by `cell_area_frac` (times stand or cell area), and summing rows without it double-counts every border cell.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.

- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

LPJmL run outputs `cft_airrig_month.nc` and `cftfrac.nc`; see Schaphoff, S. et al. (2018). LPJmL4 - a dynamic global vegetation model with managed land - Part 1: Model description. *Geoscientific Model Development*, 11, 1343-1375. doi:10.5194/gmd1113432018.

Examples

```
build_crop_water_use(example = TRUE)
```

`build_cropgrids_land_extension`

Build a per-crop physical land extension from CROPGRIDS.

Description

Convert FAOSTAT harvested area into per-crop *physical* cropland using CROPGRIDS, then return a land extension keyed by `(year, area_code, item_cbs_code)` for the FABIO footprint model.

CROPGRIDS (Tang et al. 2024) reports, per crop and country, both **harvested** area and **crop (physical) area** for 2020. Their ratio is a genuinely per-crop multi-cropping correction — e.g. rice ~0.81 (heavily double-cropped), most other crops ~0.95-1.0 (single-cropped)

— which a single country-level cropping-intensity factor, or a cell-level apportionment by harvested share, cannot reproduce. This function applies that per-(area, item) physical / harvested ratio to WHEP harvested area in each year: $physical = harvested \times (physical_{cg}/harvested_{cg})$.

Note: CROPGRIDS physical area is the land where each crop actually grows; it excludes fallow land (unlike fallow-inclusive cropland-apportionment), so totals are typically a few percent below harvested area.

Coverage: crops absent from CROPGRIDS (notably the FAOSTAT fodder items and a few minor crops) have no physical/harvested ratio and fall back to a ratio of 1 (physical = harvested, no multi-cropping correction). The share of harvested area hitting this fallback is reported via a warning.

Usage

```
build_cropgrids_land_extension(
  harvested = NULL,
  cropgrids = NULL,
  source = c("cropgrids", "cropgrids_fallow"),
  max_ratio = 1.5,
  min_cropgrids_ha = 100
)
```

Arguments

`harvested`	Tibble of harvested area with columns `year`, `area_code`, `item_cbs_code`, `harvested_ha`. If `NULL`, built from `get_primary_production()` (`unit == "ha"`).
`cropgrids`	Tibble of national crop areas with columns `area_code`, `item_cbs_code`, `physical_ha`, `harvested_ha`, where `area_code` is a `polity_area_code` (the basis `harvested` is keyed on). If `NULL`, the remote pin selected by `source` is read via `whep_read_file()` and re-keyed from its source-native raw FAOSTAT area codes to `polity_area_code`, so areas that FABIO merges (Sudan and South Sudan into 206) get one combined CROPGRIDS ratio instead of falling through to the global per-item ratio.
`source`	Which CROPGRIDS pin to read when `cropgrids` is `NULL`: `"cropgrids"` (`cropgrids-land`: physical crop area, excludes fallow) or `"cropgrids_fallow"` (`cropgrids-fallow-land`: physical area with rotational fallow attributed to crops by `attribute_fallow_to_crops()`). Also recorded in `method_land`.
`max_ratio`	Cap on the per-area physical/harvested ratio (default `1.5`). CROPGRIDS occasionally pairs a normal physical area with a near-zero harvested area for minor/aggregate crops, yielding a spurious ratio in the hundreds; physical area cannot realistically exceed harvested by more than the fallow share, so the ratio is clamped here.
`min_cropgrids_ha`	Minimum CROPGRIDS harvested area (ha, default `100`) for a per-area physical/harvested ratio to be trusted. CROPGRIDS leaves rounding stubs of a few hectares for marginal crop-country pairs whose ratio is

unreliable; below this floor the crop falls through to the global per-item ratio instead.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (physical land area in hectares), and `method_land`.

Examples

```
harvested <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~harvested_ha,
  2000L, 33L, 2511L, 1000,
  2000L, 33L, 2807L, 500
)
cropgrids <- tibble::tribble(
  ~area_code, ~item_cbs_code, ~physical_ha, ~harvested_ha,
  33L, 2511L, 990, 1000,
  33L, 2807L, 400, 500
)
build_cropgrids_land_extension(harvested, cropgrids)
```

`build_detailed_trade` *Build detailed bilateral trade matrix*

Description

Construct the detailed bilateral trade matrix (DTM) from the FAOSTAT Detailed Trade Matrix pin. Reports trade flows between pairs of countries with their trade shares, aggregated to polity level and mapped to CBS item codes.

Optionally extends the time series by joining with commodity balance sheet years and gap-filling country shares via linear interpolation.

Usage

```
build_detailed_trade(
  raw_trade = NULL,
  cbs = NULL,
  min_share = 1e-04,
  extend_time = FALSE,
  method_unbacked_quantity = c("drop", "keep", "abort"),
  method_head_units = c("convert", "drop", "abort"),
  method_time_coverage = c("cbs_cells", "cbs_years"),
  example = FALSE
)
```

Arguments

`raw_trade` A data.table or tibble of raw FAOSTAT bilateral trade data. If `NULL` (default), the data is read from the `"faostat-trade-bilateral"` pin.

`cbs` A tibble of commodity balance sheets in wide format, as returned by `build_commodity_balances()` or `get_wide_cbs()`. Required when `extend_time = TRUE`.

`min_share` Numeric. Partners with a country share below this threshold are dropped when extending time. Default `0.0001`.

`extend_time` Logical. If `TRUE`, extend the time series using CBS years and linear interpolation of country shares. Default `FALSE`.

`method_unbacked_quantity`
How to treat a reported `tonnes` quantity for a FAOSTAT trade item whose country-level mass FAOSTAT itself does not publish. See the *Quantities FAOSTAT does not back with a mass* section. One of:

- `"drop"` (default): discard those rows, warning with the tonnage removed. They are not masses, and no conversion to mass is derivable.
- `"keep"`: carry them verbatim, with the same warning. The historical behaviour, and unsafe for anything that treats the column as mass.
- `"abort"`: fail, so a refreshed pin cannot reintroduce them unnoticed.

`method_head_units`
How to treat the FAOSTAT rows denominated in `1000 Head`. See the *Live animals are reported in two head units* section. One of:

- `"convert"` (default): rescale them by 1,000 onto `heads`, the denomination the rest of the live-animal record already uses.
- `"drop"`: discard them, warning with the head count removed. The historical behaviour, which leaves live poultry, rabbit and rodent trade out of the record entirely.
- `"abort"`: fail, so a refreshed pin cannot reintroduce an unhandled unit unnoticed.

Any other unit label is dropped under every method, with a `"whep_unhandled_trade_unit"` warning naming it, and aborts under `"abort"`. Monetary units are removed silently, on purpose.

`method_time_coverage`
Which extended shares `extend_time` keeps. See the *Time extension and CBS coverage* section. One of:

- `"cbs_cells"` (default): only in the year, area, item and element cells where `cbs` reports a non-zero flow.
- `"cbs_years"`: in every CBS year, whether or not CBS reports the cell. The historical behaviour.

Ignored when `extend_time = FALSE`.

`example` Logical. If `TRUE`, return a small example tibble without downloading remote data. Default `FALSE`.

Value

A tibble with columns:

- `year`: Integer year.
- `area_code`: Numeric polity code of the reporter country, the `polity_area_code` its FAOSTAT area resolves to in that year.
- `polity_area_code`, `reporting_polity_code`, `reporting_polity_name`, `reporting_polity_has_geometry`: the reporter's polity columns, see the *Polity columns* section.
- `area_code_partner`: Numeric polity code of the partner country.
- `partner_polity_code`, `partner_polity_name`, `partner_polity_has_geometry`, `partner_polity_area_co`: the same polity columns for the partner.
- `element`: Either `"import"` or `"export"`.
- `item_cbs`: CBS item name.
- `item_cbs_code`: Numeric CBS item code.
- `unit`: Measurement unit (`"tonnes"` or `"heads"`).
- `value`: Trade quantity.
- `country_share`: Share of total trade for this partner.
- `method_unbacked_quantity`: the treatment chosen for quantities FAOSTAT does not back with a mass, recorded so a downstream consumer can tell which variant it is holding.
- `method_head_units`: the treatment chosen for the `1000 Head` rows, recorded for the same reason.
- `method_time_coverage`: the coverage rule of the time extension, or `NA` when `extend_time = FALSE`.

Time extension and CBS coverage

With `extend_time = TRUE` every `(area, item, partner, element, unit)` group observed in any trade year is carried across the union of trade and CBS years, and `fill_linear()` interpolates inside a group's observed span and holds the first and last observed share constant outside it. That fills two kinds of row: years outside the trade record, and trade years in which the reporter reported nothing at all for that item and element (the group's share there is 0/0). `method_time_coverage` then decides which of those filled shares are kept:

- `"cbs_cells"` (default) keeps an extended share only in a `(year, area_code, item_cbs_code, element)` cell where `cbs` reports a non-missing, non-zero flow for that element. A share partitions a CBS total among partners; where there is no total there is nothing to partition, so no share is invented there. Observed trade rows are always kept, whether or not CBS reports the cell.
- `"cbs_years"` keeps every extended share in every CBS year. The historical behaviour: the CBS *year axis* alone drives the extension, so shares are also emitted for cells CBS never reports.

The two methods differ only in which rows survive: a share kept by both is identical, because it comes from the same interpolation of the group's own anchors, and every kept cell keeps all its partners, so its shares still sum to one. Because filled rows inside the trade record are scoped too, a `cbs` that omits some trade years drops the filled rows of those years under `"cbs_cells"`; pass a CBS spanning the trade record.

Measured on the `"faostat-trade-bilateral"` pin against a 1850-2023 CBS, half of the `(year, area, item, element)` cells the uniform extension emits are cells CBS never reports (3.42 of 6.85 million; whep#232). Against single-year CBS builds, `"cbs_cells"` keeps 144,812 of the 286,762 rows `"cbs_years"` emits for 1975 (50.5%) and 342,617 of 384,954 for 2023 (89.0%), extending from trade years 1986-1987 and 2020-2021; and 231,530 of 249,833 for 2000 (92.7%) and 279,210 of 293,815 for 2010 (95.0%), with trade years 1999-2001 and 2009-2011.

Neither method bounds the **year axis**. The pin covers 1986-2021, while `build_commodity_balances()` defaults to 1850-2023, so 138 of the 174 extended years (79%) lie outside the trade record entirely and carry the 1986 (or 2021) partner mix held constant. CBS is itself extended back to 1850, so scoping to its cells does not shorten that back-cast; pass a year-scoped `cbs` to limit it.

Quantities FAOSTAT does not back with a mass

The Detailed Trade Matrix carries a `tonnes` column for every item, but FAOSTAT does not stand behind all of it. For trade item 1293 (*Crude organic material n.e.c.*, mapped here to CBS item 5001 `"Other"`) the aggregate *Trade: Crops and livestock products* domain publishes a **value** but no country-level **mass**: on the `faostat-trade-totals` pin `20260325T120525Z-7b85f` that item has 28,213 value rows against 6,000 quantity rows, only 1,830 of them non-zero, and every quantity above 10 Mt belongs to a FAOSTAT *regional aggregate* (area code >= 51000) flagged `E`. Spot-checked for Colombia, Kenya, South Africa and Japan over 2002-2010, the country-level quantity is `0` or absent in every year.

The detailed matrix nonetheless reports masses for it that cannot be masses (whep#1023). On the `faostat-trade-bilateral` pin `20260407T095142Z-b3f81`, Colombia's 2004 export of item 1293 to the United States is 2,579,549,000 tonnes against an export value of USD 584.4 million, i.e. **USD 0.227 per tonne**; South Africa's 2008-2013 flows with Uganda run at USD 0.011-0.015 per tonne. The mirrored report of the same flow disagrees by factors of 356 to 838,000 - Kenya says it exported 534,611,300 tonnes to the Netherlands in 2009, the Netherlands says it imported 72,076 tonnes (7,417x) - and it is always the large side that is not a mass. What the large side actually counts (stems, pieces, bunches) is **not recorded anywhere in the data and is therefore unverified**, and the implied units per tonne are not constant across cells, so no conversion factor can be derived: the figures can be dropped or carried, not corrected.

`method_unbacked_quantity` selects the treatment. Screened tonnage on the full pin, restricted to items that map to a CBS item: 15.31 Gt of 83.16 Gt (**18.4%**), 99.78% of it item 1293. By year the screen takes 12.7-70.0% of 2003-2013, 5.6-16.5% of 1986-1988, 10.1-13.0% of 2000-2002, at most 3.3% of 1989-1999 and at most 0.04% of 2014-2021 - item 1293 carries no bilateral quantity at all from 2014, which is why the recent record looks clean.

Those tonnes do not reach this function's output today whichever method is chosen, for an unrelated reason: item 1293's CBS name `"Other"` has no row in `whep::items_full`, so the item-code bridge leaves it without an `item_cbs_code` and it is dropped a step later. That

was silent until now and is warned about separately; four CBS names and 14 of the 700 trade item codes in `whep::cbs_trade_codes` are affected. The screen is therefore explicit where the mapping gap was accidental, and it keeps working if the gap is ever filled.

Live animals are reported in two head units

FAOSTAT denominates live-animal trade in `Head` for the large species and in `1000 Head` for the small ones, and the unit filter here used to cover only the first label. On the `faostat-trade-bilateral` pin `20260407T095142Z-b3f81` that silently removed **89,073 rows carrying 76,141,882 thousand head** over 1986-2021 - live chicken, turkey, duck, goose, other-bird, rabbit and rodent trade - against the 11,708,244,416 head the `Head` rows carry. It also removed 5,011 `No` rows (whep#1092).
`1000 Head` is a decimal prefix the source states, not a coefficient: `.aggregate_fao_trade_to_cbs()` already rescales its sibling label `1000 An` the same way (whep#865). The reported trade *value* of the same flows corroborates it - USD 1,408 per `1000 Head` of chickens, 2,392 for ducks, 2,483 for geese, 3,469 for turkeys and 4,945 for rabbits, i.e. USD 1.41-4.95 a bird, against USD 654 per head of cattle, 105 per pig and 74 per sheep on the `Head` rows. Read as single head, a live broiler chick would cost USD 1,408. **No head-to-mass factor is involved**: head counts stay head counts, on their own `unit` key.

The `No` rows are a different case and are **not** converted. They are FAOSTAT trade item 1181, which `whep::cbs_trade_codes` names *Beehives* while the pin names it *Bees*, so what the number counts - insects, packages or colonies - is not established by the source. They are dropped, with the warning every unrecognised label now raises.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_detailed_trade(example = TRUE)
```

---

build_energy_co2_extension

Build the livestock energy-use CO2 footprint extension (meat only).

Description

Aggregate GLEAM 3.0 on-farm (direct) and feed-production (embedded) energy use into a footprint extension keyed by `(year, area_code, item_cbs_code)`, expressed in kilograms of carbon-dioxide equivalent (CO2e). This is the energy slice of the livestock greenhouse-gas basket and is designed to be summed with `build_livestock_ghg_extension()` (enteric and manure CH4/N2O), which keys on the same live-animal sectors.

The GLEAM energy emission factors are expressed per kilogram of **live weight** (see gleam_energy_use_ef), which is well defined for meat but not for milk or eggs, so the extension covers **meat only**: bovine (`item_cbs_code` 961 non-dairy cattle and 946 buffalo), sheep (976) and goat (1016), pig (1049 and 1051) and broiler-chicken (1053) meat. Milk and eggs keep their CH4/N2O but get no energy CO2.

For each meat group the live weight produced is recovered from FAOSTAT carcass production divided by a GLEAM dressing fraction (gleam_dressing_percentages), multiplied by a per-country energy intensity (embedded + direct), and then attributed to the contributing live-animal sectors in proportion to their slaughtered head counts. Because GLEAM reports its factors by production system and climate zone but the package has no country-level system or climate shares, the intensities are collapsed to one value per country by an unweighted mean across systems and climate zones; this choice is recorded in `method_energy`. A meat group with carcass output but no slaughtered-head counts keeps its energy CO2e, split equally across the group's live-animal sectors, and triggers a warning.

`gleam_geographic_hierarchy` is the country universe of the whole extension, so a reporting area absent from it gets no energy intensity and its meat production leaves the extension. That affects both the aggregate reporting buckets (`polity_area_code` 999 "Rest of World" and the continental residuals 901-906) and the dissolved entities GLEAM's present-day country table cannot carry (USSR, Czechoslovakia, Yugoslavia, Belgium-Luxembourg, Serbia and Montenegro). The size of that loss is now **reported** on every build rather than

left to be inferred, and three opt-in treatments recover it instead of losing it: `unclassified = "polity_region"` groups the **live** reporting areas GLEAM omits (today Nauru and Tuvalu) from the polity crosswalk, `unclassified = "historical_region"` additionally groups the **dissolved** entities from the membership they themselves held while they existed, and `unclassified = "global_mean"` prices **every** unclassifiable area at the world-mean GLEAM intensity. The default keeps the historical behaviour; see whep#415, whep#492 and whep#553.

Usage

```
build_energy_co2_extension(
  method = c("gleam"),
  data = list(),
  unclassified = c("drop", "polity_region", "historical_region", "global_mean"),
  example = FALSE
)
```

Arguments

`method`	Estimation method. Only `"gleam"` (default), the GLEAM 3.0 per-live-weight factors, is currently available.
`data`	Optional named list of pre-loaded inputs to avoid remote reads: `primary_prod` (the `get_primary_production()` output). It falls back to its reader when absent.
`unclassified`	How to treat reporting areas `gleam_geographic_hierarchy` has no row for, and which therefore get no country energy intensity. `"drop"` (default) keeps the historical behaviour: their meat production leaves the extension, and a warning says how much. `"polity_region"` gives the **live, self-reporting** ones among them a grouping derived from their polity in `polity_area_crosswalk`, running GLEAM's own scheme rules on that continent, and marks those rows `"GLEAM_3.0_energy_meat_polity_region"`; the aggregate buckets and dissolved entities still drop. `"historical_region"` does everything `"polity_region"` does and also groups the **dissolved** entities (USSR, Czechoslovakia, the Yugoslav SFR, Belgium-Luxembourg, Serbia and Montenegro, the Netherlands Antilles) from the OECD and EU membership they held while they existed, marking those rows `"GLEAM_3.0_energy_meat_his` `"global_mean"` instead prices every unclassifiable area at the unweighted world mean of the published GLEAM factors, marking those rows `"GLEAM_3.0_energy_meat_glo`
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (energy-use emissions in kilograms CO2e) and `method_energy` (`"GLEAM_3.0_energy_meat"`, `"GLEAM_3.0_energy_meat_polity_region"` for rows grouped from the polity crosswalk, `"GLEAM_3.0_energy_meat_historical_region"` for dissolved entities grouped from their own era's memberships, or `"GLEAM_3.0_energy_meat_global_mean"` for rows priced at the world mean), plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_energy_co2_extension(example = TRUE)
```

`build_fao_arable_fallow_extension` *Build a per-crop physical land extension with FAO fallow-inclusive arable land.*

Description

Turn per-crop harvested-derived physical area into a fallow-inclusive physical land extension whose arable-crop total reconciles to FAO's physical **Arable land** and whose perennial-crop total reconciles to FAO's physical **Permanent crops** (`get_arable_permanent_land()`), per `(area_code, year)`.

This is the FAO-land-base analogue of `build_cropgrids_land_extension()(source = "cropgrids_fallow")`. The existing method takes the fallow *magnitude* from FAOSTAT "Temporary fallow" (item 6640, a sparse and, for many rain-fed economies, absent series) applied to a single CROPGRIDS 2020 snapshot. Here the fallow magnitude is the physical arable land that carried no harvest in that specific year, `FAO Arable land - sum(cropped arable physical)`, so a drought year's resting cropland is charged to the crops whose rotation it supports and the arable-crop footprint totals match FAO's land survey in every year (see the Tunisia/Portugal motivation in `get_arable_permanent_land()`).

Reconciliation, per `(area_code, year)`:

- **Arable crops** (`items_prod_full$Herb_Woody != "Woody"`): rotational fallow `max(0, arable_ha - S)` (with `S` the cropped arable physical total) is distributed with `attribute_fallow_to_crops()` using `fallow_weights`, so the arable total reaches `arable_ha`. Where the cropped physical already exceeds `arable_ha` (heavy multi-cropping, or inflated fodder harvested area) there is no fallow to add and the arable crops are scaled down to `arable_ha` instead, the physical-container correction. Either way the arable total equals FAO `arable_ha` by construction.
- **Perennial crops** (`Herb_Woody == "Woody"`) receive no fallow and are scaled so their total equals FAO `permanent_ha`, preserving the within-group physical pattern. A positive target without a corresponding arable crop row or positive perennial base area cannot be reconciled without inventing a crop allocation; `unsupported_target` decides what happens to it (see the unsupported-target section).

This is the crop-side default of the land-balance footprint (`build_land_balance_footprint()`).

Usage

```
build_fao_arable_fallow_extension(
  harvested = NULL,
  arable_permanent = NULL,
  base_extension = NULL,
  fallow_weights = NULL,
  temporary_grassland = NULL,
  items_prod_full = whep::items_prod_full,
  temp_grassland_basis = c("modelled", "modelled_then_fao", "fao_official", "fao_all",
    "none"),
  fodder_gap = c("as_reported", "carry_forward", "drop"),
  unsupported_target = c("unallocated", "zero", "drop", "abort")
)
```

Arguments

`harvested`	Tibble of harvested area with columns `year`, `area_code`, `item_cbs_code`, `harvested_ha`. If `NULL`, built from `get_primary_production()` (`unit ==`

"ha"); passing a cached harvested table avoids that rebuild.

arable_permanent
: Tibble of FAO physical land base with columns area_code, year, arable_ha, permanent_ha. If NULL, get_arable_permanent_land() is called for the years present in base_extension.

base_extension
: Tibble of cropped (fallow-excluding) per-crop physical area with columns year, area_code, item_cbs_code, impact_u. If NULL, built with build_cropgrids_land_exten = "cropgrids") from harvested.

fallow_weights
: Tibble of area_code, item_cbs_code, weight giving the within-country fallow allocation weight, e.g. from gridded_fallow_weights() (the recommended agro-climatic, rainfed-gated weight). If NULL, fallow is distributed in proportion to each arable crop's cropped physical area (perennials always excluded). The cropped-area fallback is used independently for an area when it has no usable supplied weights, a non-finite or negative supplied weight, or a non-positive total.

temporary_grassland
: Tibble of grassland occupation in the build_grassland_land_extension() schema (area_code, year, item_cbs_code, impact_u); its CBS 3002 rows are the temporary grassland netted out of the arable target so ordinary crops plus CBS 3002 reconcile to FAO Arable land (see the temporary-grassland section). If NULL (default) it is built with build_grassland_land_extension()(grass = "occupation") so netting still applies (correct but slow); supply the table to skip that rebuild, or pass one with no CBS 3002 rows to opt out.

items_prod_full
: Crosswalk used to classify item_cbs_code as arable or perennial via Herb_Woody. Defaults to items_prod_full.

temp_grassland_basis
: Which measurement of temporary grassland is netted out of the arable target. "modelled" (default) is the published behaviour: WHEP's own CBS 3002, which exists for 26 EU polities and stops in 2019. "modelled_then_fao" keeps that and fills every other country-year from official FAO 6633. "fao_official" uses official FAO 6633 everywhere, "fao_all" uses FAO 6633 on every observation-status flag (~81% of it is FAO-imputed), and "none" nets nothing, the behaviour before whep#349. See the netting-basis section.

fodder_gap
: How the FAOSTAT fodder items (CBS 2000-2003) are treated where their sources have run out. "as_reported" (default) is the published behaviour: no fodder from 2020, so ordinary crops absorb its land. "carry_forward" extends each fodder series' last observed physical area over the rest of that country's panel. "drop" removes fodder from the whole panel. See the fodder-gap section.

unsupported_target
: What happens to a country-year whose positive FAO land target has no crop row of the matching kind to carry it. "unallocated" (default) keeps the hectares as an item_cbs_code NA row, "zero" lets the unreconcilable

target contribute no land, `"drop"` removes the whole affected country-year, and `"abort"` refuses to continue (the behaviour before whep#1026). See the unsupported-target section.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (fallow-inclusive physical land in hectares), `method_land` (`"fao_arable_fallow"`), `temp_grassland_netted_ha` (hectares netted out of that country-year's arable target, `0` where the netting term is structurally absent), `temp_grassland_source` (the source that supplied that netted term: `"modelled_cbs_3002"`, `"fao_6633_official"`, `"fao_6633_all"`, `"absent"` where the basis has no value for that country-year, or `"not_netted"` under `"none"`), `method_temp_grassland` (the `temp_grassland_basis` in force), `method_fodder` (the `fodder_gap` in force) and `method_unsupported_target` (the `unsupported_target` in force), and `fodder_coverage`, which says per country-year whether the fodder input was there before any `fodder_gap` treatment: `"reported"` (the base carries fodder area that year), `"lapsed"` (it carries none, but did in an earlier year of that country's panel – the 2020 break of whep#938, whose land the other arable crops absorb under `"as_reported"`, or which `"carry_forward"` fills) or `"not_reported"` (none that year or before). Under `unsupported_target = "unallocated"` a row with `item_cbs_code` `NA` carries the FAO land no crop can be named for.

Temporary grassland (no double-count)

FAO's **Arable land** total includes *temporary meadows and pastures* — temporary grassland is part of cropland, not grassland. That land is also reported separately as CBS 3002 (`Temporary grassland`) by `build_grassland_land_extension()`, so summing both extensions naively would count it twice. Pass that grassland occupation as `temporary_grassland` and its CBS 3002 is netted out of the arable target before reconciling ordinary crops, enforcing the invariant per `(area_code, year)` `ordinary crop occupation (incl. fallow) + CBS 3002 = FAO Ara`. The land-balance footprint (`build_land_balance_footprint()`) does exactly this, passing the grassland occupation it has already built. When `temporary_grassland` is `NULL` (default) the grassland occupation extension is built internally so netting still happens — correct but slow, since that build reruns much of the pipeline; supply the table to avoid the rebuild. Where modelled CBS 3002 exceeds FAO Arable land (survey vs. fodder-reconstruction mismatch) the arable target is clamped at 0 and a warning is emitted.

Netting basis, and the 2019/2020 seam

Modelled CBS 3002 comes from EU AgriDB alone — FAOSTAT production item 996 is in neither production pin, and the EU AgriDB fodder source runs 1961-2019 for all 28 of its region keys. So over 2001-2023 the netting term exists for **26 EU polities and the years 2001-2019 only**, and is identically zero everywhere else. For those 26 polities the arable target therefore steps **from 96.4 Mha in 2019 to 103 Mha in 2020** while their own FAO arable land *falls*, so ordinary arable crops there gain land with no land-use change behind it. Measured over 2001-2023 on the real inputs, the land the netting removes is 8.2-9.8 Mha a year to 2019 and **exactly 0** from 2020. `temp_grassland_basis` exposes the alternatives measured in whep#937 and whep#354; `"modelled"` remains the default so this argument changes no published number until a basis is chosen deliberately. `temp_grassland_netted_ha` in the output, and `check_arable_composition()`, make the switch-off visible either way.

`temp_grassland_source` says, per country-year, which source supplied the netted term: `"modelled_cbs_3002"`, `"fao_6633_official"` or `"fao_6633_all"`, or `"absent"` where the chosen basis has no value there, so a country-year with no measurement never reads as one with no temporary meadows. On the real 2001-2023 inputs under `"modelled"`, 494 of the 4,549 country-years in the output are `"modelled_cbs_3002"` (26 polities, 2001-2019) and the other 4,055 are `"absent"`, including all 787 from 2020. Under the `"fao_*"` bases the column also separates a reported value that nets 0 ha from a missing report: 103 country-years under `"fao_official"` and 993 under `"fao_all"` have a value yet net nothing, and before this column they read exactly like the country-years with no value at all.

FAO's own item 6633 "Temporary meadows and pastures" measures the same concept, runs 2001-2023, and is what the `"fao_*"` bases read. It is not a drop-in replacement: only ~19% of it is an official value, Greece and Poland are imputed zeros throughout while WHEP models 2.10 and 4.78 Mha there, and its scope is country-dependent — for Ireland, Sweden, the United Kingdom, the Netherlands, Belgium, Luxembourg and Czechia it equals WHEP's CBS 3002 to the digit, while for Germany, Italy, Romania, Spain, Denmark, Austria and Bulgaria it is 3-40 times larger and lands near the whole green-fodder group. `validation/temp_grassland_6633.R` records that comparison.

Fodder gap

FAOSTAT's fodder tonnage (`faostat-production-old`, production only, no harvested area at all) runs to 2013, whose rows `.combine_fodder()` drops, so it effectively ends in 2012; EU AgriDB, the only other source, ends in 2019. Fodder harvested area is reconstructed from those two, so a build reaching 2020 has **no fodder at all** from that year: measured over 2001-2023 on the real inputs, fodder is 9.2% of the reconciled arable land extension in 2001 and 7.6% in 2019, then **0%** from 2020, with ordinary arable crops absorbing the difference (whep#938). A second, earlier composition change sits inside the covered window: from 2013 the FAOSTAT-derived fodder area disappears and the 2012 area is carried forward, 2.2 Mha in 2012 and 75.0 Mha from 2013, held flat to 2019. Production rows say so in `source` (`"DM_yield_estimate_carried_forward"`, whep#1027), but this extension does not see `source`, and fodder is present on both sides of 2013, so `check_arable_composition()` does not flag that year. `fodder_gap` exposes the treatments; `"as_reported"` remains the default. Whatever the treatment, `fodder_coverage` in the output marks every country-year after the fodder input stops as `"lapsed"`, and `check_series_jumps()` with `dropouts = TRUE` flags each fodder series at the year it stops.

Unsupported land targets

FAO reports positive land for some country-years in which the crop panel has nothing of the matching kind to carry it, so the reconciliation has no crop to attribute the land to. Over the full 1850-2023 span on the real default inputs there are 923 such country-years (whep#1026):

- **arable**: 33 country-years, all Marshall Islands (`area_code` 127), 1991-2023, 500 ha of FAO Arable land each (16,500 ha in total) with no arable crop row at all – its only crop area is perennial.
- **permanent crops**: 890 country-years in 8 areas whose perennial base area is zero: Poland (173) 1850-1964, 27.72 Mha summed over years (up to 287,779 ha in a year); Nepal (149) 1850-1973, 2.40 Mha; Burkina Faso (233) 1850-1976, 1.35 Mha; Denmark

(54) 1850-1984, 1.10 Mha; Saint Kitts and Nevis (188) 1850-1984, 0.67 Mha; Sweden (210) 1850-1965, 0.39 Mha; Chad (39) 1850-1984, 0.35 Mha; Mongolia (141) 1983-1985, 3,000 ha. 33.98 Mha summed over all 890 country-years.

Before whep#1026 this was an error, which made the function's own documented span unreachable and blocked `build_land_balance_footprint()` entirely. `unsupported_target` now selects the treatment, and none of them invents a crop allocation:

- `"unallocated"` (default) keeps the hectares in the ledger as one row per affected country-year with `item_cbs_code` `NA` – real reported land that no crop can be named for. Nothing is lost and the gap is visible.
- `"zero"` treats the FAO total as the error: the unreconcilable target contributes no land, the rest of the country-year reconciles unchanged.
- `"drop"` removes the whole affected `(area_code, year)`, including the crop rows that *were* supported. For Poland that deletes 115 years of Polish arable land, so it is the most destructive option.
- `"abort"` is the pre-whep#1026 behaviour and refuses to continue.

Only the `NA`-item rows separate `"unallocated"` from `"zero"`; every other country-year is identical under all three continuing treatments. Restricted to 2001-2023 the whole difference from `"abort"`'s (unreachable) output is the 23 Marshall Islands rows, 11,500 ha.

Examples

```
harvested <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~harvested_ha,
  2020L, 1L, 2511L, 300, # wheat (arable)
  2020L, 1L, 2560L, 100 # coconuts (perennial)
)
base_extension <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~impact_u,
  2020L, 1L, 2511L, 300,
  2020L, 1L, 2560L, 100
)
arable_permanent <- tibble::tribble(
  ~area_code, ~year, ~arable_ha, ~permanent_ha,
  1L, 2020L, 500, 100
)
items <- tibble::tribble(
  ~item_cbs_code, ~Herb_Woody,
  2511L, "Herbaceous",
  2560L, "Woody"
)
temporary_grassland <- tibble::tribble(
  ~area_code, ~year, ~item_cbs_code, ~impact_u,
  1L, 2020L, 3002L, 100 # temporary grassland netted out of arable
)
build_fao_arable_fallow_extension(
  harvested, arable_permanent, base_extension,
  temporary_grassland = temporary_grassland,
  items_prod_full = items
)
```

`build_feed_demand` *Build livestock feed demand.*

Description

Estimate the dry-matter feed demand of each livestock category: the first stage of `get_feed_intake()`, exposed on its own. Demand is national, per `(year, area_code, livestock_category)`, and is computed before any matching against feed supply, so it can be audited or reused (for example in land or nitrogen footprints) independently of the allocation.

Usage

```
build_feed_demand(
  demand_tier = c("ipcc", "fcr"),
  by = c("category", "feed_type"),
  region_fallback = c("member_mix", "none"),
  example = FALSE
)
```

Arguments

`demand_tier`	Demand-estimation tier. `"ipcc"` (default) uses the IPCC Tier-2 energy model for the ruminant species, Bouwman feed-conversion ratios for pigs and poultry, and Krausmann per-head intake for draft and other species. `"fcr"` uses the Bouwman / Krausmann magnitude for every species. The method actually used for each row is recorded in `method_demand`.
`by`	Output grain. `"category"` (default) returns the per-livestock category demand. `"feed_type"` splits it across feed types and returns the `feed_demand` table that `redistribute_feed()` consumes, so the two compose: `build_feed_demand(by = "feed_type") |> redistribute_feed(feed_avail)`.
`region_fallback`	How to give a Bouwman feed region to a reporting bucket the crosswalk leaves without one. `"member_mix"` (default) splits Rest of World (`area_code` 999) across the Bouwman regions of the 62 reporting areas folded into it, weighted by the livestock those members carry. `"none"` leaves the bucket unmapped, which drops its whole feed demand from the feed-type mix; that was the behaviour before whep#467 and is kept selectable for comparison. Neither value affects any bucket the crosswalk already resolves.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

With `by = "category"`, a tibble with one row per `(year, area_code, livestock_category)`:

- `year`: The year of the demand.

- `area_code`: The country code. For code details see e.g. `add_area_name()`.
- `livestock_category`: The feed-demand grouping of livestock (e.g. `Cattle_milk`, `Cattle_meat`, `Pigs`, `Poultry`).
- `demand_dm_t`: Dry-matter feed demand in tonnes.
- `method_demand`: The demand method(s) used, e.g. `ipcc_tier2_energy`, `bouwman_fcr` or `krausmann_per_head` (a `+`-joined set for a mixed category whose animals used different methods).

plus the polity columns below.

With `by = "feed_type"`, the demand split across feed types as the `redistribute_feed()` `feed_demand` contract: `year`, `territory`, `sub_territory`, `livestock_category`, `item_cbs_code`, `feed_group`, `feed_quality`, `demand_dm_t`, `fixed_demand`. That grain is keyed by `territory`, not `area_code`, so it carries no polity columns.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_feed_demand(example = TRUE)
build_feed_demand(example = TRUE, by = "feed_type")
```

`build_feed_intake_local` *Build local (per-cell) feed intake, chunked by year.*

Description

Runs the `redistribute_feed` local path (0.5-degree cell grain) one year at a time, so the per-cell allocation stays within memory and the full multi-year run is restartable. By default it sources pinned LPJmL-derived grass availability and pinned gridded livestock inputs. Pass `run_dir`, `grass_availability`, `grass_availability_path`, or `input_dir` to use custom local inputs instead.

Usage

```
build_feed_intake_local(
  years = NULL,
  out_dir = NULL,
  demand_tier = c("ipcc", "fcr"),
  feed_mode = c("historical", "scenario"),
  overwrite = FALSE,
  example = FALSE,
  run_dir = NULL,
  input_dir = NULL,
  grass_availability = NULL,
  grass_availability_path = NULL,
  feed_eligibility = c("feed_table", "none"),
  feed_loss = c("wirsenius", "flat", "none")
)
```

Arguments

`years`	Integer vector of years to build. Default `NULL` builds every year present in the production data.
`out_dir`	Directory to write per-year `feed_intake_local_<year>` parquet files to. If `NULL`, the bound result is returned in memory (only practical for a few years).
`demand_tier`	Demand-estimation tier, `"ipcc"` (default) or `"fcr"`.
`feed_mode`	Whether to distribute surplus feed availability. `"historical"` (default) suppresses the surplus-distribution pass: the CBS feed element is treated as realised consumption, so leftover availability is not dumped onto variable-demand livestock (which would inflate non-grass intake). `"scenario"` distributes the surplus.

`overwrite` Re-run years whose output file already exists. Default `FALSE` skips them so the batch is restartable.

`example` If `TRUE`, return a small example output without sourcing the remote and gridded data. Default is `FALSE`.

`run_dir` Optional path to a finished local LPJmL output directory holding `pft_npp.nc` and `cftfrac.nc`. If `NULL`, pinned grass availability is used unless `grass_availability` or `grass_availability_path` is supplied.

`input_dir` Optional directory holding locally prepared spatialization inputs. If `NULL`, pinned gridded livestock/spatial inputs are used.

`grass_availability`
Optional already-derived grass availability tibble/data frame passed to `build_grass_availability_lpjml()`.

`grass_availability_path`
Optional path to an already-derived grass availability artifact passed to `build_grass_availability_lpjml()`.

`feed_eligibility`
Which feeds each livestock category may receive. `"feed_table"` (default) follows `feed_taxonomy`: an item with a grazer feed type but no granivore feed type (straw and green fodder) is fibrous roughage that pigs and poultry do not eat, so it is never allocated to them, at any allocation level. Granivore demand that the feeds they may eat cannot cover stays unmet rather than being filled with roughage. `"none"` lets any category receive any item (the behaviour before whep#1218), for sensitivity analysis.

`feed_loss` Share of the Commodity Balance Sheet `feed` element lost between distribution and intake. FAOSTAT books storage and transport losses in a separate element, so the `feed` element is already net of them. `"wirsenius"` (default) applies the per-class losses of Wirsenius (2000, sections 3.1.2 and 3.1.5): none on Food Balance Sheet products and processing by-products, 10% on crop residues (`Straw`, `Other crop residues`), 20% on harvested grass, legume and cereal fodder (hay-making and ensiling, harvest to intake) and 5% on fodder vegetables and roots; temporary grassland, booked as grazed grass, carries none. `"flat"` applies 10% to every item, the unsourced factor used before whep#1420. `"none"` takes the `feed` element as intake. Grass from the grassland sink and scavenging are not Commodity Balance Sheet items and carry no loss under any method.

Value

When `out_dir` is `NULL`, a tibble in the `get_feed_intake()` contract plus a `sub_territory` (0.5-degree cell) column. Otherwise, invisibly, the written file paths.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.

- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_feed_intake_local(example = TRUE)
```

`build_food_supply` *Build per-capita food supply for the nourishment axis.*

Description

Assembles per-capita protein and dietary-energy supply, the state variable for the SJOS-N nourishment ("just") axis. Protein is the SJOS-N nourishment axis; dietary energy is a secondary cross-check. The default `"whep_native"` method multiplies the WHEP commodity-balance food element (tonnes fresh matter, per `year`, `area_code`, `item_cbs_code`) by the per-item nutrition coefficients in `whep::biomass_coefs` and divides by national population. Protein per kilogram fresh matter is nitrogen times 6.25 (nitrogen-to-protein factor), on the basis selected by `protein_basis`. The nitrogen density is `N_kgN_kgFM` where available, otherwise `Product_kgN_kgDM * Product_kgDM_kgFM`. `Edible_N_kgFM` is not read: it is empty in every coefficient row, upstream as well as in the packaged data, so the edible basis is derived from `Edible_portion` instead of stored redundantly. `N_kgN_kgFM`

must be a **food-composition** density for this to mean anything, because `food_t` for a processed FBS item is the primary equivalent of the products actually eaten: a whole-grain nitrogen density on that mass counts milling offal as food. Five cereal rows carried the agronomic value instead, and others a bare literal; biomass_coefs names both sets and what became of each (#1096: `Olive`, `Rye` and `Honey` now follow their food-composition rows, `Oats`, `Maize` and `Rice` do not). `Wheat` was the largest, at 1.27x FAOSTAT FBS world protein until #796, and now ships on the **flour basis**, 93 g of protein per kg, sourced and measured in biomass_coefs. Energy per kilogram fresh matter follows `GE_product_edible_portion_MJ_kgFM`, then `GE_product_MJ_kgFM` (MJ per kg fresh matter), converted to kilocalories via `MJ / 0.004184`. The energy term is GROSS (combustion) energy, not Atwater metabolisable energy, and so is only a secondary cross-check for SJOS-N; Atwater factors could refine it (O-B). Food items with no protein coefficient after the coalesce chain are excluded with a warning naming the count and a few examples (the residual gap-fill, O-B), never silently dropped.

An FBS item that covers several species reaches `biomass_coefs` through one representative `Name_biomass` in items_full. For FBS 2551 `Nuts and products`, ten species reached through `Almonds`, that one row is not representative: on FAO's own food-composition factors for the FBS, protein per 100 g of nut in shell runs from 1.8 g (chestnuts) to 10.3 g (pistachios), and almonds sit near the top at 8.0 (FAO, *Food balance sheets: a handbook*, Rome, 2001, Annex I, PDF page 65, https://www.fao.org/docrep/pdf/011/x9892e/x9892e00.pdf). With `basket = "fao_composition"` (default) the protein density of 2551 is those ten factors weighted by 2010 world production of each species (FAOSTAT production, element 5510, area 351 excluded), 64.7 g per kg against 80 for almonds alone (#1453). It replaces the protein term only, and only on the `"edible_portion"` basis, whose meaning (edible protein per kg of commodity) is what FAO's in-shell factor is. Energy keeps the representative row: FAO's factor is dietary energy, not the gross energy this function carries. Against the pre-2014 FBS, which FAO built from these same factors on what each country actually ate, the basket gives 0.98x 2010 world nut protein over 130 countries, against 1.21x for almonds alone. The world basket moves little over time (61 g/kg on 1961 production, 66 on 2024). `"representative_species"` keeps the single-species coefficient, for continuity and sensitivity analysis.

The `"faostat_fbs"` method is FAOSTAT's own per-capita supply, the independent benchmark for the default. With `data$fbs_supply` injected it is returned unchanged. Otherwise it is read from the `faostat-fbs-old` (1961-2013) and `faostat-fbs-new` (2010-2023) pins: protein and dietary energy per capita per day from item 2901 "Grand Total" (elements 674 and 664), which FAOSTAT reports directly, and population from item 2501 (element 511, thousands, converted to persons). FAOSTAT's energy element is dietary energy as FAOSTAT derives it from food composition, not the gross (combustion) energy of `"whep_native"`, so the two methods' energy columns are not like for like. FAOSTAT areas are resolved onto `area_code` exactly as `read_fbs_population()` resolves them: year by year, dropping any area that resolves to no polity (the regional aggregates and area 351 "China", the aggregate over areas 41, 96, 128 and 214, #939); where more than one FAOSTAT area lands in a bucket-year the per-capita values are population-weighted. `faostat-fbs-new` wins an overlapping `(year, area_code)`. The old pin follows FAOSTAT's pre-2014 FBS methodology and the new one the revised methodology, so a series crossing 2010 changes vintage there.

An area with food but no `population` row has no denominator, so it is absent from the output rather than wrong in it. Those areas are **named at runtime** in a warning, with the

share of food protein that leaves with them, because which areas they are moves with every refresh of the `gdp-population` pin and of the food input: this sentence carried a count of 15 that was already 16 by the time #644 measured it. Read the warning, not a number in the documentation.

For orientation only, the areas with food protein and no population row in any year, measured against the `faostat-fbs-new` pin over the population pin's 1850-2021 span, are 13, headed by the `China` aggregate (351), `Sudan` (276) and `South Sudan` (277), then Comoros, New Caledonia, Bhutan and the small island states. Two of those are known open issues rather than data gaps: 351 is the aggregate whose members carry population separately, and 151 `Netherlands Antilles` is #787. `options(whep.warn_missing_population = FALSE)` silences the warning.

Usage

```
build_food_supply(
  method = c("whep_native", "faostat_fbs"),
  data = list(),
  protein_basis = c("edible_portion", "whole_commodity", "product_nitrogen"),
  basket = c("fao_composition", "representative_species"),
  example = FALSE
)
```

Arguments

`method`	Supply source: `"whep_native"` (default, commodity-balance food tonnes times `whep::biomass_coefs` divided by population) or `"faostat_fbs"` (FAOSTAT FBS per-capita supply, injected or read from the FBS pins).
`data`	Named list of injected inputs. For `"whep_native"`: `cbs_food` (`year`, `area_code`, `item_cbs_code`, `food_t`) and `population` (`year`, `area_code`, `population`) are required, and `biomass_coefs` / `items_full` override the packaged `whep::biomass_coefs` / `whep::items_full`. For `"faostat_fbs"`: `fbs_supply` (`year`, `area_code`, `protein_g_cap_day`, `energy_kcal_cap_day`, `population`) is returned as given if supplied; otherwise `fbs_old` and/or `fbs_new`, the raw pins in their own long FAOSTAT layout (`Area Code`, `Item Code`, `Element Code`, `Year`, `Value`), replace the `whep_read_file()` read of whichever is absent.
`protein_basis`	How the inedible fraction is treated when converting nitrogen density to protein, for `"whep_native"` only: `"edible_portion"` (default) scales the nitrogen density by `Edible_portion`, which is correct when `food_t` is commodity mass while the density applies to the edible part, and agrees best with FAOSTAT FBS; `"whole_commodity"` applies no edible scaling, the behaviour before this argument existed, kept for continuity and sensitivity analysis; `"product_nitrogen"` uses the agronomic `Product_kgN_kgDM` for both the edible and inedible fractions, scaled by `Edible_portion`, ignoring `N_kgN_kgFM`. A missing `Edible_portion` counts as 1.
`basket`	Protein density of multi-species FBS items, for `"whep_native"` with `protein_basis = "edible_portion"` only: `"fao_composition"` (default) weights FAO's

FBS food-composition factors per species by world production (today FBS 2551 `Nuts and products`); `"representative_species"` uses the one `biomass_coefs` row that items_full bridges the item to.

`example` If `TRUE`, return a small fixture instead of computing. Defaults to `FALSE`.

Value

A tibble keyed by `year`, `area_code` with `protein_g_cap_day`, `energy_kcal_cap_day`, `population`, `method_food_supply`, `method_protein_basis` and `method_basket` (both `NA` for `"faostat_fbs"`), plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_food_supply(example = TRUE)
```

`build_footprint` *Compute a footprint end-to-end from an extension table.*

Description

Trace a long-format environmental extension table through the supply chain for one or more years and return a tidy footprint. This wraps the three steps that the footprint driver scripts used to repeat inline: build (or reuse) the input-output model with `build_io_model()`, align the extension to each year's sector labels with `align_extension()`, and trace it with `compute_footprint()`.

The `extension` table is the output of any `build_*_extension()` builder, such as `build_grassland_land_extensi` or `build_livestock_ghg_extension()`: rows keyed by `year`, `area_code` and `item_cbs_code`, with the pressure magnitude in `value_col`.

Usage

```
build_footprint(
  extension,
  years = NULL,
  io = NULL,
  method = c("mass", "value"),
  value_col = "impact_u",
  ...
)
```

Arguments

`extension`	Long-format extension tibble with columns `year`, `area_code`, `item_cbs_code` and the column named by `value_col`.
`years`	Years to compute. Defaults to the distinct years present in `extension`. Ignored when `io` is supplied.
`io`	Optional pre-built `build_io_model()` result (a tibble with one row per year). Supply it to reuse one model across several extensions instead of rebuilding it. When `NULL` (default), it is built for `years`.
`method`	Co-product allocation method passed to `build_io_model()`, `"mass"` (default) or `"value"`. Ignored when `io` is supplied (the model already encodes its allocation).
`value_col`	Name of the extension magnitude column, `"impact_u"` by default.
`...`	Further arguments passed to `compute_footprint()` (e.g. `conserve_extensions`, `report_conservation`).

Value

A tibble of footprint flows as returned by `compute_footprint()`, with an added `year` column.

Examples

```
io <- tibble::tibble(
  year = 2000L,
  Z = list(matrix(c(0, 5, 10, 0), nrow = 2)),
  X = list(c(100, 200)),
  Y = list(matrix(c(85, 195), ncol = 1)),
  labels = list(tibble::tibble(
    index = 1:2,
    area_code = c(1L, 1L),
    item_cbs_code = c(1L, 2L)
  )),
  fd_labels = list(tibble::tibble(area_code = 1L, fd_col = "food"))
)
extension <- tibble::tibble(
  year = 2000L,
  area_code = 1L,
  item_cbs_code = c(1L, 2L),
  impact_u = c(50, 30)
)
build_footprint(extension, io = io)
```

---

build_grass_availability

Build grazable grass availability.

Description

Multi-method wrapper for the grass forage supply ceiling that feeds allocation. The default `lpjml` method reads pinned LPJmL-derived managed-grassland net primary production/availability unless custom artifact data, a custom artifact path, or `run_dir` points to local inputs; `coefficient` applies a per-area grass-yield coefficient and is not yet implemented (it needs a `grass_yield_coef` dataset).

Usage

```
build_grass_availability(method = c("lpjml", "coefficient"), ...)
```

Arguments

`method`	Grass-availability method, `"lpjml"` or `"coefficient"`.
`...`	Passed to the selected method's builder, e.g. `build_grass_availability_lpjml()`.

Value

A tibble of grass availability with a `method_grass` column recording the method used.

Examples

```
build_grass_availability(method = "lpjml", example = TRUE)
```

`build_grass_availability_lpjml` *Build grazable grass availability from an LPJmL run.*

Description

Reads managed-grassland net primary production/availability (the LPJmL grassland CFT) from the pinned WHEP artifact by default. Pass `availability` or `availability_path` to use a custom already-derived artifact; pass `run_dir` to read a finished local LPJmL run instead and convert NPP to grazable above-ground dry-matter availability, the forage supply ceiling for feed allocation. Availability is the production flux, not the realised grazing off-take (the off-take is the intake-validation target, not the supply).

Usage

```
build_grass_availability_lpjml(
  run_dir = NULL,
  years = NULL,
  first_year = NULL,
  shares = grass_access_shares(),
  example = FALSE,
  availability = NULL,
  availability_path = NULL
)
```

Arguments

`run_dir`	Path to the LPJmL run output directory holding `pft_npp.nc` and `cftfrac.nc` (the `scenario_*` output folder). If unset, the pinned `lpjml-grass-availability` artifact is used.
`years`	Integer vector of calendar years to read.
`first_year`	First calendar year of the run's output time axis.
`shares`	Accessibility and conversion parameters from `grass_access_shares()`.
`example`	If `TRUE`, return a small fixture instead of reading a run.
`availability`	Optional already-derived grass availability tibble/data frame. Takes precedence over pinned data and `run_dir`.
`availability_path`	Optional path to an already-derived grass availability artifact (`.parquet`, `.csv`, or `.rds`). Takes precedence over pinned data and `run_dir`.

Value

A tibble with `lon`, `lat`, `year`, `grass_npp_gc_m2`, `grass_avail_dm_t_ha` and `grass_avail_dm_t`.

Examples

```
build_grass_availability_lpjml(example = TRUE)
```

`build_grass_natural_carbon_inputs`
Build grassland and natural-land soil carbon inputs from LPJmL.

Description

Assemble the carbon returned to soil under grassland and natural vegetation as the layer the soil-organic-carbon turnover models consume. The LPJmL-derived net carbon density is read from the pinned `lpjml-grass-natural-net-c` artifact by default, so running LPJmL is not a prerequisite; pass `run_dir` (or set `WHEP_LPJML_RUN_DIR`) to derive it from a finished local run instead, or `data$net_c` to supply it directly. The pin holds only LPJmL-derived quantities: the grazing excreta, both humification fractions and the polity attachment are always computed here, so they never differ between the pinned and the run-derived path. The class carbon input is the net primary production minus harvested carbon (both per-plant-functional-type, `read_lpjml_npp()`), floored at zero and converted to megagrams of carbon per hectare per year (1 gC/m2 = 0.01 MgC/ha). Natural land sums the eleven natural plant-functional-types (they coexist in one stand); grassland takes the stand-area-weighted mean of the rainfed and irrigated grassland stands and charges them one grazing removal and one excreta return, from WHEP's own livestock chain by default and from LPJmL's livestock module on request (`method_grazing`), spread uniformly over the polity's grassland area. The humification fraction is the spontaneous-grass value for grassland and the woody-residue value for natural land (both from residue_humification).

Usage

```
build_grass_natural_carbon_inputs(
  resolution = c("grid", "polity"),
  method_grazing = c("whep", "lpjml"),
  method_natural_hf = c("woody_share", "woody"),
  method_natural_c = c("litterfall", "npp"),
  data = list(),
  years = NULL,
  run_dir = NULL,
  excreta_area_basis = c("luh2_grassland", "charged_grassland", "luh2_all_grassland"),
  grazed_area_basis = c("luh2_grassland", "charged_grassland"),
  example = FALSE
)
```

Arguments

`resolution`	`"grid"` (default, per cell and class) or `"polity"` (aggregated to `area_code`, area-weighting the per-hectare densities).
`method_grazing`	Whose grazing removes carbon from grassland and returns it as excreta. `"whep"` (default) uses WHEP's own estimates: the realised grass intake of `redistribute_feed()` is the removal, converted from dry matter at the package's carbon fraction (`grass_access_shares()$w_c_dm`), and

the grassland `applied` stream of `build_livestock_nutrient_flows()` is the return. LPJmL's own grazing is backed out in full: the class starts from the whole grassland production (`npp_c_mgc_ha_yr`), not from what LPJmL left after grazing it, so no carbon is counted twice. Both `data$livestock_intake` and `data$excreta` are then required, and a requested year the intake table does not cover aborts rather than passing silently as an ungrazed year.

`"lpjml"` uses the model's own livestock module instead (`npp_c_mgc_ha_yr - net_c_mgc_ha_yr`, the `pft_harvestc` grassland bands), and refuses WHEP's excreta with a warning, because that return is already inside the layer. LPJmL 6.1 grazes `UptakeC` off the managed-grassland stand, returns `FecesC` and `UrineC` to it, and books the difference as `pft_harvestc`: on the 1750-2023 run `pft_harvestc = UptakeC - FecesC - UrineC` holds cell by cell to machine precision (maximum relative difference 0 at 1900, 1960 and 2010), so `NPP - pft_harvestc` already carries an excreta return worth 5.9% of grassland production in 1900, 3.8% in 1960 and 4.5% in 2010. Adding WHEP's excreta on top of it, which the `excreta` argument used to do, books that return twice.

The two are alternatives, not tiers: `"whep"` is the default because WHEP's livestock chain, not LPJmL's, is what the rest of the package charges for feed, excretion and manure, so a grassland input built on LPJmL's herd would not reconcile with the manure the same herd applies to cropland. Backing LPJmL's grazing out recovers the carbon MASS but not its dynamics: the production the model simulated was itself shaped by the defoliation it applied, so the whole-production starting point is a known approximation, not an ungrazed counterfactual. Where WHEP's grazing removal exceeds a cell's production the input is floored at zero and the floored carbon is reported in a warning. Recorded in `method_c_input`.

`method_natural_hf`

How natural land's humification fraction is set. `"woody_share"` (default) carbon-weights the residue_humification woody and herbaceous coefficients by the share of each cell-year's natural production that the woody PFTs made; `"woody"` applies the woody coefficient everywhere, which was the previous behaviour. Five of the fourteen natural PFTs are not woody and carry 28.1% of natural production at 2010. Falls back to `"woody"`, with a warning, when the net-carbon layer has no `woody_share` column.Which grassland hectares the grazing excreta is divided by

The excreta carbon is a polity total, so it becomes a per-hectare density by division, and `excreta_area_basis` picks the hectares to divide by. It matters because the density is charged only to cells where LPJmL wrote a grassland stand, while the divisor has always been the polity's **whole** LUH2 grassland area, and `build_carbon_balance()` then multiplies the density by the LUH2 grassland area of each charged cell. Whatever the uncharged hectares would have received is therefore lost.

Measured on the `lpjml-grass-natural-net-c` pin against `read_luh2_landuse()`, the cells with a grassland stand hold 98.6% of the LUH2 grassland area at 2010 (98.4% at 1960, 98.6% at 2020), leaving 46.6 Mha uncharged at 2010. How much excreta carbon that loses depends on where the herds

are: 1.4% if excreta is proportional to grassland area, 13.8% under an equal-carbon-per-polity probe, because the shortfall is concentrated in small polities — 101 of 188 lose more than 1%, 53 more than 10%, 7 more than half, and 13 (islands and city states, each under 0.01 Mha of grassland) have no grassland stand at all and lose everything. The reverse gap is negligible: 6 of 42,391 grassland cell-rows at 2010 have no LUH2 grassland row.

The three bases are alternatives, not fallbacks, and the chosen one is recorded in `method_excreta_area`:

- `"luh2_grassland"` (default, the published behaviour): divide by the polity's whole LUH2 grassland area. Does **not** conserve the polity's excreta carbon whenever a grassland hectare carries no LPJmL stand.
- `"charged_grassland"`: divide by the LUH2 grassland area of the cells the density is actually charged to, so every polity that has a grassland stand keeps its whole excreta carbon by construction. Raises the density on those cells by the reciprocal of the coverage above (1.4% globally on an area weighting, up to 17x in the worst measured polity) and leaves the uncovered hectares at zero, as they already are for net primary production. A polity with no grassland stand anywhere still loses all of its excreta — there is nowhere to charge it — which on the probe above is 13 polities and 6.9% of the carbon.
- `"luh2_all_grassland"`: keep the whole-area divisor and emit a grassland row for every LUH2 grassland cell instead, carrying zero net primary production where LPJmL has no stand (4,117 extra rows at 2010, 8.8% more grassland rows). The only basis that conserves the excreta carbon globally, and it keeps the original spatial spread, at the cost of grassland rows in cells the LPJmL run does not simulate as grassland.

Dividing by the LPJmL grassland **stand** area is deliberately not offered. It would conserve nothing under the area basis `build_carbon_balance()` actually uses, and it would buy little: measured at 1901 the stand area and the LUH2 grassland area agree to a median 0.1% per cell (95.5% of shared cells within 10%, global totals 1560.5 against 1576.0 Mha), because WHEP's LPJmL land-use forcing is itself LUH2-derived.Which grassland hectares the grazing removal is divided by

`grazed_area_basis` is the same question for the flux that leaves the grassland. WHEP's grass intake is a polity total too, so it also becomes a density by division and is also charged only to cells with an LPJmL grassland stand. Under the default divisor the polity's grassland gives up the coverage share of what its herd ate, and the rest is grazed off nothing.

The coverage is the same one measured for the excreta above: 98.6% of the LUH2 grassland area globally at 2010, but 0.68 for Greece, 0.77 for Somalia, 0.82 for Indonesia and 0.87 for the United Kingdom. Weighting each polity by LPJmL's own grazing removal as a stand-in for WHEP's intake (1,100 Tg C at 2010, against the 1,188 Tg C the SOC branch reports at 2020), 1.8% of the grazed carbon is removed from nothing; 50 of 175

polities lose more than 1% of their grassland carbon input to the choice and 11 more than 5%.

The two bases are alternatives, not fallbacks, and the chosen one is recorded in `method_grazed_area`:

- `"luh2_grassland"` (default, the published behaviour): divide by the polity's whole LUH2 grassland area. The density on a charged cell is then the polity's true mean grazing pressure, but the polity gives up less carbon than its herd ate.
- `"charged_grassland"`: divide by the LUH2 grassland area of the cells the density is charged to, so the grassland gives up exactly the carbon the herd removed. The cost is that the whole national herd is then charged to the modelled subset of the pasture, raising the removal density there by the reciprocal of the coverage (1.46x for Greece, 1.29x for Somalia).

The two are not a strict improvement on one another, which is why the default is unchanged: the first conserves the per-hectare density, the second conserves the mass. On the probe above, switching lowers the global grassland plant carbon input by 20.2 Tg C at 2010 (-0.13%; -0.13% at 1960 and -0.14% at 2020) and by up to 8.2% for a single polity. Neither basis reaches the uncovered hectares themselves: `build_carbon_balance()` gives them a grassland pool with a zero carbon input, which is the larger defect of the two and is tracked separately (whep#1146).

`method_natural_c`

Which LPJmL quantity is natural land's carbon input. `"litterfall"` (default) is `litfallc_nv`, what the model actually returns to the soil, excluding the increment retained in living biomass and what fire and land conversion remove; `"npp"` is the natural PFTs' whole primary production, the previous default. Litterfall is the physically correct soil input, and it is a good deal smaller than production where it matters: on the published pin at 2010 the per-cell ratio has a median of 0.844 across cells carrying natural vegetation, but weighted by LUH2 natural area the litter MASS is 0.648 of the production mass (0.632 at 2000), because the most productive cells retain the largest share of their production as growing biomass. The natural class's area-weighted mean input therefore falls from 8.1 to 5.2 MgC/ha/yr, and its equilibrium soil carbon with it. The per-cell ratio declines from 0.902 in the 1750s to 0.838 in the 2000s, so production as the input would carry a CO2-fertilisation trend into the soil. A tail of cells with almost no production carries litter above production (ratio above 2 on 3.2% of natural area, holding 0.7% of the litter mass); it is left as the run wrote it. The `lpjml-grass-natural-net-c` pin carries litterfall since its 2026-09-03 version (the 1750-2023 run); an older pin or run aborts naming the missing column rather than falling back. Recorded in `method_c_input`.

Litterfall **includes root turnover**, so natural land must not receive a separate root term on top, and does not: LPJmL adds each PFT's root turnover carbon to the belowground litter pool and to the same `LITFALLC` accumulator as leaf litter (`src/grass/turnover_grass.c` lines 130-131

and `src/tree/turnover_tree.c` lines 162-163 at PIK-LPJmL/LPJmL, plus tree root exudates at lines 143-144). The run confirms it: at 1760 over 31,889 near-pure natural cells the litterfall/NPP median is 0.894, and **0.965 on grass-dominated cells**, which allocate roughly half their production below ground – an above-ground-only litterfall would sit near 0.5 there. Adding fire closes the pre-industrial steady state at 0.930. Cropland is the opposite case: `build_soil_carbon_inputs()` assembles residues, roots and manure as separate terms, because a crop stand's roots are not an LPJmL litterfall flux.

It is a gross accumulator, not a residual: LPJmL increments it wherever carbon enters the litter pools (turnover, phenological shedding, root exudates, reproduction, mortality, fire-killed but uncombusted biomass, harvest residues). It excludes what land-use conversion releases, which the model books separately as `litfallc_luc` and which is far from small: on the same near-pure natural cells at 1755-1765 it is 14.3% of natural NPP against litterfall's 88.2%. Excluding it is what WHEP needs, because the balance derives its own land-use-change transfer from LUH2 areas and its own stocks; counting the model's conversion litter as a soil input as well would book that carbon twice.

`data` Named list of pre-loaded inputs, each falling back to its reader when absent: `net_c` (the LPJmL net carbon density, `lon`, `lat`, `year`, `land_use`, `npp_c_mgc_ha_yr`; takes precedence over both `run_dir` and the pin); `npp` and `harvestc` (per cell, PFT and year, the `read_lpjml_npp()` output); `stand_frac` (per cell, year and PFT name the managed-grassland stand fractions with columns `lon`, `lat`, `year`, `name_pft`, `stand_frac`). Supplying all three of `npp`, `harvestc` and `stand_frac` derives `net_c` without needing a run directory or the pin. Also: `country_grid`, the polycell support resolved to one row per cell and `area_code` (`lon`, `lat`, `area_code`, `cell_area_frac`), refused when a cell-`area_code` group is duplicated or `NA` (DA-23); `land_use` (per-cell class `area_ha`, used to spread excreta and to area-weight polity output); `excreta` (the `applied` tibble of `build_livestock_nutrient_flows()`, grassland rows carry `applied_c` tonnes C); `livestock_intake` (the `redistribute_feed()` result, the same tibble the nitrogen path takes as `data$livestock_intake`, whose `feed_quality == "grass"` rows carry the grazed dry matter); `residue_humification` (defaults to residue_humification).

`years` Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the inputs cover. Threaded into the default LPJmL NPP, stand-fraction and land-use readers so they slice to the requested years; ignored for inputs supplied via `data`.

`run_dir` Path to a finished LPJmL run output directory holding `pft_npp.nc`, `pft_harvestc.nc` and `cftfrac.nc` (the `scenario_*` output folder). `NULL` (default) uses `WHEP_LPJML_RUN_DIR` when set, and the pinned artifact otherwise.

`excreta_area_basis`
Which grassland hectares the polity's grazing excreta carbon is divided by: `"luh2_grassland"` (default), `"charged_grassland"` or `"luh2_all_grassland"`. See the section below; the choice is recorded in `method_excreta_area`.

Only bites under `method_grazing = "whep"`; `"lpjml"` returns no WHEP excreta to divide.

`grazed_area_basis`
: Which grassland hectares the polity's grass intake carbon is divided by: `"luh2_grassland"` (default) or `"charged_grassland"`. See the section below; the choice is recorded in `method_grazed_area`. Only bites under `method_grazing = "whep"`; `"lpjml"` takes the run's own harvest, which is already per cell.

`example`
: If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble keyed by `(lon, lat, area_code, year, land_use)` at `"grid"` resolution (or `(area_code, year, land_use)` at `"polity"`), with `c_input_mgc_ha_yr`, `humified_fraction`, `method_c_input`, `method_excreta_area` and `method_grazed_area`, for `land_use` in `"grassland"` and `"natural"`, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once.

polity_coverage_gaps() reports the stand-in rows of a built table, and options(whep.polity_mapping_status = "flag") (or "status") carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

LPJmL run net primary production and harvested carbon; grassland and natural carbon inputs per the WHEP historical carbon-balance design.

Examples

```
build_grass_natural_carbon_inputs(example = TRUE)
```

`build_grassland_intensity_classes`

Classify grassland cells as intensive or extensive through time

---

Description

Moves the IMAGE 2010 intensive/extensive grassland map behind the Schulte-Uebbing et al. (2022) critical-nitrogen archive (Zenodo doi:10.5281/zenodo.6395016) through time, one row per 0.5-degree cell and year. It is the class table the land_use = "all" comparison of build_n_boundary_exceedance() consumes (issue #1285).

Each cell belongs to its 2010 country: the crosswalk area code holding the largest share of the cell (ties to the smallest code). The national grazing density of a polity in a year is the livestock units of grazer species it reported that year per hectare of WHEP grassland (pasture plus rangeland) inside its territory. A cell's density_ratio is the density of the polity its 2010 country belonged to in the year, over the density of that country in 2010. Each 2010 country's intensive share of IMAGE grassland is scaled by the area-weighted mean ratio of its cells and clamped to [0, 1]; cells are then switched by rank of 2010 manure N per hectare until the intensive area is nearest the target (demotion from the lowest-manure intensive cell up, promotion from the highest-manure extensive cell down). At 2010 the ratio is 1 and the classes equal the IMAGE map.

Livestock units are reported under the polity that existed in the year – the USSR, not Russia, in 1961 – while the crosswalk knows present-day area codes only. Each present-day code is resolved to its reporting polity in the year with resolve_polity_lineage(), against the polities the livestock units are actually reported under that year. A lineage answer that is a different interval of a polity another present-day code reports under itself is refused (it would book, say, Belgium on the Netherlands). A code still unresolved is placed on a reporting polity whose published successors reach it (the Baltic states and Belgium-Luxembourg), when exactly one does, after WHEP's own polity_area_code fold has had its chance (Sudan and South Sudan are reported folded into area 206 from 2012). Grassland of a code that resolves to no reporting polity enters no denominator, and its cells keep their IMAGE class.

A country's density is its own reporting polity's density in the years it reports on its own (density_basis = "own"), and the shared bucket's density in the years it is reported folded

into one after a split (`"bucket"`, Sudan and South Sudan in area 206 from 2012). For the years before it reported on its own, the country keeps its own level and borrows only the trend of the polity its cells reported under (`"chained_predecessor_trend"`): with `t0` the first later year the country has a positive density and `t0_pred` the last year up to `t0 - 1` the predecessor has one, `D_k(t) = D_k(t0) * D_pred(t) / D_pred(t0_pred)`. Chains of successions are linked one step at a time from the present back (Yugoslavia, then Serbia and Montenegro, then Serbia). A missing link leaves the density, and the ratio, undefined. Densities are therefore read for every year from the earliest requested one to the last year both the livestock units and the grassland surface cover, not only the requested years. Kazakhstan shows why: in 1961 the USSR's national density is six times Kazakhstan's own 2010 density, so taking it as Kazakhstan's level would promote most of Kazakhstan's grassland to intensive.

Declared assumptions, each a constructed choice rather than a measurement:

- IMAGE's classes are livestock production systems (mixed systems intensive, pastoral systems extensive), not a threshold on grazing density. The national density trend is a proxy for the trend in that split.
- The ranked reclassification has no published precedent (issue #1285).
- WHEP grassland in a cell with no IMAGE grassland is extensive, stamped `"no_image_grassland"` (maintainer decision 2026-09-24, issue #1285). Its allowance is applied downstream.
- Livestock units are the `"LU"` rows of `get_primary_production()` for the `animals_codes` "Grazers", from `liv_lu_coefs`, whose source is generic ("standard livestock unit definitions from FAO and European agricultural statistics"). Only density ratios are used, so the coefficients matter through changes in species mix alone.
- Chaining a successor onto its predecessor's trend is a constructed rule (maintainer decision 2026-09-24, issue #1285): it assumes the successor's density moved with its predecessor's before it reported on its own.
- A cell whose 2010 country has no finite, positive 2010 density, or whose polity has no density in the year, keeps its IMAGE class and is stamped `"image2010_fixed_no_density"`. So is an IMAGE grassland cell the crosswalk assigns to no country.

Usage

```
build_grassland_intensity_classes(
  years,
  data = list(),
  cell_support = c("year_aware", "constant"),
  example = FALSE
)
```

Arguments

`years`	Integer vector of years to classify. 2010 is always read as the base of the density ratio; it is returned only when requested.
`data`	Optional named list of injected inputs, each replacing its real read: • `stock_lu`: `get_primary_production()`-shaped rows with `year`, `area_code`, `item_prod_code`, `unit` and `value`. Default: that function from the

earliest requested year (or 2010) to the last year of the grassland surface.

- `gridded_pasture`: `lon`, `lat`, `year`, `pasture_ha`, `rangeland_ha`. Default: the parquet at `WHEP_GRIDDED_PASTURE_PATH` when set, else the `spatialize-gridded-pasture` pin.
- `cell_polity`: `lon`, `lat`, `area_code`, `polity_frac`. Default: the support `cell_support` selects. An injected table replaces it and is stamped `"supplied"`.
- `grassland_layers`: one row per IMAGE cell with `cell_id`, `lon`, `lat`, `a_crop_ha`, `a_gr_int_ha`, `a_gr_ext_ha`, `manure_int_n_kg`, `manure_ext_n_kg`, `image_class_2010` and `image_region`. Default: the critical-nitrogen archive, resolved as `read_critical_n()` resolves it.

`cell_support` The cell-to-country support:

- `"year_aware"` (default): `build_cell_polity()` read at a present-day year from the polycell support, so `polity_frac` is the polity's share of the cell's measured land. It covers every land cell, coastal cells whose centre lies at sea included.
- `"constant"`: the year-invariant `spatialize-cell-polity-fraction` pin (`WHEP_POLITY_FRACTION_PATH` overrides it). It is restricted to the LPJmL grid, so it gives no country to 2,643 IMAGE grassland cells holding 56.8 Mha (issue #1297); those keep their IMAGE class.

`example` If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble, one row per cell and year, sorted by `cell_id` and `year`. Cells are those with IMAGE grassland plus those with WHEP grassland in any requested year. Columns:

- `cell_id`: canonical 0.5-degree cell key.
- `lon`, `lat`: cell centre, degrees.
- `year`: calendar year.
- `country_2010`: crosswalk area code holding the largest share of the cell; `NA` for a cell the crosswalk does not cover.
- `image_region`: IMAGE region (1-26); `NA` outside the archive.
- `a_crop_ha`: IMAGE 2010 cropland area, ha (0 outside the archive).
- `grass_ha_image`: IMAGE 2010 intensive plus extensive grassland, ha (0 where IMAGE has none).
- `whep_grass_ha`: WHEP pasture plus rangeland in the year, ha (0 where the surface carries none).
- `image_class_2010`: `"intensive"`, `"extensive"`, or `NA` where IMAGE has no grassland.
- `grassland_class`: `"intensive"` or `"extensive"`.
- `density_ratio`: the cell's grazing-density ratio to 2010; `NA` where undefined.
- `target_share`: the 2010 country's target intensive share; `NA` where undefined and for cells with no IMAGE grassland.

- `method_grassland_split`: `"image2010_density_rank"`, `"image2010_fixed_no_density"` or `"no_image_grassland"`.
- `density_basis`: how the 2010 country's density in the year was formed: `"own"`, `"chained_predecessor_trend"` or `"bucket"`; `NA` where the cell has no 2010 country or its code resolves to no reporting polity that year.
- `method_cell_polity`: the cell support that assigned the countries: `"year_aware"`, `"constant"` or `"supplied"`.

Examples

```
build_grassland_intensity_classes(example = TRUE)
```

---

build_grassland_land_extension

Build the native grassland land extension.

Description

Produce a grassland land extension keyed by `(year, area_code, item_cbs_code)`, replacing the grassland rows that used to come from the external `land_fp` pin.

Two area sources are available, selected with `source`:

- `"luh2"` (default): permanent and temporary grassland area (item_cbs 3000 and 3002, LUH2 pasture and rangeland) taken from `build_primary_production()`. This shares the gridded LUH2 land-use basis used by the crop land extensions and by livestock spatialisation. Rotational fallow (item_cbs 3003) is excluded because the `cropgrids_fallow` crop extension already attributes fallow to crops, so counting it here too would double count it.
- `"faostat_pasture"`: FAOSTAT "Permanent meadows and pastures" area (Land Use item 6655), the statistics-based basis comparable to most published footprint studies.

Two metrics are available, selected with `grassland_metric`:

- `"occupation"` (default): the full grassland area is charged as occupied land.
- `"active_grazing"`: grassland is capped at the area implied by actual grazing intake (the `"grass"` feed in `get_feed_intake()`) divided by a usable grass yield, so ungrazed or marginal rangeland is not charged.

Usage

```
build_grassland_land_extension(
  source = c("luh2", "faostat_pasture"),
  grassland_metric = c("occupation", "active_grazing"),
  usable_grass_yield_dm_t_ha = 2.06,
  data = list(),
  example = FALSE
)
```

Arguments

`source`	Grassland area source, `"luh2"` (default) or `"faostat_pasture"`.
`grassland_metric`	Grassland land metric, `"occupation"` (default) or `"active_grazing"`.
`usable_grass_yield_dm_t_ha`	Usable grass yield in dry-matter tonnes per hectare, used only by `"active_grazing"`. Defaults to `2.06`.
`data`	Optional named list of pre-loaded inputs to avoid remote reads: `primary_prod` (for `source = "luh2"`), `landuse` (the `faostat-landuse` pin, for `source = "faostat_pasture"`) and `feed_intake` (for `grassland_metric = "active_grazing"`). Each falls back to its reader (`get_primary_production()`, `whep_read_file()`, `get_feed_intake()`) when absent.
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (grassland area in hectares) and `method_grassland` (the chosen metric), plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends,

and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_grassland_land_extension(example = TRUE)
```

---

build_grazing_feed_footprint

Build a grazing-land footprint by forward feed-allocation.

Description

End-to-end grazing-land footprint for one year on real WHEP data. Grazing land (the grassland extension) is pushed forward onto livestock meat and milk via `allocate_grazing_to_products()`, then routed to consuming countries through the bilateral meat-trade network with `compute_footprint_balance()`. This attributes grazing land to the meat and dairy consumption that drives it – the chain the Leontief footprint leaks because grass is non-productive and not traded.

The inputs are assembled from `build_grassland_land_extension()`, `get_feed_intake()`, `get_primary_production()` and `get_bilateral_trade()`. Supply any of them through `data` to reuse cached inputs or to test without remote reads.

`intake_basis` selects the intake that splits grazing land across animals. The default `"grazer_forage"` uses every grazing animal's grass plus roughage-residue intake, because the feed model classes the forage of major extensive-grazing countries (for example Australia, the United States and Argentina) as residues rather than grass; keying on the `"grass"` feed type alone would silently strip that land. `"grass"` reproduces the narrower grass-feed basis for sensitivity analysis. Land in countries with no grazer intake at all cannot be attributed and is reported via a warning.

Usage

```
build_grazing_feed_footprint(
  year,
  products = c("all", "meat_milk"),
  intake_basis = c("grazer_forage", "grass"),
  data = list(),
  example = FALSE
)
```

Arguments

`year`	Year to build the footprint for.
`products`	Co-product split passed to `allocate_grazing_to_products()`. `"all"` (default) or `"meat_milk"`.
`intake_basis`	Intake used to split land across animals, `"grazer_forage"` (default) or `"grass"`.
`data`	Optional named list of pre-built inputs, any of `grass_land`, `grazer_intake`, `livestock_production` and `trade`. Each falls back to its builder when absent.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble with `area_code` (consuming country), `item_cbs_code` (meat or milk item), `value` (embodied grazing land) and `method` (`"grazing_feed_allocation"`).

Examples

```
build_grazing_feed_footprint(example = TRUE)
```

`build_gridded_landuse` *Build gridded landuse dataset*

Description

Disaggregate country-level FAOSTAT crop harvested areas to a 0.5-degree grid. This reproduces the core spatialization workflow of the LandInG toolbox, adapted to WHEP conventions and tidy data structures.

The algorithm follows three main steps:

1. Each crop's country total is distributed to grid cells proportionally to a spatial reference pattern (e.g. Monfreda) weighted by gridded cropland extent (e.g. LUH2/HYDE).
2. If total allocated harvested area in any cell exceeds its capacity (cropland times multi-cropping suitability), excess is iteratively redistributed using a logit-based transformation.
3. Individual crops are aggregated into crop functional types (CFTs).

Usage

```
build_gridded_landuse(
  country_areas,
  crop_patterns,
  gridded_cropland,
  country_grid,
  config = list()
)
```

Arguments

`country_areas` A tibble with country-level crop harvested areas. Expected columns:

- `year`: Integer year.
- `area_code`: Country code (numeric, matching WHEP polities).
- `item_prod_code`: FAOSTAT item code for the crop.
- `harvested_area_ha`: Total harvested area in hectares.
- `irrigated_area_ha`: Irrigated harvested area in hectares (optional, defaults to 0).

`crop_patterns` A tibble with per-cell spatial crop patterns. Expected columns:

- `lon`: Longitude of cell centre.
- `lat`: Latitude of cell centre.
- `item_prod_code`: FAOSTAT item code.
- `harvest_fraction`: Cropping intensity (Monfreda area divided by reference cropland).

`gridded_cropland`
A tibble with per-cell cropland extent. Expected columns:

- `lon`: Longitude of cell centre.
- `lat`: Latitude of cell centre.
- `year`: Integer year.
- `cropland_ha`: Total cropland area in hectares.
- `irrigated_ha`: Irrigated cropland in hectares (optional, defaults to 0).

`country_grid` A tibble mapping grid cells to countries. Expected columns:

- `lon`: Longitude of cell centre.
- `lat`: Latitude of cell centre.
- `area_code`: Country code.
- `cell_area_frac` (or `polity_frac`, `area_frac`, `country_frac`): This polity compartment's share of the physical cell, a partition summing to 1 over the polities that overlap the cell. Required: a grid carrying no share is refused, because defaulting it to 1 gives a border cell wholly to one polity. Pass 1 only where the polity does own the whole cell. A land fraction (`landfrac`) is a different quantity and is refused rather than reinterpreted. Optional columns:
- `polycell_id`, `cell_id`: Stable compartment/cell identifiers preserved in outputs when present.
- `year` or validity intervals (`valid_from`/`valid_to`, `start_year`/`end_year`, `from_year`/`to_year`) for historical, time-varying polity overlays. The start bound is inclusive; the end bound is **exclusive at a succession** and **inclusive at the open end**, so 2014 selects `"RUS-2014-2025"` and not `"RUS-1991-2014"`, while 2025 still selects `"RUS-2014-2025"` because no later interval of that compartment follows it. See polities for the full rule.

`config` Named list of optional extras. Unknown keys raise an error. Recognised keys:

- `years`: Integer vector of years to spatialize. If `NULL` (default), all years present in `country_areas` are processed. When supplied, `country_areas`, `gridded_cropland`, and `type_cropland` are filtered to this set before processing.
- `cft_mapping`: A tibble mapping FAOSTAT items to CFT names (`item_prod_code`, `cft_name`). If `NULL`, no CFT aggregation is performed and individual crop results are returned.
- `type_cropland`: A tibble with per-cell, per-year, per-type cropland (`lon`, `lat`, `year`, `luh2_type`, `type_ha`, `type_irrig_ha`). When provided alongside `type_mapping`, each crop is allocated only into cells containing its LUH2 type. If `NULL`, falls back to total cropland.
- `type_mapping`: A tibble (`item_prod_code`, `luh2_type`) that maps each crop to its LUH2 type. If `NULL`, type-aware allocation is disabled even when `type_cropland` is provided.
- `multicropping`: A tibble with per-cell multi-cropping suitability factors. Required columns: `lon`, `lat`, `mc_rainfed`, `mc_irrigated`. An optional `year` column keys factors to year (one row per cell per year); when present, the table is filtered to the current year before the capacity constraint is applied. When absent, the table is treated as a static spatial layer applied to every year. If `NULL` (default), the capacity constraint still runs with `mc_rainfed = mc_irrigated = 1` (harvested area capped at physical cropland).
- `max_iterations`: Maximum iterations for the redistribution loop. Default: `1000L`.
- `polity_support`: the unfolded polity support, used to reconcile `country_areas`' polity vintage with a year-aware `country_grid`'s. Absent, the reconciliation is skipped, which is correct for a snapshot grid. See `read_polycell_support()`.
- `pattern_signal_floor`: The `harvest_fraction` below which a `crop_patterns` cell is treated as float underflow rather than an allocated area, and zeroed before the placement weights are formed. Default `1e-12`, argued from EarthStat's own float32 precision in `.crop_pattern_signal_floor()`. `0` restores the untoleranced behaviour of whep#1070, in which a (country, crop) whose whole pattern is underflow is placed proportional to that underflow instead of uniformly.
- `expansion_threshold`: defunct. It was accepted but never changed the allocation, and is dropped with a warning (whep#1001). LandInG's expansion step is not implemented.
- `area_key`: Which area code the output is keyed on, `"grid"` (default) or `"polity_area"`. See *Which area code the output is keyed on.*
- `mc_factor`: Which multi-cropping factor sets the capacity ceiling of a granted-depth compartment, `"unit"` (default) or `"national"`. See *The capacity ceiling and its breach.* A grid with no `level_polity_code` has no unit, so the two are the same table there and a level-0 run is unaffected.
- `pattern_extension`: `"none"` (default) or `"granted_units"`. Under `"granted_units"`, a granted-depth compartment carrying no

`crop_patterns` row for an item the national table gives it a target for gains one at `harvest_fraction = 0`, so the crop is placed uniformly over that unit's cropland of its LUH2 type instead of being dropped. Level-0 rows are never extended.

Value

A tibble with gridded crop (or CFT) harvested areas. Columns:

- `lon`, `lat`: Cell centre coordinates.
- `year`: Integer year.
- `area_code`: WHEP polity code for this cell compartment.
- `polity_area_code`, `reporting_polity_code`, `reporting_polity_name`, `reporting_polity_has_geometry`: Polity metadata for `area_code`.
- `grid_area_code`: Only under `area_key = "polity_area"`; the reporting code the engine allocated on.
- `polycell_id`, `cell_id`: Preserved when supplied in `country_grid`.
- `crop_name` or `cft_name`: Crop or CFT identifier.
- `rainfed_ha`: Rainfed harvested area in the cell.
- `irrigated_ha`: Irrigated harvested area in the cell.

The per-compartment capacity breach is a second table and is therefore **not** returned here: this function's return value is a contract two in-package consumers and every `run_spatialize()` caller already depend on, and an attribute would ride through every dplyr verb into comparisons that are about the allocation. Allocate at a granted depth with `allocate_level_crops()`, which returns the breach beside the allocation. See *The capacity ceiling and its breach*.

The capacity ceiling and its breach

The ceiling is soft: `.redistribute_country_dt()` rescales every crop back to its target after the logit passes, so where a compartment's cells cannot hold the target the target wins and the ceiling gives way. That excess is measured rather than absorbed, and returned as the `breach` table of `allocate_level_crops()`: one row per compartment, cell, item and `mc_basis`, with `rf_over` and `ir_over` in hectares, which T14's reconciliation consumes as a table and never as warning text. A cell's ceiling is shared by every item in it, so the cell's excess is attributed to items **pro rata by their allocated area** – no other split is available, and summing the rows of a cell recovers its excess exactly.

`mc_basis` records which multi-cropping factor the row was scored against. Rows carrying a `level_polity_code` are scored twice: at `"national"`, the factor the `multicropping` layer supplies (which is a national figure broadcast to every cell, so it inflates a unit's breach by construction), and at `"unit"`, the unit's own implied intensity – its whole allocated harvested area over the same physical cropland the ceiling uses, floored at 1 because a factor below 1 would forbid single cropping on land that exists. `in_force` marks the basis `mc_factor` selected, which is the one the redistribution ran against.

The unit factor makes a unit's *aggregate* ceiling equal its own allocation by construction, so a breach at `"unit"` is always a within-unit concentration, never a shortfall of the unit as a whole; that is the difference the pair of bases exists to show. The plan phrases the

unit factor as unit harvested over unit *herbaceous* LUH2 cropland; the ceiling carries no LUH2 type split, so it is computed over the same cropland the ceiling itself multiplies, and a herbaceous-only denominator would leave the two inconsistent.

Which area code the output is keyed on

The chain allocates *from* a national table keyed on `area_code` and *into* a `country_grid` keyed the same way, so both sides speak the raw reporting vocabulary the grid was rasterized in. WHEP's polity-keyed national tables are aggregated on `polity_area_code` instead, a bucket that a reporting code need not equal: `276` Sudan and `277` South Sudan both fall in bucket `206`. Every such output row therefore carries two territorial keys that disagree, and whether a consumer joins on `area_code` or on `polity_area_code` decides whether Sudan exists in its result (whep#582).

`area_key` selects which of the two the output carries. It is not a fallback: `"grid"` is the default, reproduces today's codes bit-for-bit, and warns naming the codes that cannot join; `"polity_area"` resolves each code to its bucket through polity_area_crosswalk before the polity columns are attached, so `area_code` and `polity_area_code` agree in every row. It respects `options(whep.unfold_rest_of_world)` (see `folded_reporting_areas()`), so the output and the national tables agree about where a Rest-of-World member's rows belong.

Under `"polity_area"` the raw reporting code is **carried, not replaced**: the output gains `grid_area_code`, joined with `+` where two reporting areas of one bucket meet in a cell and their rows collapse. So the fold stays recoverable at the join rather than baked into the output, the shape `build_cell_polity()` adopted for the same reason (whep#579).

Methodology

This function reimplements the spatial crop allocation from the LandInG toolbox (Ostberg et al. 2023, doi:10.5194/gmd-16-3375-2023) with the following extensions:

- LUH2 crop-functional-type constraints (`type_cropland` + `type_mapping` parameters) restrict each crop to cells containing its LUH2 type (c3ann, c4ann, c3per, c3nfx). LandInG allocates to total cropland without type constraints.
- MIRCA2000 crop-specific irrigated fractions (Portmann et al. 2010) for irrigation distribution, falling back to LUH2-proportional allocation.

Data sources

- Country areas: FAOSTAT QCL via `build_primary_production`
- Crop patterns: EarthStat / Monfreda et al. (2008)
- Gridded cropland: LUH2 v2h (Hurtt et al. 2020)
- Irrigation: MIRCA2000 (Portmann et al. 2010) + LUH2

Examples

```
# Minimal example with toy data
country_areas <- tibble::tribble(
  ~year, ~area_code, ~item_prod_code, ~harvested_area_ha,
```

```
  2000L, 1L, 15L, 1000
)
crop_patterns <- tibble::tribble(
  ~lon, ~lat, ~item_prod_code, ~harvest_fraction,
  0.25, 50.25, 15L, 0.6,
  0.75, 50.25, 15L, 0.4
)
gridded_cropland <- tibble::tribble(
  ~lon, ~lat, ~year, ~cropland_ha,
  0.25, 50.25, 2000L, 800,
  0.75, 50.25, 2000L, 500
)
country_grid <- tibble::tribble(
  ~lon, ~lat, ~area_code, ~cell_area_frac,
  0.25, 50.25, 1L, 1,
  0.75, 50.25, 1L, 1
)
build_gridded_landuse(
  country_areas, crop_patterns, gridded_cropland, country_grid,
  config = list(years = 2000L)
)
```

`build_gridded_livestock` *Build gridded livestock dataset*

Description

Disaggregate country-level FAOSTAT livestock stocks and emissions to a 0.5-degree grid. Each species group uses a tailored spatial proxy:

- **Ruminants** (cattle, buffalo, sheep/goats, equines): LUH2 managed pasture (`pastr`) plus rangeland (`range`), optionally weighted by a static manure-intensity reference (West et al. 2014).
- **Confined animals** (pigs, poultry): LUH2 aggregate cropland, reflecting intensive farming co-location with crop production.
- **Range specialists** (camels): LUH2 rangeland only.
- **Mixed** (other animals): 50/50 blend of pasture and cropland.

For each country, year, and species group the function distributes the national total proportionally to cell-level proxy weights:

$$\text{cell} = \frac{w_i}{\sum_{j \in \text{country}} w_j} \times T_{\text{country}}$$

where w_i is the proxy weight in cell i (land-use hectares times optional reference-pattern intensity) and T is the country total (heads or emissions).

Methodology:

Livestock spatialization is not covered by LandInG (Ostberg et al. 2023), which focuses on crops only. The approach here extends the LandInG framework by using the same LUH2-based spatial proxies (pasture, rangeland, cropland) for livestock distribution.

Country-level data comes from `build_primary_production()` (stocks) and the `faostat-emissions-livestock` pin (CH4/N2O emissions), with predecessor redistribution and pre-1961 backfill already applied.

The Zenodo livestock density input (Heinke 2025, doi:10.5281/zenodo.14946695) provides an alternative calibrated LSU/ha reference for use with the `glw_density` parameter.

Data sources and references:

Source	Use
FAOSTAT Production_Livestock (FAO 2024)	Country-level heads
FAOSTAT Emissions_livestock (FAO 2024)	Enteric CH4, manure CH4/N2O
LUH2 v2h (Hurtt et al. 2020)	Time-varying pasture + cropland
West et al. (2014)	Static manure-N intensity reference
GLW3 (Gilbert et al. 2018)	Species-specific density (optional)
Heinke (2025)	Calibrated LSU/ha density (optional)
IPCC 2006/2019	N-excretion rates, emission factors

Usage

```
build_gridded_livestock(
  livestock_data,
  gridded_pasture,
  gridded_cropland,
  country_grid,
  species_proxy = NULL,
  manure_pattern = NULL,
  glw_density = NULL,
  grass_productivity = NULL,
  years = NULL,
  proxy_method = c("luh2", "glw3"),
  area_key = c("grid", "polity_area"),
  polity_support = NULL
)
```

Arguments

`livestock_data`

A tibble with country-level livestock data. Required columns:

- `year`: Integer year.
- `area_code`: Country code (WHEP polities).
- `species_group`: Livestock functional-type name (e.g. `"cattle"`, `"pigs"`, `"poultry"`).

- `heads`: Live animal count (number of head). Any additional numeric columns (e.g. `enteric_ch4_kt`, `manure_ch4_kt`, `manure_n2o_kt`, `manure_n_mg`) are distributed to the grid using the same proportional weights as `heads`.

`gridded_pasture`

A tibble with annual gridded pasture extent. Required columns:

- `lon`, `lat`: Cell centre coordinates (0.5 degree).
- `year`: Integer year.
- `pasture_ha`: Managed pasture area in hectares (LUH2 `pastr`).
- `rangeland_ha`: Rangeland area in hectares (LUH2 `range`).

`gridded_cropland`

A tibble with annual gridded cropland extent. Required columns:

- `lon`, `lat`: Cell centre coordinates.
- `year`: Integer year.
- `cropland_ha`: Total cropland area in hectares.

`country_grid` A tibble mapping grid cells to countries. Required columns:

- `lon`, `lat`: Cell centre coordinates.
- `area_code`: Country code.
- `cell_area_frac` (or `polity_frac`, `area_frac`, `country_frac`): This polity compartment's share of the physical cell, a partition summing to 1 over the polities that overlap the cell. Required: a grid carrying no share is refused, because defaulting it to 1 gives a border cell wholly to one polity. Pass 1 only where the polity does own the whole cell. A land fraction (`landfrac`) is a different quantity and is refused rather than reinterpreted. Optional columns:
- `polycell_id`, `cell_id`: Stable compartment/cell identifiers preserved in outputs when present.
- `year` or validity intervals (`valid_from`/`valid_to`, `start_year`/`end_year`, `from_year`/`to_year`) for historical, time-varying polity overlays. The start bound is inclusive; the end bound is **exclusive at a succession** and **inclusive at the open end**, so 2014 selects `"RUS-2014-2025"` and not `"RUS-1991-2014"`, while 2025 still selects `"RUS-2014-2025"` because no later interval of that compartment follows it. See polities for the full rule.

A `livestock_data` row whose `area_code` has no cell in a year, but whose polity_area_crosswalk bucket does, is folded onto that bucket for that year (and summed with any row already there), with a message. A bucket-keyed grid holds Sudan and South Sudan only as 206, while the livestock table keys them on 276 and 277 from 2012. A code with cells of its own is never folded.

`species_proxy` A tibble mapping each `species_group` to its spatial proxy type: `"pasture"`, `"cropland"`, `"rangeland"`, or `"mixed"`. Required columns:

- `species_group`: Group name (must match `livestock_data`).
- `spatial_proxy`: One of `"pasture"`, `"cropland"`, `"rangeland"`, or `"mixed"`. If `NULL`, a default mapping is used (see Details).

`manure_pattern`
: A tibble with static manure-intensity weights (e.g. from West et al. 2014). Optional. Expected columns:
  - `lon`, `lat`: Cell centre coordinates.
  - `manure_intensity`: Relative intensity (kg N per ha or similar). Values are used multiplicatively with the land-use proxy. If `NULL`, land-use weights are used alone.

`glw_density`
: A tibble with species-specific gridded livestock density from GLW3 (Gilbert et al. 2018). Required when `proxy_method = "glw3"`; ignored, with a warning, under `proxy_method = "luh2"`. Expected columns:
  - `lon`, `lat`: Cell centre coordinates.
  - `species_group`: Must match `livestock_data`.
  - `density`: Heads per cell (reference year ~2010).
  - `glw_variant`: Optional, `"DA"` or `"AW"`; `read_glw_density()` always supplies it. A table mixing the two products is refused, since one recorded label cannot describe both geographies. Under `"glw3"` it **replaces** the LUH2 proxy for every group, still masked by that year's LUH2 extent so a cell whose land use has gone receives nothing.

`grass_productivity`
: A tibble with grass productivity per cell (`lon`, `lat`, `grass_npp`) from `read_lpjml_grass_productivity()`. Optional. When provided, it multiplies the `pasture`/`rangeland` (grazer) proxy weights so animals follow grass production rather than area alone; cropland/mixed proxies are unaffected. If `NULL`, area proxies are used alone.

`years`
: Integer vector of years to spatialize. If `NULL` (default), all years present in `livestock_data` are processed. When supplied, `livestock_data`, `gridded_pasture`, and `gridded_cropland` are filtered to this set before processing.

`proxy_method`
: Which spatial proxy carries the within-country weight: `"luh2"` (default) or `"glw3"`, validated with `rlang::arg_match()`. They are alternatives, never fallbacks: under `"glw3"` a `NULL` `glw_density`, or a `species_group` that table has no positive cell for, aborts instead of quietly reverting to the LUH2 proxy. The resolved value is recorded per row in `method_livestock_proxy`. See *Which livestock proxy the weights come from.*

`area_key`
: Which area code the output is keyed on: `"grid"` (default, the reporting codes `livestock_data` and `country_grid` are keyed on) or `"polity_area"` (the polity_area_crosswalk bucket national tables are aggregated on). See `build_gridded_landuse()`'s *Which area code the output is keyed on.*

`polity_support`
: The UNFOLDED polity support, used to reconcile `livestock_data`'s polity vintage with a year-aware `country_grid`'s before the match is judged. `NULL` (default) skips the reconciliation, which is correct for a snapshot grid. It cannot be recovered from `country_grid`: the level-0 fold summarises `polity_code` away, so a grid alone cannot say which polity holds a cell. `read_polycell_support()` returns it, and `run_spatialize()` passes it automatically.

Value

A tibble with gridded livestock data. Columns:

- `lon`, `lat`: Cell centre coordinates.
- `area_code`: WHEP polity code for this cell compartment.
- `polity_area_code`, `reporting_polity_code`, `reporting_polity_name`, `reporting_polity_has_geometry`: Polity metadata for `area_code`.
- `grid_area_code`: Only under `area_key = "polity_area"`; the reporting code the engine allocated on.
- `polycell_id`, `cell_id`: Preserved when supplied in `country_grid`.
- `year`: Integer year.
- `species_group`: Livestock functional type.
- `heads`: Allocated live animal count.
- Any additional numeric columns from `livestock_data` (e.g. `enteric_ch4_kt`, `manure_ch4_kt`).
- `method_livestock_proxy`: Which proxy produced the weights for this row, one of `"luh2_area"`, `"luh2_grass"`, `"glw3_da"`, `"glw3_aw"` (or plain `"glw3"` for a `glw_density` table carrying no `glw_variant`). Constant within a `(year, species_group)` block.

Which livestock proxy the weights come from

`proxy_method` selects the within-country weight, and every output row records the resolved value in `method_livestock_proxy`:

- `"luh2_area"`: LUH2 extent alone (`proxy_method = "luh2"`).
- `"luh2_grass"`: LUH2 extent times grass NPP (`proxy_method = "luh2"` with `grass_productivity` supplied). The grass weighting reaches the `pasture` and `rangeland` proxies only, so `cropland` and `mixed` groups in the same call stay `"luh2_area"`. The label is per species group, not per cell: a grazer cell with no `grass_npp` keeps its area weight but still travels under `"luh2_grass"`, because what the column records is the weighting regime the group ran under.
- `"glw3_da"` / `"glw3_aw"`: GLW3 density masked by that year's LUH2 extent (`proxy_method = "glw3"`), from the dasymetric or the areal-weighted product respectively. The two are different within-country geographies, so the label names which one: it is read off `glw_density`'s `glw_variant` column, which `read_glw_density()` stamps, and therefore cannot disagree with the raster the weights came from. A `glw_density` built by hand, carrying no such column, travels as plain `"glw3"`.

The default stays `"luh2"` even though `"glw3"` is the better-informed proxy. GLW3 now has a data mechanism – `read_glw_density()`, the `WHEP_GLW3_DIR` environment variable and `inst/scripts/download/download_glw3.R` (whep#1000, task T15a-ii) – but it is an opt-in local raster set, so a `"glw3"` default would abort every run on a machine that has not fetched it, and GLW3 covers nine of the eleven species groups (not `camels`, not `other`). This is a deliberate, documented deviation from "the default is the most rigorous available method", of the same shape as the interim `area_key = "grid"` default in `R/spatialize_compartments.R`; whep#1000 task T20 is the gate that revisits it.

Species groups must be mapped, not guessed

Every `species_group` in `livestock_data` must have a row in `species_proxy`; an unmapped group aborts naming it. It used to fall back to the `"pasture"` proxy silently, so a typo or a new FAOSTAT item was given a grazing distribution with no trace in the output.

The catch-all is explicit, not implicit. `inst/extdata/livestock_mapping.csv` maps FAOSTAT items 1140 and 1150 (rabbits and hares, other rodents) and 1171 (live animals nes) onto the group `"other"` with the `cropland` and `mixed` proxies, so a catch-all group is reached by an explicit item mapping upstream and never by name-matching here. A group carrying several proxies keeps the first, as before.

Which area code the output is keyed on

The chain allocates *from* a national table keyed on `area_code` and *into* a `country_grid` keyed the same way, so both sides speak the raw reporting vocabulary the grid was rasterized in. WHEP's polity-keyed national tables are aggregated on `polity_area_code` instead, a bucket that a reporting code need not equal: `276` Sudan and `277` South Sudan both fall in bucket `206`. Every such output row therefore carries two territorial keys that disagree, and whether a consumer joins on `area_code` or on `polity_area_code` decides whether Sudan exists in its result (whep#582).

`area_key` selects which of the two the output carries. It is not a fallback: `"grid"` is the default, reproduces today's codes bit-for-bit, and warns naming the codes that cannot join; `"polity_area"` resolves each code to its bucket through polity_area_crosswalk before the polity columns are attached, so `area_code` and `polity_area_code` agree in every row. It respects `options(whep.unfold_rest_of_world)` (see `folded_reporting_areas()`), so the output and the national tables agree about where a Rest-of-World member's rows belong.

Under `"polity_area"` the raw reporting code is **carried, not replaced**: the output gains `grid_area_code`, joined with `+` where two reporting areas of one bucket meet in a cell and their rows collapse. So the fold stays recoverable at the join rather than baked into the output, the shape `build_cell_polity()` adopted for the same reason (whep#579).

Examples

```
# Minimal example with toy data
livestock_data <- tibble::tribble(
  ~year, ~area_code, ~species_group, ~heads,
  2000L,         1L,       "cattle",   5000
)
gridded_pasture <- tibble::tribble(
  ~lon,  ~lat,  ~year, ~pasture_ha, ~rangeland_ha,
   0.25, 50.25, 2000L,         600,           200,
   0.75, 50.25, 2000L,         400,           100
)
gridded_cropland <- tibble::tribble(
  ~lon,  ~lat,  ~year, ~cropland_ha,
   0.25, 50.25, 2000L,          800,
   0.75, 50.25, 2000L,          500
)
country_grid <- tibble::tribble(
```

```
  ~lon,  ~lat, ~area_code, ~cell_area_frac,
   0.25, 50.25,         1L,               1,
   0.75, 50.25,         1L,               1
)
build_gridded_livestock(
  livestock_data, gridded_pasture, gridded_cropland, country_grid
)
```

`build_gridded_livestock_emissions`
Build per-cell livestock greenhouse-gas emissions.

Description

Run the IPCC 2019 livestock emission model on a gridded herd, one 0.5-degree cell at a time, and return enteric CH4, manure CH4 and manure N2O in kilotonnes per cell, year and species.

Unlike a spatial disaggregation of a national emission total, this resolves the drivers that actually vary within a country: the manure-management climate zone and the ambient temperature come from the cell (`build_cell_climate_zone()`), and the diet quality comes from the cell's own feed mix when one is available. A national total spread over cells cannot show any of that, because every cell of a country then carries the same implied climate and diet.

Both grains are reported. The gridded sum is the primary output. The same model is also run once per country on the head-weighted national mean temperature and the national diet, and each cell additionally carries that national estimate rescaled onto it (`*_national_kt`) plus the per-country ratio between the two (`divergence_*`). The global difference between the two grains is small while the per-country difference is not, so the per-country ratio is emitted per row rather than summarised away.

Usage

```
build_gridded_livestock_emissions(
  gridded_livestock = NULL,
  method_diet = c("per_cell_feed", "national_feed", "uniform_medium"),
  method_species = c("national_head_share", "refuse"),
  method_climate_gap = c("nearest_cell", "drop", "refuse"),
  tier = 2,
  options = list(),
  data = list(),
  example = FALSE
)
```

Arguments

`gridded_livestock`
: A tibble of gridded head counts (required unless `example = TRUE`), for example from `build_gridded_livestock()`, with columns `lon`, `lat`, `year`,

`area_code`, `heads`, and either `species` (an IPCC species label such as `"Cattle, dairy"`) or `species_group` (a spatializer group label). Groups that name more than one IPCC species (`"sheep_goats"`, `"equines"`, `"poultry"`, `"other"`) are split as `method_species` says. Polity columns and cell identifiers are preserved when present.

`method_diet` How each row's `diet_quality` is resolved, in decreasing rigour:

- `"per_cell_feed"` (default): from the cell's own feed mix, falling back per row to that country's national mix where a cell has no classifiable feed. Needs `data$feed_intake` at cell grain (a `sub_territory` column), for example from `build_feed_intake_local()`.
- `"national_feed"`: from the country's feed mix. Needs `data$feed_intake`, for example from `get_feed_intake()`.
- `"uniform_medium"`: every row gets the IPCC `"Medium"` diet. This is an assumption, not a measurement, and is never selected implicitly.

Whatever is requested, the value actually used is recorded per row in `method_diet`. A row that no requested method resolves aborts.

`method_species`

How a `species_group` that names more than one IPCC species is split into its species:

- `"national_head_share"` (default): each cell's group head count is divided among the group's members in the proportions the country itself reports, from the national head counts by live-animal item (`data$species_heads`). The members are read from the same `livestock_mapping.csv` the spatializer summed them with, so the split inverts that grouping at national level: a country's gridded species mix equals its reported one, and the group's gridded head total is conserved exactly. It assumes the species mix is the same in every cell of a country, which is also what the spatializer assumed, because it gave all members of a group one spatial proxy. Countries are matched on `area_code`; a cell whose `area_code` the head table does not carry is matched on `polity_area_code` instead (for example Sudan and South Sudan, reported together as polity 206), and `method_species` says which. A cell matched by neither is returned with `NA` emissions and a warning, never split on an invented ratio.
- `"refuse"`: abort on any aggregate group, the behaviour before whep#1126.

The rung used is recorded per row in `method_species`.

`method_climate_gap`

What happens to a cell that has no row in the climate table, which is where the climate land mask disagrees with the livestock grid (coastlines, reclaimed land, small islands):

- `"nearest_cell"` (default): the cell takes the mean annual temperature of its nearest climate cell in the same year (great-circle distance between cell centres; equidistant cells are averaged), and its zone is classified from that. `method_climate_zone` reads `"nearest_cell; <the donor's method>"`. Only cells up to three grid steps away are searched; a gap with none that close aborts, because it means the climate table does not cover the herd.

- `"drop"`: such rows are removed, with a warning naming their head count. Their emissions are then absent from every total.
- `"refuse"`: abort, the behaviour before whep#1126.

No option gives a gap cell a default zone.

`tier` IPCC tier, `2` (default) or `1`. Tier 2 is the default here because the per-cell drivers only enter the Tier 2 energy and manure-management equations; Tier 1 emission factors carry no climate, temperature or diet dimension, so a Tier 1 grid differs from a disaggregated national total only by rounding.

`options` A named list of manure-engine options. All but nine defaults reproduce the behaviour in force before whep#949. The exceptions are `enteric_ef_source`, `ef3_source` and `slurry_crust`, which since whep#601 read the 2019 Refinement's Tier 1 enteric factors and direct manure N2O EF3 and move Tier 1 enteric CH4 and both tiers' direct manure N2O, `mcf_source`, which moved from the shipped table to the 2019 Refinement in whep#1022 and does move Tier 2 manure CH4, `mms_shares`, which moved from the unsourced placeholder table to the GLEAM 2.0 ingest in whep#958 and does move both tiers' manure N2O, `pasture_bo`, which since whep#1137 pairs the 2019 pasture MCF with its published `Bo` and moves Tier 2 manure CH4, and `tier2_uncovered`, which since whep#1028 gives species with no Tier 2 method their Tier 1 values instead of `NA`, `indirect_n2o_source`, which since whep#1245 reads the 2019 Refinement's leaching factors and moves both tiers' indirect manure N2O, and `indirect_n2o_fractions`, which since whep#1365 weights the Table 10.22 fractions over the manure-management split and moves it again.

`indirect_n2o_source` selects the edition of indirect_n2o_ef the indirect manure N2O reads:

- `"ipcc_2019"` (default): EF5 0.011 and FracLEACH-(H) 0.24, from Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26 of the 2019 Refinement, the current IPCC guidance and the EF5 the nitrogen balance already uses.
- `"ipcc_2006"`: EF5 0.0075 and FracLEACH-(H) 0.30, from Table 11.3, p. 11.24 of the 2006 Guidelines. These are the values WHEP shipped before whep#1245 under a 2019 citation, kept selectable so earlier figures stay reproducible.

EF4 (0.010) is the same under both. The edition also selects the Ch 11 FracGASM (0.21 in 2019, 0.20 in 2006) the pasture stream is priced at.

`indirect_n2o_fractions` selects which nitrogen fractions the indirect manure N2O volatilises and leaches:

- `"per_mms"` (default): IPCC 2019 Refinement, Vol 4, Ch 10, Eq. 10.26 and 10.27 (Updated), pp. 10.76-10.77. `FracGasMS` and `FracLeachMS` per animal category and system, from Table 10.22 (Updated), pp. 10.96-10.97, weighted over the row's own manure-management split, the same split the direct EF3 is weighted over. Pasture/range/paddock N, which Table 10.22 does not cover, is priced at the Ch 11 Table 11.3 FracGASM and FracLEACH-(H) (Eq. 11.9 and 11.10), as the IPCC prices grazing deposition; no other WHEP chain prices it, and the

direct path prices the same N at EF3, so it is counted once. The managed-system fractions are the 2019 table's under either edition, as in `apply_management_losses()`: the 2006 Table 10.22 leaves most pairs blank.

- `"single"`: one FracGasMS 0.20 (assumed, unverified) and one FracLEACH-(H) on all excreted N, the form before whep#1365. Kept so earlier figures stay reproducible.

`method_manure_n2o` records both choices, e.g. `indirect_ipcc_2019_per_mms` or `indirect_ipcc_2006_single`.

`enteric_ef_source` selects the Tier 1 enteric CH4 emission factors. Although it sits among the manure-engine options, it governs the enteric path only:

- `"ipcc_2019"` (default): ipcc_enteric_ef_tier1, the 2019 Refinement's Tables 10.10 and 10.11 (Updated) as published: cattle and buffalo by IPCC region, and sheep, goats and swine at the productivity system the Table 10.10 footnote assigns the region (high in North America, Europe and Oceania, low elsewhere). A row the edition prices at nothing keeps the as-shipped factor, stamped `enteric_ef_as_shipped_fallback`.
- `"as_shipped"`: ipcc_2019_enteric_ef_cattle and ipcc_2019_enteric_ef_other, the 2006 Guidelines' factors with four cells in neither edition, read before whep#601. Kept so earlier figures stay reproducible.

`method_enteric` records which (`enteric_ef_ipcc_2019` or `enteric_ef_as_shipped`). On FAOSTAT 2020 heads the default raises Tier 1 enteric CH4 from 110.8 to 125.3 Tg. Taking high productivity everywhere instead would give 134.2 Tg, low everywhere 124.2 Tg.

`ef3_source` selects the direct manure N2O emission factors, at both tiers: `"ipcc_2019"` (default) or `"as_shipped"`, the two editions of ipcc_manure_ef3.

`slurry_crust` sets the share of liquid slurry with a natural crust, which mixes the 2019 Table 10.21 (Updated) EF3 with a crust (0.005) and without one (0) into the engine's single `"Liquid/Slurry"` label: `"half"` (default, 0.0025, GLEAM 2.0's stated assumption for its pig and chicken slurry, extended to the other species), `"none"` (0) or `"all"` (0.005). It has no effect under `ef3_source = "as_shipped"`, whose 0.002 is a single row.

`method_manure_n2o` records the EF3 choice (`ef3_ipcc_2019; slurry_crust_half` or `ef3_as_shipped`). On FAOSTAT 2020 heads the default lowers Tier 1 direct manure N2O from 1.523 to 1.178 Tg.

`mms_shares` selects which half of regional_mms_distribution the split is read from: `"gleam_2_0"` (default) is the GLEAM 2.0 Supplement S1 Tab. 4.2-4.11 ingest, `"placeholder"` the unsourced table it replaced in whep#958. The placeholder stays selectable so the values WHEP published before that ingest remain reproducible and the sensitivity to it stays measurable; it is not a defensible alternative estimate.

`mms_region` selects how the manure-management split in regional_mms_distribution is keyed:

- `"as_available"` (default): a row uses its own region when the frame already carries a `region` column, and the `region == "Global"` split

otherwise. Tier 1 resolves a region for the (sourced) per-head N-excretion table and so takes the region-specific split; Tier 2 carries no region and so takes the Global one.

- `"resolve"`: the IPCC region is resolved from `iso3`, `area_code` or `polity_area_code` where it is missing, which makes the table's region-specific rows live on the Tier 2 path too. Opt-in because it changes which rows of the table apply, not because the rows are doubtful: since whep#958 they are the GLEAM 2.0 ingest.
- `"global"`: every row takes the `region == "Global"` split, whatever region column it carries.

`mcf_source` selects which methane conversion factor table the Tier 2 manure CH4 weighting reads:

- `"ipcc_2019"` (default): the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 (Updated) of the 2019 Refinement, which is the current IPCC guidance.
- `"ipcc_2006"`: the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 of the 2006 Guidelines.
- `"as_shipped"`: climate_mcf, whose live rows are predominantly the 2006 Guidelines Table 10.17 but with six cells that match no published IPCC value (whep#601, whep#1022). Kept selectable so an older run can be reproduced; it is no longer the default, because values whose provenance could not be established should not be what ships.

Neither edition publishes one number per Cool/Temperate/Warm zone for every system, so both as-published tables apply a stated collapse rule; see climate_mcf_ipcc. Both rules are WHEP's, not the IPCC's, and the default makes them live. `method_manure_ch4` records the table used.

`pasture_bo` selects the methane potential (`Bo`) the Tier 2 manure CH4 prices the pasture/range/paddock stream at. The 2019 Refinement publishes its single 0.47 percent pasture MCF as half of a pair that "must always be used in conjunction with a B0 value of 0.19" (Vol 4, Ch 10, Table 10.17 (Updated) footnote 2, p. 10.70; Section 10.4.2, p. 10.66); the pair is the `paired_bo_m3_kg_vs` column of climate_mcf_ipcc (whep#1137).

- `"paired"` (default): a stream whose MCF row carries a paired `Bo` is priced at it, every other stream at the animal-category `Bo` of ipcc_tier2_bo_values. Only `mcf_source = "ipcc_2019"` publishes a pair, so under the other two sources this changes nothing.
- `"species"`: every stream takes the animal-category `Bo`, the behaviour before whep#1137. Under `"ipcc_2019"` that is the hybrid the Refinement rejects; kept so the earlier figures stay reproducible and the sensitivity to the pairing stays measurable.

`method_manure_ch4` records per row which applied (`pasture_bo_paired` or `pasture_bo_species`) wherever the row has manure on a stream that carries a published pair.

`climate_source` selects where the climate zone the methane conversion factors in the MCF table are read at comes from. A `climate_zone` a

row already carries is always used and stamped `climate_from_data`; the option governs only the rows left without one, whether that is a hole in a supplied column or a wholly absent column.

- `"assumed"` (default): fill with `assumed_climate_zone`.
- `"from_data"`: abort instead of assuming.

`assumed_climate_zone` is the zone `"assumed"` fills in: `"Cool"`, `"Temperate"` (default) or `"Warm"`. It is an assumption, not a measurement; `method_manure_ch4` records per row which of the sources applied, and this argument exists so the sensitivity to the assumption can be measured (whep#949).

`tier2_uncovered` says what the Tier 2 path does with a species WHEP has no Tier 2 method for. The energy balance needs the maintenance and activity coefficients of Tables 10.4 and 10.5, which ipcc_tier2_energy_coefs ships for cattle, buffalo, sheep and goats only; the 2019 Refinement itself suggests Tier 1 for camels, horses, mules and asses and swine and has no enteric method for poultry (Vol. 4 Ch. 10, Table 10.9 (Updated)), and its Tier 2 manure equations for swine and poultry need a country-specific dry-matter intake (Equation 10.32A) that WHEP does not hold. Although it sits among the manure-engine options, it governs the enteric path too.

- `"tier1"` (default): those species take the Tier 1 enteric CH4, manure CH4 and manure N2O, written into the Tier 2 output columns and stamped `"IPCC_2019_Tier1"` in `method_enteric`, `method_manure_ch4` and `method_manure_n2o`, with a message naming them. This is the IPCC's own suggested method for them, so a Tier 2 inventory keeps the whole herd rather than silently covering fewer animals than Tier 1.
- `"leave_na"`: they keep `NA` emissions, the behaviour before whep#1028, with a warning naming them. Kept so a ruminant-only Tier 2 figure stays reproducible.
- `"abort"`: any such species aborts, naming it.

`data` Optional named list of pre-loaded inputs: `cell_climate` (a `build_cell_climate_zone()` output), `feed_intake` (a feed-intake table) and `species_heads` (national head counts by live-animal item, with `year`, `area_code`, `item_cbs_code` and `value` or `heads`, optionally `polity_area_code` and `unit`; rows with a `unit` other than `"heads"` are ignored, except `"t_head"` rows, which give the milk yield, see section "Milk yield"). `cell_climate` falls back to `build_cell_climate_zone()`, which reads CRU from `WHEP_CRU_DIR`. `species_heads` falls back to `get_primary_production()`, the source the spatializer's country table is built from, and is read only when an aggregate group is present. `feed_intake` has no fallback: the readers that produce it rebuild the whole feed allocation, so it is supplied or the diet method is `"uniform_medium"`.

`example` If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with one row per `year`, `area_code`, `lon`, `lat` and `species`:

- `heads`: Head count in the cell.
- `enteric_ch4_kt`, `manure_ch4_kt`, `manure_n2o_kt`: Gridded emissions (kilotonnes), the primary output.
- `enteric_ch4_national_kt`, `manure_ch4_national_kt`, `manure_n2o_national_kt`: The national-grain estimate rescaled onto the cell, so summing these over a country reproduces the national-grain run.
- `divergence_enteric_ch4`, `divergence_manure_ch4`, `divergence_manure_n2o`: Per-country ratio of the gridded total to the national-grain total. `1` means the two grains agree.
- `mean_annual_temp_c`, `climate_zone`, `diet_quality`: The resolved per-cell drivers.
- `species_group`: The spatializer group, when the input carried one.
- `method_species`: How the row's `species` was resolved: `"supplied"`, `"one_to_one"` (the group is a single species), `"national_head_share"`, `"polity_bucket_head_share"` or `"unsplit_no_national_mix"` (emissions `NA`, see `method_species`).
- `method_climate_zone`: How the cell's climate zone was resolved, from `build_cell_climate_zone()`, prefixed `"nearest_cell; "` when it was taken from the nearest climate cell (see `method_climate_gap`).
- `method_diet`, `method_enteric`, `method_manure_ch4`, `method_manure_n2o`: Method tracking.

plus the polity columns below.

Milk yield

Tier 2 lactation energy (IPCC 2019 Eq 10.8) needs a milk yield per dairy cow. It is taken, in this order, from a `milk_yield_kg_day` column on `gridded_livestock`, or from the realised national yield of the `t_head` rows of `data$species_heads` (milk tonnes per head and year, converted to kilograms per day as `prepare_livestock_emissions()` does). A dairy row that neither supplies aborts: the yield is never filled with zero, which would book every dairy cow as giving no milk.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_gridded_livestock_emissions(example = TRUE)
```

---

build_hayr_land_extension

Build a hectare-year (land-occupation) crop land extension.

Description

Per-crop land *occupation* in hectare-years (the LCA `m2*year` convention): the land-time each crop's production ties up, $occupation_i = harvested_i \times L_i/12 + fallow_i$.

The first term is **active growing occupation** — harvested area times mean cycle length L_i (months, from MIRCA2000). Because it uses harvested area, a field double-cropped twice contributes both cycles, and a long-cycle perennial contributes close to a full year. The second term is the **rotational fallow** attributed to the crop, which occupies land the whole year while it rests.

This is "active" occupation: land is charged only while a crop is growing on it or resting in its rotation, so the national total falls below physical cropland area (which also counts off-season idle). It is **distinct from, and complementary to**, the physical-area extensions (`build_cropgrids_land_extension()`): those measure the field area each crop holds; this measures how much land-*time* each crop's activity occupies. Short single-cropped crops and intensively double-cropped staples occupy less land-time per hectare than long-cycle and perennial crops.

Usage

```
build_hayr_land_extension(
  harvested = NULL,
  fallow = NULL,
  season = NULL,
  base = c("cropgrids_fallow", "cropgrids")
)
```

Arguments

`harvested`	Tibble of harvested area with columns `year`, `area_code`, `item_cbs_code`, `harvested_ha`. If `NULL`, built from `get_primary_production()` (and reused to build the fallow term).
`fallow`	Tibble of attributed rotational fallow with columns `year`, `area_code`, `item_cbs_code`, `fallow_ha`. If `NULL`: for `base = "cropgrids_fallow"` it is the difference between the fallow-inclusive and cropped CROPGRIDS physical extensions; for `base = "cropgrids"` it is zero (growing occupation only).
`season`	Tibble of mean crop cycle length with columns `item_cbs_code`, `season_months` (strictly positive, unique keys). If `NULL`, the packaged MIRCA2000 season table is used. Crops with no season are given the median cycle length.
`base`	`"cropgrids_fallow"` (default, include rotational fallow) or `"cropgrids"` (growing occupation only). Recorded in `method_land` as `<base>_hayr`.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (land occupation in hectare-years), and `method_land`.

Examples

```
harvested <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~harvested_ha,
  2000L, 1L, 2807L, 200, # rice, double-cropped (two harvests per field)
  2000L, 1L, 2511L, 100 # wheat, single-cropped
)
season <- tibble::tribble(
  ~item_cbs_code, ~season_months,
  2807L, 5,
  2511L, 8
)
fallow <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~fallow_ha,
  2000L, 1L, 2511L, 20
)
build_hayr_land_extension(harvested, fallow, season)
```

`build_historical_land_areas`

Build a pre-1962 land series measured on each year's own borders

Description

Emit the land table the pre-1962 production back-cast consumes – `year`, `area_code`, `Cropland`, `Pasture` and `agriland`, all in Mha – with the hectares summed from gridded LUH2 inside the polygon of the polity that `area_code` resolved to in that year, instead of inside present-day borders.

The cell-by-polity intersection is not measured here: it is read from the polycell support table (`read_polycell_support()`), which is that intersection measured geodesically on s2 and keyed on each polity's validity interval.

A cell's land is shared among the polities whose territory covers it, in proportion to that territory renormalised to one per cell, which is the rule `build_cell_polity_fraction()` already uses. Renormalising matters: LUH2's state fractions are fractions of the whole cell and already discount open water, so weighting them by a raw coastal cell's land share would discount it twice and lose 12-15% of the land of an island or heavily coastal country.

`fill_proxy_growth()` consumes only this series' year-on-year ratios, so a change of territory can only reach the back-cast as a ratio. What that ratio should be is a real choice, and `boundary_step` makes it:

- `"relink"` re-measures the previous year inside the **incoming** polity's polygon before taking the ratio, so only within-territory growth is ever used and annexing a province never moves the back-cast. On Ethiopia in 1952, when Eritrea joins, that is +1.9% instead of +8.0%.
- `"level_step"` (default) takes the ratio between the two polygons as measured, so the territorial change passes through as a level step and the 1850 row is scaled to the smaller empire it is labelled with. That is the reframing the whole method exists for; it is also the option most exposed to a bad polygon, because an artefact of the polity database then compounds down the back-cast exactly as a real annexation would.

Measured over 1850-1961 against the present-day series, 18.0% of back-cast crop tonnage at 1850 sits between the two rules, falling to 0.07% by 1960.

This reads gridded LUH2 for every requested year and is minutes-to-tens-of- minutes of work, so it belongs in a `data-raw/` materialisation step, not in a test or an example.

Usage

```
build_historical_land_areas(
  years = 1850:1961,
  boundary_step = c("level_step", "relink"),
  data = NULL,
  example = FALSE
)
```

Arguments

`years`	Integer vector of calendar years to measure. Defaults to `1850:1961`, the span the back-cast uses.
`boundary_step`	How a year-on-year ratio is taken across a change of territory: `"level_step"` (default) or `"relink"`. They answer different questions and differ by up to 18% of back-cast tonnage, so the choice is the method, not a tuning knob – see the description. `"level_step"` lets the series step when the territory changes, because a different polity is a different thing being measured. That is what a per-polity series means, and it is why this function exists: on Ethiopia it puts 1850 cropland at 1.52 Mha against the present-day 3.22, dropping the land Menelik annexed in the 1880s-90s that the area never held in 1850. `"relink"` re-measures the previous year inside the incoming polity's polygon so a change of territory never appears as growth. That suits a FIXED-territory series, where the step is an artefact. It is NOT the conservative choice here: because `fill_proxy_growth()` consumes only ratios, suppressing that channel also suppresses the correction, and Ethiopia's 1850 comes back to 3.24 Mha – within 0.6% of the present-day figure this method exists to replace (whep#761).
`data`	Named list of pre-loaded inputs bypassing the readers, for tests: `polity_areas` (`year`, `area_code`, `polity_code`), `support` (a `read_polycell_support()` table), `cover` (`polity_code`, `lon`, `lat`, `frac`, which is `support` already reduced to one weight per cell) and `cell_areas` (`year`, `lon`, `lat`, `land_use`, `area_ha`). Each falls back to its reader when absent.
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `year`, `area_code`, `polity_code`, `Cropland`, `Pasture` and `agriland`. `area_code` is the `polity_area_code` aggregation bucket, the same key `.read_land_areas()` emits, so the result is a drop-in for it at the back-cast seam. `polity_code` names the territory each year was measured on, and is semicolon-separated where a bucket holds more than one polity in a year.

Examples

```
build_historical_land_areas(example = TRUE)
```

---

build_human_n *Build gridded human-population nitrogen inputs to agriculture.*

Description

Estimates the nitrogen the human population returns to agricultural land through municipal solid waste, sewage sludge and human excreta, per WHEP 0.5-degree grid cell. Each polycell's population is converted to a nitrogen load via a per-capita rate interpolated from a national historical benchmark series, the Spanish series taken as reference (human_n_reference; see Details), then spilled from cells with no local cropland room to same-polity neighbouring cells with spare capacity via `allocate_manure_transport()`, the same buffering used by the manure engine.

The population and the rate are chosen together by `population_basis`, so a per-urban-inhabitant rate is never applied to a total population:

- `"total"` (default): UN WPP total population downscaled by HYDE's total-population pattern (`build_total_population_grid()`) times human_kgn_cap_total_reference, kg N per inhabitant.
- `"urban"`: HYDE's urban population count (`read_hyde_population(variable = "urban")`) times human_kgn_cap_reference, kg N per urban inhabitant.

Both rates are the calibration nitrogen divided by the calibration population on the same basis, so either regenerates its calibration total. Elsewhere they differ by how far a population's urban share departs from the calibration one: in 2010 the global WPP total population is 2.0 times HYDE's global urban count.

`build_urban_n()` is the deprecated former name of this function. It forwards every argument to `build_human_n()` and warns (class `whep_build_urban_n_deprecated`, also `lifecycle_warning_deprecated`); it will be removed in a future release. The output columns were renamed with it: `urban_n_t` is now `human_n_t`, and `method_urban`, `method_urban_population` and `method_urban_kgn_cap` are now `method_human`, `method_human_population` and `method_human_kgn_cap`.

Usage

```
build_human_n(
  years = NULL,
  population_basis = c("total", "urban"),
  polity_validity = c("keep", "flag", "drop"),
  data = list(),
  example = FALSE,
  method_residual = c("nearest", "polity", "keep", "drop"),
  method_local_residual = c("room_cap", "uncapped")
)

build_urban_n(...)
```

Arguments

`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year the supplied population covers; it is required when the population is read rather than supplied.
`population_basis`	Which population, and the per-capita rate on the same basis, generates the load: `"total"` (default) or `"urban"`. See Description. Recorded

in the `method_human_population` and `method_human_kgn_cap` output columns.

`polity_validity`
: What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`data`
: Optional named list of pre-loaded inputs: `total_population` (`lon`, `lat`, `area_code`, `year`, `population`, one row per polycell as `build_total_population_grid()` returns it; read under `population_basis = "total"`, falling back to that builder when absent; taken per polycell, not re-split by `polity_frac`, and every polycell must exist in `cell_polity`), `urban_population` (`lon`, `lat`, `year`, `urban_pop`; read under `population_basis = "urban"`, falling back to `read_hyde_population(variable = "urban")` when absent), `cell_polity` (`lon`, `lat`, `area_code`, plus optional `polity_frac`; a missing `polity_frac` is treated as 1 for backwards compatibility) and `cropland_ha` (`lon`, `lat`, `area_code`, `year`, `cropland_ha`, required: the gridded cropland area used as the simple room proxy, `cropland_ha * 0.170` t N/ha, the same EU-Nitrates fixed ceiling used by `allocate_manure_to_land()`'s `fixed_ceiling_kg_ha` default). Supplying only the other basis's population aborts with class `whep_human_n_population_basis_mismatch` rather than silently reading a default. Both frames' `area_code` must be the numeric WHEP area code, whole-numbered, as `build_cell_polity()` emits it. Anything else – an ISO3 literal, an area name, a fractional value – aborts with class `whep_human_n_area_code_unresolved` (also `whep_urban_area_code_unresolved`, its former name), naming the frame that carries it. It is not bridged: the two frames key the same transport partition, so one written in a different vocabulary from the other would silently strand a cell's load on a cell with no room instead of placing it, and an ISO3 resolves to a `polity_area_code` aggregation bucket that is not every territory's own code (`"SSD"` would become 206, Sudan (former)). Map to the code first, via `add_area_code()` or regions_full.

`example`
: If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

`method_residual`
: What happens to human N that the transport step cannot deliver: the residual on a source cell with **no cropland**, plus, under the default `method_local_residual = "room_cap"`, the part of a cropland source cell's residual that exceeds its own room. On the 2010 global grid, under the default `"total"` basis, the first is 13,198 cells and 47,365 t N, 0.700% of the 6.77 Mt of human N, and the second adds 24,346 t on 1,424 cells; see `method_local_residual` for what `"nearest"` leaves stranded.

  - `"nearest"` (default): run the transport step's own rule again with a growing radius. Each such cell offers its nitrogen to the same-polity

cropland cells in its nearest ring (Chebyshev distance in 0.5-degree steps, wrapped at the antimeridian) that still has room, in proportion to that room; a cell's room is its 170 kg N/ha ceiling minus the human N already on it, and an over-subscribed cell is filled only to its room. The radius grows one ring at a time until the nitrogen is placed or the polity has no room left. Conserves mass and keeps the nitrogen as close to the people who produced it as the room allows. No distance cap or transport coefficient is applied.

- `"polity"`: pool it per polity-year and spread it over all of that polity-year's cropland in proportion to cropland room (area), the rule `build_n_inputs()` applies under `method_unsupported = "reallocate"`. Conserves mass, but places the nitrogen anywhere in the polity.
- `"keep"`: leave it on its source cell, as before this argument existed, flagged in `human_n_stranded_t`. On a cell with no cropland, `build_n_inputs()`'s `method_unsupported` then decides its fate (by default, it aborts); an over-room excess is applied on its own cell's cropland, as `"uncapped"` would.
- `"drop"`: discard it. Loses the mass, biased towards dense, cropland-free cells.

`"nearest"` and `"polity"` never cross a polity, like the transport step itself, so a polity-year with population and no cropland anywhere keeps its nitrogen on the source cell under either, flagged in `human_n_stranded_t`; `"nearest"` does the same with whatever its polity has no room left for. Whenever any cell is undelivered, the count, the tonnes and the share of human N are reported (a warning, class `whep_human_n_undelivered`, when any nitrogen is dropped or left stranded; otherwise a message of the same class), and the per-year figures are attached as `attr(x, "human_n_undelivered")`. Recorded in the `method_human_residual` output column.

`method_local_residual`

What happens to the residual the transport step hands back to a source cell that **has** cropland. The transport step offers a cell's load to its ring-1 neighbours only, so a dense cell with little cropland gets back most of its own load, on that little cropland.

- `"room_cap"` (default): the cell keeps only what fits in its own room, 170 kg N/ha times its cropland minus the human N the transport step already landed on it – the room the transport step and `"nearest"` respect everywhere else. The excess is undelivered N, placed by `method_residual` like the residual of a cell with no cropland. On the 2010 global grid (`"total"` basis) that is 24,346 t on 1,424 of the 2,180 cropland source cells left with a residual. Under `"nearest"`, 17,083 t of it is moved and 7,263 t stays stranded on its own cell, in the three polities with no room left anywhere (Hong Kong 6,701 t, Kuwait 363 t, the Bahamas 200 t), alongside the 2,145 t on cells with no cropland (Qatar, Iceland, Samoa).
- `"uncapped"`: the cell keeps its whole residual, as before this argument existed, whatever its cropland area. On the same grid 1,417 cropland cells then end above 170 kg N/ha, holding 24,346 t above it, and the largest load is booked on 3.6e-7 ha.

No minimum-cropland threshold is offered: the room cap already moves a sliver's whole residual, and a threshold would be a new, unsourced number that misses the excess on larger cells (15,202 t of the 24,346 t sits on cells with at least 1 ha). Recorded in the `method_human_local_residual` output column; the part of `undelivered_t` it adds is the summary's `over_room_t`.

`...` For `build_urban_n()`, arguments passed on to `build_human_n()`. `population_basis` defaults to `"urban"` here if omitted, matching this function's historical behaviour, unlike `build_human_n()`'s own `"total"` default.

Details

The current per-capita rate is a documented placeholder (one national historical series applied as a global default). For a future refinement, human N should instead be derived from two distinct, more mechanistic streams: (1) sewage/human-excreta N estimated from actual historical per-capita dietary protein/N intake (already reconstructable in WHEP via its FAOSTAT/commodity-balance food-supply data, rather than a fixed external per-capita constant), and (2) food-waste/municipal-solid-waste N from actual historical food-loss and waste estimates. This is out of scope for the current task and is not implemented here.

Value

A tibble with `lon`, `lat`, `area_code`, `year`, `human_n_t`, `human_n_relocated_t` (the part of `human_n_t` placed on the cell by `method_residual`), `human_n_stranded_t` (the undelivered part no rule could place: on a cell with no cropland, which no downstream cropland allocation can place, or above the room of the cell's own cropland, which `build_n_inputs()` then applies there, over the ceiling), `method_human`, `method_human_population` (`"total_population"` or `"urban_population"`), `method_human_kgn_cap` (`"kg_n_per_total_inhabitant"` or `"kg_n_per_urban_inhabi` and `method_human_residual` and `method_human_local_residual`, plus the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`. The attribute `"human_n_undelivered"` is a tibble with one row per year: `year`, `human_n_t`, `n_cells` (undelivered source cells), `undelivered_t`, `over_room_t` (the part of `undelivered_t` that exceeded the room of its own cell's cropland), `relocated_t`, `stranded_t`, `dropped_t`, `undelivered_share` (of `human_n_t`), `method_human_residual` and `method_human_local_residual`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.

- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_human_n(example = TRUE)
```

`build_intake_dispersion` *Build the within-country intake dispersion.*

Description

Returns the log-scale standard deviation of habitual intake per country and year, the input a lognormal shortfall model needs. The coefficient of variation comes from FAOSTAT Suite of Food Security Indicators item 21058 and is converted with `sigma = sqrt(log(1 + cv^2))`, which is the conversion FAO performs itself; treating a raw CV as a log-SD errs by about 1.5% at a CV of 0.25.

`nutrient = "protein"` (default) scales the energy-based dispersion by 1.062, the mean protein-to-energy log-SD ratio measured across eight countries with deconvolved usual-intake surveys. `"energy"` leaves it unscaled and is about 6% low for a protein floor.

`estimand = "faostat"` (default) takes item 21058 on its own scale and carries its understatement as a documented signed bias. `"calibrated"` multiplies by 1.477, the median ratio of survey usual-intake dispersion to this series across eight countries; it improves the level and rests on a calibration whose countries all sit in the lower half of the CV range.

Item 21058 begins in 2001. `temporal = "hold_constant"` (default) carries each area's earliest published value back over earlier years, which is FAO's own documented rule for

unsupported country-years, and stamps those rows accordingly. A covariate hindcast was specified and tested against this and did not beat it: it wins RMSE by 2.7% and loses MAE over 1,332 country-years.

Usage

```
build_intake_dispersion(
  data = list(),
  years = NULL,
  nutrient = c("protein", "energy"),
  estimand = c("faostat", "calibrated"),
  temporal = c("hold_constant", "observed_only")
)
```

Arguments

`data`	Named list of injected inputs. `habitual_cv` (`year`, `area_code`, `cv`) overrides the FAOSTAT read.
`years`	Optional integer vector of years to return. Years before the series begins are filled per `temporal`.
`nutrient`	`"protein"` (default) or `"energy"`.
`estimand`	`"faostat"` (default) or `"calibrated"`.
`temporal`	`"hold_constant"` (default) or `"observed_only"`, which returns nothing before the series begins rather than carrying it back.

Value

A tibble keyed by `year`, `area_code` with `cv`, `sigma`, `method_dispersion` and `method_cv_year`, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_intake_dispersion(
  data = list(
    habitual_cv = tibble::tribble(
      ~year, ~area_code, ~cv,
      2001L, 10L,        0.25,
      2002L, 10L,        0.24
    )
  ),
  years = 2000:2002
)
```

---

build_io_model *Build multi-regional input-output model.*

Description

Construct a multi-regional input-output (MRIO) model from supply-use tables, bilateral trade, and commodity balance sheets. Uses the industry technology assumption to derive symmetric product-by-product tables.

The resulting matrices follow the FABIO methodology (Bruckner et al., 2019). Rows and columns of `Z` represent (country, item) pairs. Each entry `Z[i,j]` gives the intermediate flow from sector `i` to sector `j`.

Usage

```
build_io_model(
  supply_use = NULL,
```

```
  bilateral_trade = NULL,
  cbs = NULL,
  years = NULL,
  endogenize_losses = FALSE,
  method = c("mass", "value"),
  prices = NULL,
  trade_recovery = .cbs_trade_recovery_choices()
)
```

Arguments

supply_use	Tibble from build_supply_use(). By default, this function calls build_supply_use() internally. Must have columns: year, area_code, proc_group, proc_cbs_code, item_cbs_code, type, value.
bilateral_trade	Tibble from get_bilateral_trade(). By default, this function calls get_bilateral_trade() internally. Must have columns: year, item_cbs_code, bilateral_trade (list-column of matrices).
cbs	Tibble from get_wide_cbs(). By default, this function calls get_wide_cbs() internally. Must have columns: year, area_code, item_cbs_code, production, export, stock_withdrawal, stock_addition, plus final demand columns (food, other_uses). import is read only when bilateral_trade is NULL, by get_bilateral_trade(), which balances the trade matrices against it. year, area_code and item_cbs_code must hold no NA, here and in supply_use: a row with a missing code cannot be placed in the model.
years	Numeric vector of years to compute, or NULL. If NULL, computes all years in the intersection of available data across inputs. If specified, must be a subset of available years.
endogenize_losses	Logical. If TRUE and cbs contains a losses column, losses are moved from final demand to the diagonal of Z (self-use), following the FABIO convention. The losses column is removed from Y and fd_labels. get_wide_cbs() emits no losses column, so TRUE with such a cbs warns and builds the same model as FALSE. Defaults to FALSE.
method	Co-product allocation method. "mass" (default) splits a multi-output process's inputs across its products by physical mass; "value" splits them by economic value (mass times export price), so high-value co-products (e.g. oil over cake, meat over hides) carry a larger share of upstream pressures. A process whose co-products lack usable prices falls back to mass.
prices	Optional tibble of item prices as from build_cbs_prices() (year, element, item_cbs_code, price). Used only when method = "value"; built automatically when NULL.
trade_recovery	One of "byproduct_residual" (default), "pass_through", "net_import" or "none", selecting the CBS every internally built input is derived from — the wide CBS, the processing coefficients, the supply-use tables, the bilateral trade matrices and the prices. See build_commodity_balances()

for what each method does. It has no effect on an input supplied directly, so a `cbs` built with one method and the rest left to build under another is the caller's to avoid; pass the same value everywhere.

Value

A tibble with one row per year and list-columns:

- `Z`: Inter-industry flow matrix (product-by-product).
- `Y`: Final demand matrix.
- `X`: Total output vector.
- `labels`: Tibble mapping row/column indices to `area_code`, `item_cbs_code`, and reporting polity metadata.
- `fd_labels`: Tibble mapping each Y column to its `area_code` (consuming polity), `fd_col` (demand category, e.g. `"food"`), and reporting polity metadata. Pass to `compute_footprint()` as `fd_labels` to get a `target_fd` column in the footprint output.

Examples

```
su <- build_supply_use(example = TRUE)
btd <- get_bilateral_trade(example = TRUE)
cbs <- get_wide_cbs(example = TRUE)
build_io_model(su, btd, cbs)
```

---

build_land_balance_footprint

Build a consumption land footprint by physical trade balance.

Description

End-to-end land-balance footprint for one year on real WHEP data: assemble production (primary crop production plus grass dry-matter derived from grassland area times yield), the bilateral trade network, and the crop-plus-grassland direct-land extension, then trace land to consumers with `compute_footprint_balance()`. This is the independent, non-Leontief estimator for stress-testing the multi-regional input-output footprint via `compare_footprint_methods()`.

The crop side is `build_fao_arable_fallow_extension()` with `unsupported_target = "unallocated"`, pinned explicitly: FAO land that no crop can be named for arrives on an `item_cbs_code` of `NA`, which the balance cannot route, so it is reported by the orphan-land warning rather than folded into a crop (whep#1026).

Its temporary-grassland netting basis is pinned explicitly too, to `temp_grassland_basis = "modelled"`: modelled CBS 3002 covers 26 EU polities over 2001-2019 only, so for any other country, and for every country from 2020, nothing is netted and the arable land keeps FAO's temporary meadows (whep#937). The balance output does not carry that provenance per row; read `temp_grassland_source` in `build_fao_arable_fallow_extension()`'s output, or `check_arable_composition()`, to see which country-years were netted.

Grass items (`item_cbs_code` 3000 and 3002) are barely traded, so their land stays with the producing country: the balance, unlike the input-output model, does not route grass through the grass-to-livestock chain. Any `production`, `trade` or `extension` supplied explicitly is used as-is instead of being built, which is useful for reusing cached inputs or for testing.

Usage

```
build_land_balance_footprint(
  year,
  production = NULL,
  trade = NULL,
  extension = NULL,
  example = FALSE
)
```

Arguments

`year`	Year to build the footprint for.
`production`	Optional production tibble (`area_code`, `item_cbs_code`, `value`); built when `NULL`.
`trade`	Optional trade tibble (`from_code`, `to_code`, `item_cbs_code`, `value`); built when `NULL`.
`extension`	Optional direct-land tibble (`area_code`, `item_cbs_code`, `value`); built when `NULL`.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble as returned by `compute_footprint_balance()`.

Examples

```
build_land_balance_footprint(example = TRUE)
```

`build_level_crop_targets` *Split a national crop total across the units of its container*

Description

The first half of `allocate_level_crops()`, exposed on its own because the reconciliation diagnostics read it: it turns national totals plus resolved admin shares into one target per `(year, container, unit, item)`, and says on every row which regime produced it.

Usage

```
build_level_crop_targets(
  country_areas,
  unit_weights,
  admin_shares = NULL,
  tolerance_relative = 0.1,
  tolerance_absolute = 1000
)
```

Arguments

`country_areas`	National crop areas keyed on the container; see `allocate_level_crops()`.
`unit_weights`	Per-unit pattern weights, from `.alloc_unit_weights()`: `year`, `area_code`, `level_polity_code`, `item_prod_code`, `weight_rainfed`, `weight_irrigated`, `cropland_rainfed_ha`, `cropland_irrigated_ha`, `n_cells`.
`admin_shares`	Resolved admin shares, or `NULL`.
`tolerance_relative`	Relative discrepancy above which a complete-coverage group is refused, `0.10` by decision T31(d).
`tolerance_absolute`	Absolute discrepancy, in hectares, above which the same group is refused, `1000` by decision T31(d). BOTH must be breached.

Value

A list of two tibbles, `targets` and `coverage`; see `allocate_level_crops()` for their columns.

How the denominator is chosen

Per `(year, area_code, item_prod_code)`, writing `D` for the denominator each unit's reported value is divided by:

`"pattern"` No unit reports. Every unit's share is its own pattern weight over the group's, `method_crop_alloc = "pattern_implied"` – or `"pattern_national"` where the container is at level 0 and is therefore its own single unit.

`"admin_sum"` Every unit of the layer reports a value: complete coverage, so `D = admin_sum` and the reported units rescale proportionally onto the national total. The leftover is a diagnostic; it is refused only when it breaks BOTH tolerances (decision T31(d)).

`"residual"` Some units report a value: `D = max(admin_sum, national)`, so a reported value binds in absolute hectares while `admin_sum` is below the national total, and rescales down when it is above. The residual, `1 - admin_sum / D` of the total, is split across the units that reported NO absolute area, and never touches a unit that did (decision T31(a)). Those units' pattern weights split it – unless every one of them declares a share, in which case the declared shares do, because that is what a declared share states.

`"share_normalised"` The rows carry a `share` and no `value`, which is what `backcast_admin_shares()` produces: the shares already sum to 1 over the units they cover, so they are used as they stand and a unit outside them gets zero. `method_crop_alloc = "admin_backcast_luh2"`.

`method_crop_alloc` therefore takes one of `"pattern_national"` (a container the layer holds at level 0, which is its own single unit), `"pattern_implied"`, `"admin_area_shares"`, `"admin_backcast_luh2"`, `"admin_residual"` (a unit carrying its share of the residual on the weights that split it – whether because it reported nothing, or because it declared a share the split did not run on) and `"unallocated"` (a unit whose target is 0: the shares do not cover it and no residual reaches it, or the group had neither pattern weight nor cropland for anything to be split on, `coverage$weight_basis == "none"`). The column names what placed the hectares, so a unit whose declared share was displaced by the pattern weights is `"admin_residual"` and not `"admin_area_shares"`, and a group the pattern never ran on is `"unallocated"` and not `"pattern_implied"`.

A group whose reported values are all zero while the national total is positive cannot set a shape. Under complete coverage every share is 0, the hectares are counted in `coverage$dropped_ha`, and the T31(d) refusal fires on any group where that matters (the discrepancy is then 100% of the national total). Where that leaves NO positive target at all, `allocate_level_crops()` returns the empty allocation beside this coverage rather than calling an engine with nothing to place.

An admin-share row carrying no `level_polity_code` names no unit – that is what `resolve_admin_units()` leaves on a unit it could not resolve – and is dropped, counted and named before the split. `NA` is the layer's own value for "this container is at level 0 and IS its own unit", so such a row would otherwise bind the container's own row and publish one unresolved province as the country's complete subnational evidence.

What counts as a report

One predicate. A unit reports when it carries a `value`, a `share`, or both, and `coverage$n_units_reporting` counts exactly the units that then bind. `reports_value` – an ABSOLUTE area – is a narrower question, and it decides two things only: which denominator the group uses, and which units the residual is spread over. So a unit declaring a share and no value is a reporter, and it takes a slice of the residual the valued units left. What sizes that slice is a property of the candidate SET, not of the unit: where every residual candidate declares a share, those shares split the residual (`coverage$weight_basis` is then `"declared"`) and the declared share binds; where only some do, the pattern weights split it, because a declared fraction and a hectare of potential are not the same quantity. In that second case the unit is allocated exactly as a silent one is, and `method_crop_alloc` says so.

A negative reported value, a negative declared share and a negative derived target are all refused (classes `whep_alloc_negative_value` and `whep_alloc_negative_target`); `allocate_level_crops()` refuses a negative national total ahead of both, as `whep_alloc_negative_national`. A negative target is dropped by the engine's positivity filter, after which the remaining units divide the whole national total between them: the container over-allocates and nothing downstream can see the row that is not there.

Irrigation

The unit's irrigated target is the national irrigated area split by the engine's own `ir_potential` aggregated per unit, so the two steps compose; where a group has no irrigated potential

the split falls to the units' irrigated cropland, and where there is none of that either the irrigated area is unplaceable and is reported as such, because the engine would drop it too. The rainfed target is the remainder, and the irrigated target is clipped at the unit's area target so the remainder is never negative – the engine computes `harvested - irrigated` with no clipping and lets a negative row through its output filter. Clipped hectares are reported per unit in `irrigation_clipped_ha` and are NOT redistributed to units with room: that would be an allocation rule nothing has decided.

Examples

```
weights <- tibble::tibble(
  year = 2000L,
  area_code = 1L,
  level_polity_code = c("A1", "A2"),
  item_prod_code = 15L,
  weight_rainfed = c(400, 600),
  weight_irrigated = 0,
  cropland_rainfed_ha = c(800, 1200),
  cropland_irrigated_ha = 0,
  n_cells = 1L
)
national <- tibble::tibble(
  year = 2000L, area_code = 1L, item_prod_code = 15L,
  harvested_area_ha = 1000
)
build_level_crop_targets(national, weights)$targets
```

`build_livestock_ghg_extension` *Build the livestock greenhouse-gas emissions extension.*

Description

Aggregate per-animal IPCC livestock emissions into a footprint extension keyed by `(year, area_code, item_cbs_` expressed in kilograms of carbon-dioxide equivalent (CO2e). This bridges the cohort-level emissions pipeline (`calculate_livestock_emissions()`) to the input-output grain used by `build_io_model()` and `compute_footprint()`, exactly like `build_grassland_land_extension()` does for land.

Live-animal head counts come from `get_primary_production()`, are bridged to IPCC species with `prepare_livestock_emissions()`, and the resulting enteric and manure emissions are converted to CO2e and summed back to the live-animal commodity sector (`item_cbs_code`, e.g. 961 for non-dairy cattle), which is itself a sector in `build_io_model()`.

Two IPCC tiers are available, selected with `tier`:

- `1` (default): Tier 1 regional emission factors (IPCC 2019). It needs only species, country and head counts, so it is complete for every country in `get_primary_production()`. It covers enteric and manure **methane** and manure **N2O** (direct and indirect, from default per-head nitrogen excretion rates).

- 2: Tier 2 cohort energy balance (IPCC 2019). It derives enteric CH4 and manure N2O from a per-animal energy and nitrogen balance, for finer resolution, but requires cohort weight and diet inputs. It covers cattle, buffalo, sheep and goats; swine, poultry, horses, mules and asses and camels, for which the IPCC suggests Tier 1, take Tier 1 by default (see `tier2_uncovered` under `options`), so their sectors show `"IPCC_2019_Tier1"` in `method_manure_ch4`. Rows that still cannot be resolved (missing diet data) are dropped with a warning rather than entering the footprint as `NA`. Its per-head enteric and manure emissions now sit in the same range as the Tier 1 regional factors. Tier 1 remains the default because it is complete for every country in `get_primary_production()`, whereas Tier 2 needs cohort and diet inputs.

The CO2e conversion uses 100-year global warming potentials selected with `gwp`:

- `"ar6"` (default): IPCC AR6 (2021) Table 7.15, biogenic CH4 = 27, N2O = 273.
- `"ar5"`: IPCC AR5 (2013), CH4 = 28, N2O = 265 (no climate-carbon feedback).
- `"ar4"`: IPCC AR4 (2007), CH4 = 25, N2O = 298.

`options` is handed to `calculate_livestock_emissions()` unchanged, so the manure engine's method levers (which half of the manure-management table is read and how it is keyed, and which methane conversion factor table is read at which climate zone) are selectable from here too; passing none takes the engine's own defaults, so this entry point publishes what the engine does. `mms_shares`, `mms_region` and `ef3_source` bite at both tiers and `enteric_ef_source` at Tier 1; the `tier = 1` default here is unaffected by `mcf_source` and by the climate options, because Tier 1 manure CH4 comes from regional emission factors rather than a climate-zone MCF and both reach the MCF on the Tier 2 path only. Whichever choice each row took is recorded in `method_mms` and `method_manure_ch4`, which the extension carries into its own output.

Usage

```
build_livestock_ghg_extension(
  tier = 1,
  gwp = c("ar6", "ar5", "ar4"),
  method_diet = c("per_cell_feed", "national_feed", "uniform_medium"),
  options = list(),
  data = list(),
  example = FALSE
)
```

Arguments

`tier`	IPCC tier, `1` (default) or `2`.
`gwp`	100-year global warming potential standard, `"ar6"` (default), `"ar5"` or `"ar4"`.
`method_diet`	How Tier 2 resolves each herd's `diet_quality`, which sets DE% and so gross energy, enteric CH4, volatile solids and nitrogen excretion at once. `"per_cell_feed"` (default) derives it from the feed mix of the cell the herd is in; `"national_feed"` from the country's own mix; `"uniform_medium"` assumes the IPCC `"Medium"` diet for every herd. The gridded rung is the

default because WHEP resolves a diet per cell and a diet varies within a country, so a national mix is a coarsening and an assumed Medium is coarser still. Both remain selectable. The assumption is never chosen implicitly, and the rung used is recorded in `method_ghg`. Ignored at Tier 1, whose emission factors carry no diet dimension.

`options` A named list of manure-engine options. All but nine defaults reproduce the behaviour in force before whep#949. The exceptions are `enteric_ef_source`, `ef3_source` and `slurry_crust`, which since whep#601 read the 2019 Refinement's Tier 1 enteric factors and direct manure N2O EF3 and move Tier 1 enteric CH4 and both tiers' direct manure N2O, `mcf_source`, which moved from the shipped table to the 2019 Refinement in whep#1022 and does move Tier 2 manure CH4, `mms_shares`, which moved from the unsourced placeholder table to the GLEAM 2.0 ingest in whep#958 and does move both tiers' manure N2O, `pasture_bo`, which since whep#1137 pairs the 2019 pasture MCF with its published `Bo` and moves Tier 2 manure CH4, and `tier2_uncovered`, which since whep#1028 gives species with no Tier 2 method their Tier 1 values instead of `NA`, `indirect_n2o_source`, which since whep#1245 reads the 2019 Refinement's leaching factors and moves both tiers' indirect manure N2O, and `indirect_n2o_fractions`, which since whep#1365 weights the Table 10.22 fractions over the manure-management split and moves it again.

`indirect_n2o_source` selects the edition of indirect_n2o_ef the indirect manure N2O reads:

- `"ipcc_2019"` (default): EF5 0.011 and FracLEACH-(H) 0.24, from Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26 of the 2019 Refinement, the current IPCC guidance and the EF5 the nitrogen balance already uses.
- `"ipcc_2006"`: EF5 0.0075 and FracLEACH-(H) 0.30, from Table 11.3, p. 11.24 of the 2006 Guidelines. These are the values WHEP shipped before whep#1245 under a 2019 citation, kept selectable so earlier figures stay reproducible.

EF4 (0.010) is the same under both. The edition also selects the Ch 11 FracGASM (0.21 in 2019, 0.20 in 2006) the pasture stream is priced at.

`indirect_n2o_fractions` selects which nitrogen fractions the indirect manure N2O volatilises and leaches:

- `"per_mms"` (default): IPCC 2019 Refinement, Vol 4, Ch 10, Eq. 10.26 and 10.27 (Updated), pp. 10.76-10.77. `FracGasMS` and `FracLeachMS` per animal category and system, from Table 10.22 (Updated), pp. 10.96-10.97, weighted over the row's own manure-management split, the same split the direct EF3 is weighted over. Pasture/range/paddock N, which Table 10.22 does not cover, is priced at the Ch 11 Table 11.3 FracGASM and FracLEACH-(H) (Eq. 11.9 and 11.10), as the IPCC prices grazing deposition; no other WHEP chain prices it, and the direct path prices the same N at EF3, so it is counted once. The managed-system fractions are the 2019 table's under either edition, as in `apply_management_losses()`: the 2006 Table 10.22 leaves most pairs blank.

- `"single"`: one FracGasMS 0.20 (assumed, unverified) and one FracLEACH-(H) on all excreted N, the form before whep#1365. Kept so earlier figures stay reproducible.

`method_manure_n2o` records both choices, e.g. `indirect_ipcc_2019_per_mms` or `indirect_ipcc_2006_single`.

`enteric_ef_source` selects the Tier 1 enteric CH4 emission factors. Although it sits among the manure-engine options, it governs the enteric path only:

- `"ipcc_2019"` (default): ipcc_enteric_ef_tier1, the 2019 Refinement's Tables 10.10 and 10.11 (Updated) as published: cattle and buffalo by IPCC region, and sheep, goats and swine at the productivity system the Table 10.10 footnote assigns the region (high in North America, Europe and Oceania, low elsewhere). A row the edition prices at nothing keeps the as-shipped factor, stamped `enteric_ef_as_shipped_fallback`.
- `"as_shipped"`: ipcc_2019_enteric_ef_cattle and ipcc_2019_enteric_ef_other, the 2006 Guidelines' factors with four cells in neither edition, read before whep#601. Kept so earlier figures stay reproducible.

`method_enteric` records which (`enteric_ef_ipcc_2019` or `enteric_ef_as_shipped`). On FAOSTAT 2020 heads the default raises Tier 1 enteric CH4 from 110.8 to 125.3 Tg. Taking high productivity everywhere instead would give 134.2 Tg, low everywhere 124.2 Tg.

`ef3_source` selects the direct manure N2O emission factors, at both tiers: `"ipcc_2019"` (default) or `"as_shipped"`, the two editions of ipcc_manure_ef3.

`slurry_crust` sets the share of liquid slurry with a natural crust, which mixes the 2019 Table 10.21 (Updated) EF3 with a crust (0.005) and without one (0) into the engine's single `"Liquid/Slurry"` label: `"half"` (default, 0.0025, GLEAM 2.0's stated assumption for its pig and chicken slurry, extended to the other species), `"none"` (0) or `"all"` (0.005). It has no effect under `ef3_source = "as_shipped"`, whose 0.002 is a single row.

`method_manure_n2o` records the EF3 choice (`ef3_ipcc_2019; slurry_crust_half` or `ef3_as_shipped`). On FAOSTAT 2020 heads the default lowers Tier 1 direct manure N2O from 1.523 to 1.178 Tg.

`mms_shares` selects which half of regional_mms_distribution the split is read from: `"gleam_2_0"` (default) is the GLEAM 2.0 Supplement S1 Tab. 4.2-4.11 ingest, `"placeholder"` the unsourced table it replaced in whep#958. The placeholder stays selectable so the values WHEP published before that ingest remain reproducible and the sensitivity to it stays measurable; it is not a defensible alternative estimate.

`mms_region` selects how the manure-management split in regional_mms_distribution is keyed:

- `"as_available"` (default): a row uses its own region when the frame already carries a `region` column, and the `region == "Global"` split otherwise. Tier 1 resolves a region for the (sourced) per-head N-excretion table and so takes the region-specific split; Tier 2 carries no region and so takes the Global one.

- `"resolve"`: the IPCC region is resolved from `iso3`, `area_code` or `polity_area_code` where it is missing, which makes the table's region-specific rows live on the Tier 2 path too. Opt-in because it changes which rows of the table apply, not because the rows are doubtful: since whep#958 they are the GLEAM 2.0 ingest.
- `"global"`: every row takes the `region == "Global"` split, whatever region column it carries.

`mcf_source` selects which methane conversion factor table the Tier 2 manure CH4 weighting reads:

- `"ipcc_2019"` (default): the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 (Updated) of the 2019 Refinement, which is the current IPCC guidance.
- `"ipcc_2006"`: the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 of the 2006 Guidelines.
- `"as_shipped"`: climate_mcf, whose live rows are predominantly the 2006 Guidelines Table 10.17 but with six cells that match no published IPCC value (whep#601, whep#1022). Kept selectable so an older run can be reproduced; it is no longer the default, because values whose provenance could not be established should not be what ships.

Neither edition publishes one number per Cool/Temperate/Warm zone for every system, so both as-published tables apply a stated collapse rule; see climate_mcf_ipcc. Both rules are WHEP's, not the IPCC's, and the default makes them live. `method_manure_ch4` records the table used.

`pasture_bo` selects the methane potential (`Bo`) the Tier 2 manure CH4 prices the pasture/range/paddock stream at. The 2019 Refinement publishes its single 0.47 percent pasture MCF as half of a pair that "must always be used in conjunction with a B0 value of 0.19" (Vol 4, Ch 10, Table 10.17 (Updated) footnote 2, p. 10.70; Section 10.4.2, p. 10.66); the pair is the `paired_bo_m3_kg_vs` column of climate_mcf_ipcc (whep#1137).

- `"paired"` (default): a stream whose MCF row carries a paired `Bo` is priced at it, every other stream at the animal-category `Bo` of ipcc_tier2_bo_values. Only `mcf_source = "ipcc_2019"` publishes a pair, so under the other two sources this changes nothing.
- `"species"`: every stream takes the animal-category `Bo`, the behaviour before whep#1137. Under `"ipcc_2019"` that is the hybrid the Refinement rejects; kept so the earlier figures stay reproducible and the sensitivity to the pairing stays measurable.

`method_manure_ch4` records per row which applied (`pasture_bo_paired` or `pasture_bo_species`) wherever the row has manure on a stream that carries a published pair.

`climate_source` selects where the climate zone the methane conversion factors in the MCF table are read at comes from. A `climate_zone` a row already carries is always used and stamped `climate_from_data`; the option governs only the rows left without one, whether that is a hole in a supplied column or a wholly absent column.

- `"assumed"` (default): fill with `assumed_climate_zone`.
- `"from_data"`: abort instead of assuming.

`assumed_climate_zone` is the zone `"assumed"` fills in: `"Cool"`, `"Temperate"` (default) or `"Warm"`. It is an assumption, not a measurement; `method_manure_ch4` records per row which of the sources applied, and this argument exists so the sensitivity to the assumption can be measured (whep#949).

`tier2_uncovered` says what the Tier 2 path does with a species WHEP has no Tier 2 method for. The energy balance needs the maintenance and activity coefficients of Tables 10.4 and 10.5, which ipcc_tier2_energy_coefs ships for cattle, buffalo, sheep and goats only; the 2019 Refinement itself suggests Tier 1 for camels, horses, mules and asses and swine and has no enteric method for poultry (Vol. 4 Ch. 10, Table 10.9 (Updated)), and its Tier 2 manure equations for swine and poultry need a country-specific dry-matter intake (Equation 10.32A) that WHEP does not hold. Although it sits among the manure-engine options, it governs the enteric path too.

- `"tier1"` (default): those species take the Tier 1 enteric CH4, manure CH4 and manure N2O, written into the Tier 2 output columns and stamped `"IPCC_2019_Tier1"` in `method_enteric`, `method_manure_ch4` and `method_manure_n2o`, with a message naming them. This is the IPCC's own suggested method for them, so a Tier 2 inventory keeps the whole herd rather than silently covering fewer animals than Tier 1.
- `"leave_na"`: they keep `NA` emissions, the behaviour before whep#1028, with a warning naming them. Kept so a ruminant-only Tier 2 figure stays reproducible.
- `"abort"`: any such species aborts, naming it.

`data` Optional named list of pre-loaded inputs to avoid remote reads: `primary_prod` (the `get_primary_production()` output) and, for Tier 2 with either feed-derived diet, `feed_intake` (the `get_feed_intake()` output). `primary_prod` falls back to its reader when absent; `feed_intake` does not, because `get_feed_intake()` rebuilds the whole feed allocation and would silently turn this extension into an hours-long build. Supply it, or choose `method_diet = "uniform_medium"`.

`example` If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (livestock emissions in kilograms CO2e), `method_ghg` (the chosen tier and GWP standard, e.g. `"IPCC_2019_Tier1_AR6"`), `method_mms` and `method_manure_ch4` (the manure-engine choices the summed rows took, `NA` when nothing reached the manure engine), plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_livestock_ghg_extension(example = TRUE)
```

---

build_livestock_nutrient_flows

Build livestock nutrient flows from realised feed intake.

Description

Top-level driver that traces the nitrogen, carbon and volatile solids in realised feed intake (the `redistribute_feed()` result) through livestock excretion, manure management and application to soil by land use and crop, plus the management-loss side-streams. It chains `estimate_n_excretion()`, `split_manure_management()`, `apply_management_losses()` and `allocate_manure_to_land()`; at the `"subnational"` resolution it additionally spills each cell's un-placeable surplus to neighbouring cells with `allocate_manure_transport()` before local disposal. Every method choice is recorded in a `method_*` provenance column and the nitrogen balance (excreted = applied + management losses) is conserved.

Usage

```
build_livestock_nutrient_flows(
  intake,
  resolution = "national",
  methods = list(),
  gridded = NULL,
  bedding = NULL
)
```

Arguments

`intake`	A tibble of realised feed intake (the `redistribute_feed()` result); see `estimate_n_excretion()` for the required columns. At `"subnational"` resolution `sub_territory` is the `"lon_lat"` cell id.
`resolution`	One of `"global"`, `"national"` (default) or `"subnational"`. Transport between cells runs only at `"subnational"`.
`methods`	A named list of per-stage option lists, any of `excretion`, `split`, `bedding`, `losses`, `allocation` and `transport`, each forwarded to the matching pipeline function's `options`. `transport` also takes `land_split`, which decides where manure trucked into a cell lands inside that cell's remaining room: `"cropland_first"` (default) fills the remaining cropland room and puts the rest on grassland, the same priority `allocate_manure_to_land()` gives a cell's own collected manure; `"room_share"` splits it in proportion to the remaining cropland and grassland room; `"cropland_only"` books all of it as cropland, the behaviour before whep#341, which counts grassland room as cropland. The total delivered to each cell is the same under all three.
`gridded`	The land-surface layer (`crops` and optional `grass` tibbles) passed to `allocate_manure_to_land()` required for the default `"potential_uptake"` cap. `NULL` is treated as an empty list. The `crop` column of `crops` is a *code*: `as.character(item_prod_code)` from items_prod_full. That is the one contract, honoured by both consumers of this function's `applied` stream (the carbon balance and the nitrogen balance); a crop name is still resolved by the nitrogen path but is deprecated and warns.
`bedding`	An optional bedding supply from `build_residue_bedding_supply()`. When given, `add_manure_bedding()` places it on the litter-using housed streams before the management losses, so the manure reaching the field is bedded farmyard manure rather than excreta alone. `NULL` (default) leaves every number excreta-only. Build the supply from `calculate_residue_destinies()`, whose default bedding share is Wirsenius (2000) Table 3.21 (whep#1005); the turnkey soil-carbon reader of `build_soil_carbon_inputs()` does so.

Value

A named list with `applied` (manure applied per `land_use x crop (x cell)` with `manure_type` (`"Excreta"`/`"Solid"`/`"Liquid"`) and all `method_*` provenance columns, including `method_transport_land_use`, the `land_split` used, `NA` unless the resolution is `"subnational"`; and the grazing-excreta

species mix: `excreta_cpp_share`, the share of a row's N deposited by cattle, poultry and pigs, the IPCC 2019 EF3PRP,CPP animals (Vol 4, Ch 11, Table 11.1 (Updated), p. 11.13), with sheep and other animals (SO) making up the rest, and `method_excreta_species`, which records where the share came from (`"engine_livestock_category"`) or why an `"Excreta"` row has none (`"fallback_label_no_species"`, a row the manure-type split labelled `"Excreta"` for want of a share, which is not grazing deposition). Both are `NA` on `"Solid"` and `"Liquid"` rows), `losses` (management-loss side-streams per polity) and `excretion` (the per-category excretion totals).

Examples

```
intake <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~livestock_category,
  ~item_cbs_code, ~feed_quality, ~intake_dm_t,
  2020L, "203", NA, "Cattle_milk", 2513L, "high_quality", 200,
  2020L, "203", NA, "Cattle_milk", NA, "grass", 600
)
gridded <- list(
  crops = tibble::tribble(
    ~year, ~territory, ~sub_territory, ~crop, ~manure_n_receptivity, ~crop_n_cap,
    2020L, "203", NA, "44", 6, 200,
    2020L, "203", NA, "15", 4, 200
  )
)
build_livestock_nutrient_flows(intake, gridded = gridded)
```

`build_loss_wedge` *Build the supply-to-intake loss wedge.*

Description

Returns the share of retail-level protein supply that does not become intake, per country and year, and the divisor `1 / (1 - omega)` that turns a requirement into a supply floor. It replaces part of the unsourced 1.35 multiplier the SJOS-N axis used to carry (whep#753).

Rates come from Gustavsson et al. (2011), *Global food losses and food waste*, FAO, Annex 4, which tabulates loss percentages for seven commodity groups in seven world regions at five food-chain steps. Only the two steps at or after the retail shelf are composed, because FBS food availability is already measured there: `omega_group = 1 - (1 - d/2) * (1 - c/2)`, with `d` and `c` the distribution and consumption rates. Country-year `omega` is the protein-weighted mean of the group values over the supplied food basket, so it varies with a country's diet composition while the underlying rates do not.

`method = "gustavsson_half_min"` (default) takes the minimum of each rate across the seven regions and halves it. It is a **deliberate lower bound on unavoidable loss, not an estimate of achievable loss**, for two reasons that the halving does not repair. The consumption-step minimum is sub-Saharan Africa in every commodity group, and those are scarcity figures rather than efficiency figures; the distribution-step minima, by contrast, are genuine best practice. On WHEP's 2010 basket it gives `omega` near 2.5%, which is below FAOSTAT's retail-only median of 2.83% on energy (item 21059) even though this wedge

additionally spans the household step – a different basis and a different statistic, so not a contradiction, but confirmation that the construction is conservative.

`"gustavsson_min"` drops the halving and roughly doubles the wedge (near 4.9% on the same basket); `"none"` sets it to zero and makes every floor an explicit lower bound. All are alternatives, never fallbacks: the choice is stamped in `method_loss_wedge`.

`"gustavsson_regional_actual"` is the sensitivity arm, and it is **not an unavoidable-loss estimate at all**: it drops both the minimum and the halving and gives each country its Annex 1 region's own observed rates. On the 2010 world basket that is 14.2% against the default's 2.5%, a floor divisor of 1.166, spanning 4.1% to 21.4% across countries. Its country structure is contested – Gustavsson's rich-high gradient is the opposite sign to UNEP's Food Waste Index – so it quantifies that disagreement rather than resolving it.

Annex 1 lists 152 countries, covering 99.0% of 2010 world food protein. Areas outside it take the unweighted mean rate across the seven regions by default (`coverage = "global_mean"`), stamped as such in `method_region`; `"annex1_only"` returns nothing for them instead. Annex 1's "China" does not disambiguate, so both WHEP codes take Industrialized Asia: `CHN` (area 41, "China, mainland") and the aggregate area 351, which carries no `iso3c` and is what the FBS pin reports food on. Keying on `iso3c` alone would drop area 351 and with it a fifth of world food protein.

Items that Gustavsson's Annex 2 does not place in a commodity group carry no rate rather than borrowing a neighbour's. They are dropped from the weighting, which gives them the basket's mean wedge implicitly, and their share is reported in `protein_grouped_share` so the choice stays visible. On the 2010 world basket they are 5.0% of food protein, eggs alone being 3.7%; assigning eggs to meat or to dairy instead moves `omega` by less than 0.1 percentage points either way.

Usage

```
build_loss_wedge(
  data = list(),
  method = c("gustavsson_half_min", "gustavsson_min", "gustavsson_regional_actual",
    "none"),
  coverage = c("global_mean", "annex1_only"),
  protein_basis = c("edible_portion", "whole_commodity", "product_nitrogen")
)
```

Arguments

`data`	Named list of injected inputs. Supply the basket either as `protein_supply` (`year`, `area_code`, `item_cbs_code`, `protein_t`) or as `cbs_food` (`year`, `area_code`, `item_cbs_code`, `food_t`), which is converted through the same nutrition lookup `build_food_supply()` uses. `biomass_coefs`, `items_full`, `food_loss_wedge`, `food_loss_item_groups` and `food_loss_regions` override the packaged tables.
`method`	`"gustavsson_half_min"` (default), `"gustavsson_min"`, `"gustavsson_regional_actual"` or `"none"`.
`coverage`	How areas outside Annex 1's 152 countries are handled by `"gustavsson_regional_actual"`: `"global_mean"` (default) or `"annex1_only"`. Ignored by the region-invariant methods.

protein_basis Passed to the nutrition lookup when the basket is given as cbs_food; see build_food_supply(). Defaults to "edible_portion", matching the supply the floor is compared against.

Value

A tibble keyed by year, area_code with omega, floor_divisor, protein_grouped_share, method_loss_wedge and method_region, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its area_code resolves to in that row's year:

- polity_area_code: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use reporting_polity_code to say which territory a row belongs to.
- reporting_polity_code: The polity itself, e.g. ESP-1846-1914. It is year-aware, so the same area_code resolves to different polities in different years, which is the point of the crosswalk.
- reporting_polity_name: Its name. It can differ from the area's own name where the area folds into an aggregate.
- reporting_polity_has_geometry: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. FALSE is a documented gap upstream, not an error.

Rows whose area_code resolves to no polity keep the columns with NA rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See add_polity_code() for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 (area, year) cells – add_polity_code() says so as mapping_status == "backcast_anchor", and polity_coverage_gaps() reports it as gap_kind == "backcast_anchor". These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so reporting_polity_code can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after SUD-1956-2011 ends, and its post-2011 rows carry that code. These columns do not say so: add_polity_code() reports such a row as mapping_status == "out_of_span", and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. polity_coverage_gaps() reports the stand-in rows of a built table, and options(whep.polity_mapping_status = "flag") (or "status") carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_loss_wedge(
  data = list(
    protein_supply = tibble::tribble(
      ~year, ~area_code, ~item_cbs_code, ~protein_t,
```

```
      2010L, 10L,        2511L,          100,
      2010L, 10L,        2605L,          100
    )
  )
)
```

`build_n_boundary_country` *Summarise gridded nitrogen exceedance to a country-year table.*

Description

Turns the crop rows of a `build_n_boundary_exceedance()` grid result into one row per country and year, holding the exceedance, the share of nitrogen inputs it represents, the share of the country's positive surplus it represents, and the share of that surplus that lies in comparison units above their critical surplus. Every quantity is formed from the attributed crop rows of cells with `coverage_state == "valid"`, so the numerator and every denominator cover the same rows. Inputs in cells that are out of the critical-surplus domain, lack a critical value or have no allowance area fall outside all of them, as do rows of a grassland-split component the comparison leaves out; rows naming no crop were removed by the exceedance itself.

The comparison unit is the cell, or, under the grassland split (`land_use = "all"`, `grassland_split = "image_density"`), the managed and the extensive-grassland component of a cell, which are compared with their own allowances and never net against each other. Under `regime_comparison = "separate"` the unit is the rainfed or the irrigated part of the cell or component, and `surplus` must keep its `water_regime` rows, which are then joined regime by regime. Deficit units never offset excess elsewhere. Two country quantities follow and both use the country's **own** crop surplus (`actual_n_t`), never a whole-cell surplus, which sums every polity in a shared border cell and would count that cell once per polity:

- `positive_surplus_n_t`: the country's own crop surplus in the comparison units whose surplus is positive. Without the split this is the cells with positive whole-cell surplus; with it, the positive managed surplus plus the positive extensive-grassland surplus of each cell, so a positive managed surplus is counted even beside a negative extensive one;
- `exceeding_surplus_n_t`: the same, restricted to the units whose overshoot over their allowance is positive.

The fraction of the country's positive surplus that is excess is `exceedance_share_of_positive_surplus = exceedance_n_t / positive_surplus_n_t`. It is the country exceedance itself over the same positive surplus, so it is not `beyond_share`, which counts the whole surplus of an exceeding unit and not only the part above its allowance.

Unit membership is decided from the unit's own state (its surplus and its overshoot), never from the crop-attributed exceedance. A unit whose crop shares are undefined (zero or ill-conditioned total surplus) attributes no exceedance to any crop and leaves it on a residual record, yet its crop rows stay in `positive_surplus_n_t` and `exceeding_surplus_n_t`

when the unit is positive and exceeding, so `beyond_share` does not depend on the attribution. The residual exceedance stays in `unallocated_exceedance_n_t` of the diagnostics, which also count the units and rows involved.

The country's own crop surplus in a shared unit can be negative while the unit total is positive (the other polity carries the excess), and crop exceedance is a signed share of the unit overshoot, so a country ratio can fall outside `[0, 1]`: such negative own rows lower the country's positive surplus, and its exceedance can be negative or larger than that surplus. This affects `exceedance_share_of_positive_surplus`, `beyond_share` and `excess_share_of_inputs`. Such rows are kept, flagged in `ratio_outside_unit`, and counted in the diagnostics; they are never clipped.

The world sum of `positive_surplus_n_t` equals the sum of the positive unit surpluses of the valid cells. The world sum of `exceedance_n_t` equals the summed cell exceedance minus the exceedance left on unallocated residual rows, which the diagnostics report and reconcile per year; the function aborts if they do not.

Every exceeding unit is a positive-surplus unit only with `negative_critical = "clamp"`: a zero allowance means an overshoot needs a positive surplus. With `"keep"` a unit with a negative critical surplus can overshoot with a zero or negative surplus. Its exceedance stays in `exceedance_n_t`, but the unit is outside both `positive_surplus_n_t` and `exceeding_surplus_n_t`, so `beyond_share` does not see it; the diagnostics report that overshoot in `overshoot_without_surplus_n_t` (zero under the clamp).

Usage

```
build_n_boundary_country(
  exceedance,
  surplus,
  ag_land,
  nourishment = NULL,
  beyond_share_cut = 0.5,
  example = FALSE
)
```

Arguments

`exceedance`	A `build_n_boundary_exceedance()` result at `resolution = "grid"` with `metric = "surplus"`, possibly bound over several years. Its `negative_critical`, `land_use`, `grassland_split` and `regime_comparison` stamps must each be constant. It may carry the grassland split.
`surplus`	The `calculate_n_surplus()` output the grid was computed from, carrying `lon`, `lat`, `area_code`, `item_cbs_code`, `year` and `n_input_std_t`. Every crop row of `exceedance` must find exactly one row here; a row that does not aborts.
`ag_land`	A `build_ag_land_support()` table (`area_code`, `year`, `area_ha`). Its hectares are summed per country and year into `ag_area_ha`, WHEP's agricultural area, the basis of per-hectare exceedance. A country-year with no row keeps `NA`.

`nourishment` Optional `normalize_nourishment()` output (`year`, `area_code`, `nourish`), one row per country-year. When supplied, the result gains `nourish` and `sjos_class`, the country boundary side crossed with the nourishment class over the levels of sjos_levels; a country-year with no boundary side or no class is `NA`.

`beyond_share_cut` Share of the positive surplus above which the country is on the `"Exceedance"` side, a number in `[0, 1)`. Defaults to `0.5`, a WHEP criterion.

`example` If `TRUE`, return a small fixture.

Value

A named list of two tibbles.

`country`, one row per `area_code` and `year`:

- `exceedance_n_t`: country exceedance, t N (sum of the crop-attributed `exceedance_n_t`, signed).
- `input_std_n_t`: sum of `n_input_std_t` over the same rows, t N.
- `excess_share_of_inputs`: `exceedance_n_t / input_std_n_t`.
- `exceedance_share_of_positive_surplus`: `exceedance_n_t / positive_surplus_n_t`, `NA` when the denominator is not positive.
- `positive_surplus_n_t`, `exceeding_surplus_n_t`, `beyond_share`, `exceedance_share_of_positive_surplu` `boundary_side`: see above.
- `ag_area_ha`: WHEP agricultural area, ha.
- `signed_denominator_nonpositive`: `positive_surplus_n_t` or `input_std_n_t` is zero or negative.
- `ratio_outside_unit`: `beyond_share`, `exceedance_share_of_positive_surplus` or `excess_share_of_inputs` lies outside `[0, 1]` (beyond a rounding tolerance of `1e-9`).
- `nourish`, `sjos_class` when `nourishment` is given.
- `negative_critical`, `land_use`, `grassland_split`, `regime_comparison`, `beyond_share_cut`: the run stamps.
- the polity columns below.

`diagnostics`, one row per `year`, world level: `n_countries`, `input_std_n_t` and `all_input_n_t` (inputs in the compared rows and in every crop row of the land-use scope), `valid_input_fraction` (their ratio), `exceedance_n_t` (sum over countries), `unallocated_exceedance_n_t` (exceedance left on residual rows), `cell_exceedance_n_t` (summed cell exceedance), `exceedance_gap_n_t` (`cell - country - unallocated`, zero up to rounding), `positive_surplus_n_t`, `exceedance_share_of_positive` and `excess_share_of_inputs` (world ratios), `world_ratio_outside_unit` (either world ratio lies outside `[0, 1]`, beyond the `1e-9` tolerance), `overshoot_without_surplus_n_t` (overshoot of units with no positive surplus, zero under the clamp), and the counts `n_undefined_beyond_share`, `n_undefined_exceedance_share`, `n_undefined_excess_share`, `n_signed_denominator_nonpositive`, `n_ratio_outside_unit`, `n_missing_ag_area`, `n_undefined_attribution_rows` (crop rows whose attribution is undefined), `n_unallocated_units` (units whose overshoot sits on a residual record) and `n_unclassified` (`NA` without `nourishment`).

Boundary side

`boundary_side` is decided after aggregation, from the country `beyond_share = exceeding_surplus_n_t / positive_surplus_n_t`: the country is `"Exceedance"` when more than `beyond_share_cut` of its positive surplus lies in units above their critical surplus, and `"Within_boundary"` otherwise. The one-half default of `beyond_share_cut` is a WHEP criterion, not a published threshold. The share is `NA`, and the side `NA`, when `positive_surplus_n_t` is zero or negative: the country-year is left unclassified, flagged in `signed_denominator_nonpositive` and counted in the diagnostics. The labels are those `classify_sjos_n()` uses for its per-crop boundary side, which stays the producer-side classification.

Excess share of inputs

`excess_share_of_inputs = exceedance_n_t / input_std_n_t`, where `input_std_n_t` is the sum of `n_input_std_t` (synthetic fertiliser, manure and excreta, biological fixation, deposition and urban nitrogen) over the same crop rows as the exceedance. It is `NA` when `input_std_n_t` is zero or negative. The country ratio has a signed numerator, so it is bounded by `[0, 1]` only when the country's contributions to every cell with exceedance are non-negative; at world level, summed over all attributed rows, it is bounded once negative critical surpluses are clamped (`negative_critical = "clamp"`), because a unit overshoot is then at most its positive surplus, which is at most its inputs. The last step needs `surplus_n_t <= n_input_std_t` on every row. That holds for `surplus_method = "harvest_removal"`, where the surplus is the inputs minus non-negative harvest removals, but not for `"full_balance"`: its `n_balance_t` includes soil organic matter mineralisation, which `n_input_std_t` excludes, so the world share can exceed one even after the clamp. Under `"keep"` a unit with a negative critical surplus can overshoot by more than its inputs. The world ratio outside `[0, 1]` is reported in `world_ratio_outside_unit` of the diagnostics.

Exceedance share of positive surplus

`exceedance_share_of_positive_surplus = exceedance_n_t / positive_surplus_n_t`, with the exceedance signed as above. It is `NA` when `positive_surplus_n_t` is zero or negative, exactly where `beyond_share` is `NA`. The country ratio is guaranteed to lie in `[0, 1]` only when negative critical surpluses are clamped and the country's contributions to every positive unit are non-negative. At world level it is bounded once negative critical surpluses are clamped, because a unit overshoot is then at most its positive surplus; unlike the share of inputs, that bound needs no assumption on the surplus method.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.

- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_n_boundary_country(example = TRUE)
```

---

build_n_boundary_exceedance

Build source-exact gridded critical-nitrogen exceedance.

Description

Compares WHEP crop nitrogen pressure with the deposited Schulte-Uebbing et al. (2022) 2010 yield-gap critical surface. The calculation first aggregates every crop and polity contribution in a source cell, compares that one cell pressure with one critical allowance, and only then attributes the cell allowance, signed margin and positive overshoot back to crops. Total-input attribution uses crop input shares; surplus attribution uses signed crop surplus shares. Signed surplus shares can be negative or greater than one.

The critical layer must carry its deposited `source_area_ha` and `image_region`. `resolution = "cell"` returns one row per source cell and year. Other resolutions return the approved crop attribution. Exactly zero or numerically near-zero pressure denominators keep the complete cell result but allocate no crop share; an explicit `cell_residual` record carries the unallocated critical allowance, signed margin and positive overshoot. APIs requiring complete crop attribution hard-error rather than fabricate a fallback.

Actual-pressure rows naming no crop cannot meet a critical allowance and are excluded before the cell comparison. The exclusion is reported, never silent: a message names the rows and the pressure they carried when that pressure is zero (the only case a gridded `build_nitrogen_balance()` produces), and a warning when it is not.

Usage

```
build_n_boundary_exceedance(
  surplus = NULL,
  critical = NULL,
  land_use = c("ara", "all", "igl"),
  resolution = c("grid", "cell", "polity", "country", "image_region"),
  metric = c("surplus", "input", "new_fixation"),
  cell_polity = NULL,
  allocation_scenario = c("yield_gap", "no_increase", "new_fixation"),
  actual_year = NULL,
  critical_reference_year = NULL,
  actual = NULL,
  boundary = NULL,
  indicator = NULL,
  land_class = NULL,
  impact_scope = NULL,
  grassland_split = c("image_density", "none"),
  grassland = list(classes = NULL, extensive_budget = NULL, critical_ara = NULL,
    critical_igl = NULL),
  negative_critical = c("keep", "clamp"),
  regime_comparison = c("netted", "separate"),
  binding = NULL,
  example = FALSE
)
```

Arguments

`surplus`	A `calculate_n_surplus()` output with the grid/crop/year key. Surplus mode uses signed `surplus_n_t` when present, otherwise derives it from `surplus_kgn_ha * area_ha / 1000`. Input mode uses `n_input_std_t`.
`critical`	A `read_critical_n()` critical layer. In addition to layer identity, it must carry deposited `source_area_ha` and `image_region`.
`land_use`	Source land class: `"ara"`, `"igl"`, or `"all"`.
`resolution`	Output grain: source `"cell"`, crop-cell `"grid"`, crop `"polity"`/`"country"`, or crop `"image_region"`.
`metric`	Selected actual pressure: signed `"surplus"` or total agricultural `"input"`.
`cell_polity`	Retained for API compatibility. IMAGE membership now comes from the deposited cell-key crosswalk in `critical`; country attribution comes from the crop rows themselves.
`allocation_scenario`	Only source-exact `"yield_gap"` is supported. `"no_increase"` and `"new_fixation"` hard-error.

`actual_year` The actual-pressure year to compare. Must select exactly one year from `surplus` and is always retained in results.

`critical_reference_year`
Must be `2010`, matching the deposited fixed reference surface, and is always retained in results.

`actual` Alias of `surplus` for the selector-oriented interface.

`boundary` Alias of `critical` for the selector-oriented interface.

`indicator` Selector-oriented pressure name: `"surplus"`, `"total_input"`, or the unsupported `"new_fixation"` mode (which hard-errors). When supplied, it overrides `metric`.

`land_class` Alias of `land_use`.

`impact_scope` Deposited impact surface: `"mi"`, `"sw"`, `"gw"`, or `"de"`. When supplied, it is validated against the critical layer.

`grassland_split`
Grassland treatment under `land_use = "all"`: `"image_density"` (default) splits each cell into a managed and an extensive component (see the section below); `"none"` compares one cell pressure with the deposited allowance, as before the split existed. Ignored for `"ara"` and `"igl"`. Under `"image_density"`, `metric = "new_fixation"` aborts: the archive has no extensive budget for it.

`grassland` Named list of the inputs the split needs, all required under `grassland_split = "image_density"` (their absence aborts; there is no fallback): `classes`, the `build_grassland_intensity_classes()` table (one row per cell and year: `cell_id`, `lon`, `lat`, `year`, `country_2010`, `image_region`, `a_crop_ha`, `grass_ha_image`, `whep_grass_ha`, `image_class_2010`, `grassland_class`, `method_grassland_split`), and `extensive_budget`, IMAGE's 2010 extensive-grassland budget per cell (`cell_id`, `ext_input_kgn_ha`, `ext_surplus_kgn_ha`); and `critical_ara` and `critical_igl`, the `read_critical_n()` layers with `land_use = "ara"` and `"igl"` for the same threshold and metric as `critical` (validated). `critical` itself still defines the cell domain and is checked against the class table's areas. A cell absent from `classes` must carry no grassland pressure; it is compared as cropland only.

`negative_critical`
Treatment of a cell whose critical value is below zero. `"keep"` (default) compares the actual pressure with the deposited value as it is, as the source does. `"clamp"` sets it to zero (a zero allowance) before the cell comparison. The choice is stamped in every output row as `negative_critical`; see the Negative critical surplus section.

`regime_comparison`
Treatment of a balance split into rainfed and irrigated rows (`build_nitrogen_balance()`'s `methods$regime`). `"netted"` (default) sums the two regimes of a cell before the comparison, so irrigated excess is offset by rainfed headroom in the same cell. `"separate"` compares each regime as its own part of the cell (or of the grassland-split component), so it is not offset. It needs a `water_regime` column in `surplus` and aborts without one. The choice is stamped in every output row as `regime_comparison`; see the Rainfed and irrigated parts section.

`binding` Optional `build_critical_n_binding()` output for the same land-use scope. When supplied, its per-cell `binding_threshold` and `binding_matches_mi` are carried into the cell and grid results; when `NULL` (default) both columns are `NA`.

`example` If `TRUE`, return the package fixture.

Value

A tibble at the requested grain. Cell results retain actual and critical masses, signed margin, positive overshoot, coverage state, integer source-grid key, IMAGE context, explicit years, selectors, and provenance. `critical_kgn_ha` is the value compared (after the `negative_critical` treatment) and `source_critical_kgn_ha` the deposited one; the two differ only in clamped cells. Crop results additionally retain the signed pressure share and crop-attributed quantities, which reconcile algebraically to the cell. `exceedance_n_t` is the crop's share of the cell overshoot `pmax(actual - critical, 0)` and `within_boundary_n_t` is `actual_n_t - exceedance_n_t`, so the two always sum to the actual pressure. Summed over a cell, `within_boundary_n_t` is `min(actual, critical)`: under `negative_critical = "keep"` it is negative wherever the critical value is negative, and under `"clamp"` it is negative only where the actual pressure itself is. Under the grassland split these hold per component (managed, extensive) rather than per cell. Cell and grid results also carry the split components: `managed_actual_n_t`, `managed_critical_n_t`, `managed_positive_overshoot_n_t`, `extensive_actual_n_t`, `extensive_critical_n_t`, `extensive_positive_overshoot_n_t`, their areas (`managed_area_ha`, `extensive_area_ha`), compared rates (`managed_critical_kgn_ha`, `extensive_critical_kgn_ha`, after the `negative_critical` treatment) and the rates before it (`source_managed_critical_kgn_ha`, `source_extensive_critical_kgn_ha`), coverage states, `excluded_actual_n_t` (of which `excluded_igl_actual_n_t` is intensive grassland without an `"igl"` rate), `grassland_class`, `method_allowance_managed`, `method_allowance_extensive` and `method_grassland_split` (the per-cell class method, `"no_grassland"` for a cell outside the class table, `"none"` without the split). These are `NA` when the split is not applied, so the schema does not depend on it. Under the split, `critical_kgn_ha` and `source_critical_kgn_ha` remain the `"all"`-scope surface (it defines the domain and checks the `"ara"`/`"igl"` layers); the cell allowance is the sum of the component allowances. Grid rows add `boundary_component` (`"managed"`/`"extensive"`, `NA` without the split); within the split, `pressure_share` is the row's share of its component, and `binding_threshold`/`binding_matches_mi` are `NA` on extensive rows (see the Negative critical surplus section). Every row carries the call-level `grassland_split`; aggregated rows list the per-cell methods they span in `method_grassland_split`, separated by `;`.

Grassland intensity split (`land_use = "all"`)

The deposited `all`-scope critical rate is per hectare of cropland plus IMAGE-intensive grassland, while WHEP's pressure covers all grassland. With `grassland_split = "image_density"` (the default) each cell is compared as two independent components:

- **managed** – crop rows plus grassland rows (CBS 3000, 3002, 3003) of a cell classed intensive, against the cell's own `"ara"` critical rate times its IMAGE 2010 cropland plus, when the cell is classed intensive, an `"igl"` critical rate times its IMAGE 2010 grassland. The deposited per-hectare `"ara"` and `"igl"` layers combine exactly into

the `"all"` layer (area-weighted, measured on the archive for every threshold and both metrics), so with the IMAGE 2010 classes this reproduces the `"all"`-scope allowance;

- **extensive** – grassland rows of a cell classed extensive, against IMAGE's 2010 extensive-grassland input or surplus per hectare (Schulte-Uebbing et al. 2022, SI Supplementary Table 4) times the cell's IMAGE 2010 grassland.

Cell overshoot is the **sum** of the two component overshoots: headroom on one never nets against excess on the other. Cell actual, critical and margin are the sums over the compared components. A component with no allowance area but non-zero pressure (in practice crop pressure where IMAGE has no cropland and the cell is extensive) is excluded from its comparison (`managed_coverage_state`/`extensive_coverage_state` `"zero_land"`); a component with area but no rate even after transfer is `"missing_critical"`. Excluded pressure is reported per cell in `excluded_actual_n_t` and in a message (classes `whep_nbx_zero_land_component`, `whep_nbx_missing_critical_component`), and never counts as overshoot. Crop attribution shares each component's allowance, margin and overshoot among that component's rows only, and reconciles to the component and to the cell. A compared component with allowance area but no pressure row keeps its allowance in a `cell_residual` record naming the component.

Declared assumptions (constructed methods without published precedent): the classes are IMAGE's 2010 production-system map moved through time by a national grazing-density proxy (see `grassland$classes`); the extensive allowance is IMAGE's 2010 budget held constant – "no more than in 2010", not an environmental limit, so 2010 extensive exceedance is zero wherever WHEP's 2010 extensive pressure equals IMAGE's; a rate a cell lacks for its class is borrowed from the nearest cell (great-circle distance) with one in the same 2010 country, else the same IMAGE region – an `"igl"` rate for grassland promoted from extensive to intensive, an extensive budget rate for grassland classed extensive; cropland keeps its own `"ara"` rate and is never lent one – stamped in `method_allowance_managed`/`method_allowance_extensive` (`"archive"`, `"nearest_country"`, `"nearest_region"`, `"none"`, or `"no_area"` for a component without area; `NA` outside the critical domain). IMAGE 2010 intensive grassland with no published `"igl"` value is not lent one (maintainer decision 2026-09-24): its pressure is left out of the comparison and reported in `excluded_igl_actual_n_t`, the cell's cropland is still compared at its own `"ara"` rate, and the managed method is `"none"`. Allowance areas are IMAGE 2010 areas, fixed, except WHEP grassland in a cell with no IMAGE grassland, whose extensive allowance uses WHEP's own grassland area of the year (method suffix `"_whep_area"`, maintainer decision 2026-09-24). `grassland_split = "none"` reproduces the unsplit comparison exactly.

Negative critical surplus

Schulte-Uebbing et al. (2022, Methods) set critical fertilizer and manure inputs to zero where non-agricultural losses alone exceed a threshold, but keep biological fixation and deposition in the critical input, so their deposited critical surplus stays negative in those cells (on the `"mi"` surface: 1,796 of 28,881 cells for `"all"`, minimum -396 kg N/ha; 2,075 of 28,573 cells for `"ara"`, minimum -317 kg N/ha). `negative_critical = "keep"` follows the source, and it is the setting under which the published 2010 decomposition (43 Mt N allowable plus 76 Mt N exceedance, 119 Mt N current surplus) is reproduced. `"clamp"` is a declared departure from the source: it gives such cells a zero allowance instead of a negative one, which lowers their overshoot to the actual pressure and keeps the cell within-boundary mass at or above zero wherever the actual pressure is.

Under `grassland_split = "image_density"` the treatment applies to each component's allowance, the unit a component is compared against: a negative managed allowance (the cell's `"ara"` rate on its cropland plus its `"igl"` rate on its intensive grassland, which can net against each other as they do inside the `"all"` rate) becomes zero, and so would a negative extensive one (IMAGE's 2010 extensive budget, which is never negative on the deposited archive). The component rates are not clamped one by one, so with the 2010 classes the managed comparison under `"clamp"` matches the unsplit `"all"`-scope comparison under `"clamp"` (to the 1 % within which the `"ara"` and `"igl"` layers combine into `"all"`). A binding threshold names the impact that sets a critical surplus, so it describes the managed allowance only; the extensive allowance is a 2010 level, not a threshold, and its grid rows carry no binding label.

Rainfed and irrigated parts

Under `regime_comparison = "separate"` each comparison unit (the cell, or under the grassland split its managed and its extensive component) is divided into a rainfed and an irrigated part. Each part gets a share of the unit allowance equal to its share of the unit's `area_ha`, summed over its rows. The deposited critical surface has one rate per hectare and no regime axis, so this keeps the rate the same on both regimes. It also keeps every unit allowance, actual pressure and signed margin exactly as under `"netted"`. The unit overshoot becomes the sum of the part overshoots, `sum(pmax(part actual - part allowance, 0))`, which is never below the netted `pmax(actual - allowance, 0)`. The areas are WHEP's harvested hectares. Irrigated land is more often multi-cropped, so harvested area can overstate the irrigated share of physical land. This is a declared assumption with no published precedent. A unit with no harvested area gives its whole allowance to the one part that carries pressure, which is the netted comparison. It aborts (`whep_nbx_regime_no_area`) when both parts do. Crop rows share the allowance, margin and overshoot of their own part. They keep their `water_regime`, plus that part's `regime_actual_n_t`, `regime_critical_n_t` and `regime_positive_overshoot_n_t`. Cell and grid rows add the per-cell totals `rainfed_actual_n_t`, `rainfed_critical_n_t`, `rainfed_positive_overshoot_n_t` and their `irrigated_*` counterparts, summed over the parts that carry pressure rows. Under `"netted"` all of these are `NA`, so the schema does not depend on the choice.

Examples

```
build_n_boundary_exceedance(example = TRUE)
```

---

build_n_boundary_percapita

Build the per-capita nitrogen-boundary versus nourishment scatter.

Description

Normalizes each country's total anthropogenic reactive-nitrogen per capita against the world per-capita planetary-nitrogen boundary and joins it to the nourishment normalization, yielding the safe-and-just nitrogen scatter (one point per country-year: nourishment adequacy on the x axis, boundary pressure on the y axis, population as the

point weight). The world per-capita boundary converts the Tg N/yr limits in `params` (`boundary_low`, `boundary_high`) to kg N/cap/yr by dividing by the world population (the complete `population` table summed per year, falling back to `nourishment`). The upper per-capita boundary is capped at the packaged `per_capita_cap`. The normalization is the Global piecewise: `n_percapita_kg / low_pc` below the lower bound, `1 + (n_percapita_kg - low_pc) / (high_pc - low_pc)` within the band, and `min(1 + n_percapita_kg / high_pc, 6)` above the upper bound. The result is then scaled by `afs_share`, the agri-food-system share of the boundary (a parameter, default 0.8, flagged provisional).

Usage

```
build_n_boundary_percapita(
  n_percapita,
  nourishment,
  population = NULL,
  params = NULL,
  afs_share = 0.8
)
```

Arguments

`n_percapita`	A tibble keyed by `year`, `area_code` with `n_percapita_kg`, the country total anthropogenic reactive N per capita (kg N/cap/yr, synthetic plus biological fixation), from `build_n_percapita()` or injected directly.
`nourishment`	A tibble keyed by `year`, `area_code` with `value_norm` (the nourishment normalization, for example a `normalize_nourishment()` output) and `population` (absolute persons), whose per-year population sum sets the world per-capita boundary.
`population`	Optional complete population tibble keyed by `year`, `area_code`, with `population` in persons. Supplying it prevents countries missing nourishment data from being omitted from the world denominator.
`params`	Boundary parameters, defaulting to n_boundary_params, read for the `boundary_low` and `boundary_high` Tg N/yr limits.
`afs_share`	The agri-food-system share of the planetary boundary applied to the normalized boundary pressure. Defaults to `0.8` (provisional).

Value

A tibble keyed by `year`, `area_code` with `nourish_norm` (the nourishment normalization), `boundary_norm` (the afs-scaled per-capita boundary normalization) and `population`.

Examples

```
build_n_boundary_percapita(
  n_percapita = tibble::tribble(
    ~year, ~area_code, ~n_percapita_kg,
    2000L, 10L, 5,
    2000L, 20L, 15
  ),
```

```
  nourishment = tibble::tribble(
    ~year, ~area_code, ~value_norm, ~population,
    2000L, 10L, 0.8, 3e9,
    2000L, 20L, 1.5, 3e9
  )
)
```

`build_n_deposition` *Build gridded atmospheric nitrogen deposition inputs.*

Description

Combines HaNi NHx and NOy deposition into a total nitrogen deposition rate per WHEP grid cell, using the true latitude-dependent 0.5-degree cell area to convert the deposited mass into a per-hectare rate, and a polity share of the cell to derive the absolute mass a polity receives.

The cell's deposited mass is split across the polities holding the cell in proportion to `polity_area_ha`, the geodesic territory each holds in it, as `build_polycell_support()` measures it. The transitional alternative is `polity_frac`, the subcell-count share `build_cell_polity()` carries, which is quantised to 1/36 of a cell; it stays selectable so the two partitions can be compared, and it is what a support table carrying no `polity_area_ha` is split by. Either way the split is a share of the cell, so the source mass is redistributed and never created or destroyed.

Each polity's share is then decomposed over the territory it lands on: land, inland water and ice, the three separately addressable categories `build_polycell_support()` carries. Deposition to freshwater is a real flux on the eutrophication pathway rather than a rounding error, so the three are reported side by side and the consumer chooses. A support with no category columns cannot be decomposed and carries the single `"territory"` category, which says the row is undecomposed rather than claiming it is land.

Usage

```
build_n_deposition(
  years = NULL,
  data = list(),
  split = c("auto", "polity_area_ha", "polity_frac"),
  categories = c("auto", "land_water_ice", "none"),
  polity_validity = c("keep", "flag", "drop"),
  example = FALSE
)
```

Arguments

`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year the inputs cover.

`data` Optional named list of pre-loaded inputs: `nhx` and `noy` (each `lon`, `lat`, `year`, `value_g` and optionally `method_deposition`, falling back to `read_n_deposition()` when absent) and `cell_polity` (`lon`, `lat`, `area_code`, `cell_area_ha` and the `split` key column, required). Neither species is optional: a `nhx` or `noy` that contributes nothing to any cell the support carries – no rows, identically zero, or on a longitude convention of its own – is refused with a `whep_absent_input` error rather than halving every deposition total in silence (#1034).

`split` Which polity share splits the cell's deposited mass: `"auto"` (default) takes `polity_area_ha` when the support carries it and `polity_frac` otherwise, `"polity_area_ha"` and `"polity_frac"` demand that key and abort when it is absent. The resolved key is recorded in the `method_polity_split` output column, so a table's split is readable from the table.

`categories` How each polity's share is decomposed over the territory it lands on: `"auto"` (default) decomposes when the support carries `land_area_ha`, `inland_water_ha` and `ice_area_ha` and emits the single `"territory"` category otherwise, `"land_water_ice"` demands those columns and aborts when they are absent, `"none"` keeps one undecomposed row per polycell. The resolved choice is recorded in `method_area_split`.

`polity_validity` What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`example` If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `area_code`, `year`, `area_category`, `deposition_kgn_ha`, `deposition_n_t`, `method_deposition`, `method_polity_split` and `method_area_split`, plus the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`.

`area_category` is `"land"`, `"inland_water"` or `"ice"` under `"land_water_ice"`, and `"territory"` under `"none"`. Summing `deposition_n_t` over the categories of a polycell recovers that polycell's whole share, so an unfiltered sum over the table is still the source mass; a consumer wanting one category **must filter**.

`deposition_kgn_ha` is the whole-cell mean rate: the cell's total mass over its whole area, so every polity of a cell carries the same rate on every category row and the rate is **not** conserved on re-aggregation. Only `deposition_n_t` is a mass.

`method_deposition` names the product the cell's deposition came from, **per cell**, and it is read off the supplied field rather than asserted: `read_n_deposition()` tags its own rows `"hani"`, a field injected through `data` keeps whatever tag it carries, and an injected field carrying no tag is recorded as `"supplied"`. A corrected or substituted de-

position field is therefore visible in the output instead of inheriting HaNi's name (#1097): `correct_n_deposition()` stamps the rows it rescales `"hani_emep_trend"`.

Rows are keyed on `area_code`. `build_polycell_support()` keys on `polity_code` and does not derive the reporting vocabulary (DA-23), and `polity_area_crosswalk` folds distinct polities into one `area_code`, so a support table must be converted to one row per cell and `area_code` **before** it is passed here. That conversion is refused rather than performed silently.

Which land definition governs what

Two different land definitions meet in this function and they govern different things, which is the single fact most likely to be misread from the output.

WHEP's territory governs placement. Where a cell's deposited mass goes – which polity receives it, and how much of it lands on land, on inland water or on ice – is decided entirely by `build_polycell_support()`'s `polity_area_ha` and its three categories.

HaNi's own land mask governs the total. The mass being placed is the HaNi block sum, and HaNi is referenced to the whole 5 arcmin cell inside a land-masked domain: its mask is a **third** land definition, measuring 13.5977 Gha against 14.3195 Gha of whole cells and 12.9931 Gha of LUH2 terrestrial. Nothing here re-references the mass to WHEP's land, and that is deliberate. Forming a rate on the whole cell and multiplying it by `land_area_ha` would shed about 9% of the source mass, because the rate's denominator includes ocean while land plus inland water plus ice is territory; re-referencing to HaNi's own mask instead is the cleanest per-hectare rate but moves the global total by about 4.5%.

So a global sum out of this function is HaNi's total, redistributed onto WHEP's territory – not WHEP's land multiplied by a WHEP rate. Conservation is exact against the source (34.77 Tg NHx in 2014), and the reconciliation that is *not* available is `rate * (land + inland_water + ice) ==` which does not hold as an identity and is not asserted anywhere.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See add_polity_code() for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 (area, year) cells – add_polity_code() says so as mapping_status == "backcast_anchor", and polity_coverage_gaps() reports it as gap_kind == "backcast_anchor". These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so reporting_polity_code can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after SUD-1956-2011 ends, and its post-2011 rows carry that code. These columns do not say so: add_polity_code() reports such a row as mapping_status == "out_of_span", and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. polity_coverage_gaps() reports the stand-in rows of a built table, and options(whep.polity_mapping_status = "flag") (or "status") carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_n_deposition(example = TRUE)
```

---

build_n_exceedance_extension

Build the embodied-nitrogen footprint extension.

Description

Selects one nitrogen category from a build_n_boundary_exceedance() country-resolution output into the impact_u column of a build_footprint() extension: "exceedance" (the default) carries exceedance_n_t, "within_boundary" carries within_boundary_n_t, and "production" carries production_n_t (harvested product plus used residue plus grazed forage). The chosen category is stamped in method_n_exceedance.

Signed crop-attributed values are retained. They arise legitimately when a negative crop-surplus contribution receives its approved signed share of a positive cell overshoot. build_sjos_n_footprint() traces positive and negative parts separately through the non-negative footprint engine and recombines them linearly. If the upstream result contains a zero/near-zero-denominator cell residual, this function raises a typed undefined-attribution error: a mandatory crop footprint cannot silently discard or invent an allocation for that residual.

The per-crop (item_cbs_code) granularity is preserved so the footprint can be traced to origin (locked plan decision 14). Rows with a missing key are dropped defensively and the excluded impact is reported with a warning, never dropped silently; zero-impact crops are kept because they still consume trade.

Usage

```
build_n_exceedance_extension(
  exceedance,
  category = c("exceedance", "within_boundary", "production")
)
```

Arguments

exceedance	A build_n_boundary_exceedance() output at resolution = "country", keyed by year, area_code, item_cbs_code with the mass terms exceedance_n_t, within_boundary_n_t, actual_n_t and, for category = "production", production_n_t.
category	Which nitrogen mass to carry into impact_u: "exceedance" (default), "within_boundary" or "production". Validated with rlang::arg_match().

Value

A tibble with the build_footprint() extension contract columns year, area_code, item_cbs_code, impact_u (tonnes N) and method_n_exceedance (the chosen category).

Examples

```
build_n_exceedance_extension(
  tibble::tribble(
    ~year,
    ~area_code,
    ~item_cbs_code,
    ~exceedance_n_t,
    ~within_boundary_n_t,
    ~actual_n_t,
    2010L, 10L, 2511L, 5, 3, 8,
    2010L, 10L, 2513L, 0, 4, 4
  ),
  category = "exceedance"
)
```

---

build_n_inputs *Assemble gridded nitrogen inputs from every WHEP N-input source.*

Description

Combines biological nitrogen fixation (calculate_bnf()), residue/root N recycling (calculate_npp_carbon_nitr livestock manure (build_livestock_nutrient_flows()), atmospheric deposition (build_n_deposition()), human-population N (build_human_n()), soil organic-matter mineralization (build_carbon_balance()'s son_change_kgn_ha) and synthetic fertiliser (a country total spatialized to crops and cells via spatialize_country_n_to_crops()) into one long-format tibble of nitrogen inputs to agricultural land.

fert_type values: "bnf", "recycling", "manure_solid", "manure_liquid", "excreta", "deposition", "human", "som_mineralization", "synthetic" and "accum_loss". The last is a documented gap (perennial-crop standing-biomass N accumulation from the source implementation's nitrogen balance): its source computation was not available for this task, so it is never emitted, only reserved in the vocabulary.

Terms that are fundamentally per-cell or per-land-use rather than per-crop are allocated over the agricultural land support, either supplied as `data$ag_land_support` or derived by `build_ag_land_support()` from the gridded inputs already present. Deposition uses both cropland and grassland support. `"human"`, `"som_mineralization"`, and manure already assigned upstream to Cropland but lacking a crop use only local cropland support, so manure is not reassigned to grassland after the manure engine's capacity allocation. Forest and natural land are outside that support and therefore outside the agricultural balance. Grassland is represented by CBS 3000; no intensive/extensive class is inferred.

Usage

```
build_n_inputs(
  years = NULL,
  resolution = c("grid", "polity"),
  synthetic_method = NULL,
  method_unsupported = NULL,
  unattributed_method = NULL,
  polity_validity = c("keep", "flag", "drop"),
  data = list(),
  example = FALSE,
  manure_method = NULL
)
```

Arguments

`years` Optional integer vector of calendar years to keep. `NULL` keeps every year the assembled inputs cover.

`resolution` `"grid"` (default, per cell/crop/year/fert_type) or `"polity"` (summed to `area_code/item_cbs_code/year/fert_type`).

`synthetic_method` Synthetic-N crop allocation method, `"coello"` or `"area_share"`. When `NULL` (default), uses `data$synthetic_method %||% "coello"` for backwards compatibility.

`method_unsupported` What happens to non-item nitrogen (deposition, human, soil-organic-matter mineralization, unattributed manure) whose own cell-year carries no allocation support at all. The support is the one `unattributed_method` selects: cropland under the default `"cropland_area"`, cropland plus grassland under `"agricultural_area"`, so the second argument decides both which rows are stranded and where they are rescued to (whep#1191). `"abort"` (the default, and the behaviour before this argument existed) refuses to continue and names the streams and the mass, so a coverage gap cannot be lost silently. `"reallocate"` spreads each such row over its own polity-year's support in proportion to area, which conserves mass and is the same rule `spatialize_country_n_to_crops()` already applies to a polity-crop with no crop-pattern cell; it still aborts when the polity-year has no support anywhere. `"reallocate_drop"` places the same rows the same way and then discards that irreducible remainder, warning with its mass, so a global build is not refused over a few

polities with population and no agricultural land. `"drop"` reallocates nothing and discards every such row. Whichever is chosen is stamped on every output row as `method_unsupported`. May also be supplied as `data$method_unsupported`, which is how `build_nitrogen_balance()` forwards it.

`unattributed_method`

Where nitrogen that reached agricultural land but no single crop goes – manure the engine placed on Cropland or landed by transport without a crop, plus deposition, human N and SOM mineralization, all of which carry `item_cbs_code = NA`. `"cropland_area"` (default) spreads it over the cell-year's cropland support in proportion to each crop's hectares; `"agricultural_area"` spreads it over cropland *and* grassland support (CBS 3000), which is the land the manure engine's own transport sizing already counts as available room; `"exclude"` drops it, the pre-whep#532 behaviour, retained as a measurable sensitivity and not as a fallback – it warns with the excluded tonnage by stream. The allocating methods conserve mass or abort; `"exclude"` does not conserve it. When `NULL` (default), uses `data$unattributed_method %||% "cropland_area"`.

This choice also governs `method_unsupported` (whep#1191): both run on one allocation support. Under `"agricultural_area"` a cell holding only grassland is no longer a stranded cell – its nitrogen lands on its own grassland and `method_unsupported` never sees it – and a `"reallocate"`/`"reallocate_drop"` rescue spreads over the polity-year's cropland *and* grassland rather than its cropland alone. Under `"exclude"` nothing reaches the stranded step, so `method_unsupported` is inert apart from its stamp.

`polity_validity`

What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`data`

Named list of pre-loaded, caller-supplied upstream inputs. Each of the following is required for its corresponding `fert_type` to be emitted (a missing one skips that source rather than erroring, since callers may only want a subset). Supplying one and getting **no** nitrogen back is a different matter and is refused with a `whep_absent_input` error, because a stream that arrived empty is subtracted from the balance without changing anything the mass check can see – that check compares the assembled rows against themselves (#1034). The one exception is `carbon_balance`, which warns instead: its stream keeps only `son_change_kgn_ha > 0`, so contributing nothing is an observation there as well as a symptom.

- `bnf_input`: `calculate_bnf()`'s required input tibble (`lon`, `lat`, `area_code`, `year`, `item_prod_code`, `crop_npp_n_t`, `product_n_t`, `weed_npp_n_t`, `land_use`, `legumes_seeded`, `seeded_cover_crop_share`, `area_ha`).

The "bnf" term's n_input_t is the bnf_t column of calculate_bnf()'s output.

- npp_n_input: calculate_npp_carbon_nitrogen()'s required input tibble (lon, lat, area_code, year, item_prod_code, item_cbs_code, product_dm_t, residue_dm_t, root_dm_t, optionally residue_soil_dm_t). The "recycling" term's n_input_t is root_n_t plus residue_soil_n_t, the residue N left on the field, from calculate_npp_carbon_nitrogen()'s output. When the input does not carry residue_soil_dm_t, calculate_residue_destinie is applied to it first, with residue_destiny_method, residue_destiny_recovery and residue_bedding_fraction (see build_nitrogen_balance(); the same arguments split the residue removal outputs there) and the polity's region_krausmann and region_un_sub taken from area_code when the input does not carry them. An input that already carries residue_soil_dm_t is recycled on the caller's destinies, while build_nitrogen_balance() books its residue removals from residue_destiny_input with the arguments above: the soil-returned and the removed residue add up to the gross residue only when both come from the package's own destiny split.
- recycling_residue_basis: "residue_soil_returned" (default) or "total_residue", which recycles the gross residue N (residue_n_t), removed residue included. Recorded in method_recycling_n.
- livestock_intake: build_livestock_nutrient_flows()'s intake argument (the redistribute_feed() realised-intake contract), plus gridded (its land-surface layer) and resolution/methods (forwarded as-is). The manure terms sum applied_n from the $applied table of build_livestock_nutrient_flows()'s output, keyed by its territory, sub_territory, crop, land_use, manure_type and year; manure_type selects the "excreta", "manure_solid" or "manure_liquid" term. Under manure_method = "faostat" it supplies only the solid:liquid split.
- manure, livestock_spatial: the manure_method = "faostat" inputs, read only under that method. manure is the faostat-emissions-livestock pin, scoped by the caller to the years wanted, as fertilizer is. livestock_spatial is a list of the surfaces build_gridded_livestock() spreads pasture manure over: gridded_pasture, gridded_cropland and manure_pattern (the spatialize-* pins its production callers read), plus an optional species_proxy overriding the mapping's proxies; needed only when cell_polity is supplied.
- nhx, noy, cell_polity: build_n_deposition()'s inputs. From its output the "deposition" term reads deposition_kgn_ha (the whole-cell rate), deposition_n_t and area_category (only to form the in-scope share of each polycell's mass, see deposition_scope) and method_deposition; n_input_t is deposition_kgn_ha times that share times the support's area_ha, divided by 1000.
- deposition_scope: which of a polycell's territory the "deposition" term is credited with. "territory" (default) is land plus inland water plus ice: nitrogen deposited on a lake or a glacier still drives indirect N2O and still reaches the eutrophication pathway, so the

impact terms have to account for it. `"land"` is the terrestrial surface alone, for purposes that want it, and it needs a `cell_polity` carrying `land_area_ha`, `inland_water_ha` and `ice_area_ha`; asking for it without them aborts rather than silently returning the whole territory. Recorded in `method_deposition_scope`.

- `ag_land_support`: agricultural physical land support keyed by `lon`, `lat`, `area_code`, `year`, `item_cbs_code`, with `land_use` (`"cropland"` or `"grassland"`) and positive `area_ha`. Optional: when absent it is derived natively by `build_ag_land_support()` from `cell_polity`, `type_cropland` and `crop_patterns` (plus `states` or `grassland_ha` for the grassland side). Supply it to override that derivation with a better land surface. Cropland rows identify crop CBS items; all pasture/rangeland rows use CBS 3000.
- `grassland_source`, `gridded_pasture`, `grassland_ha`, `states`: forwarded to `build_ag_land_support()` when the support is derived. `grassland_source` selects its `grassland` argument (`"gridded_pasture"` default, `"luh2"`, or `"none"` for cropland-only support).
- `total_population`, `urban_population`, `cropland_ha`, `cell_polity`: `build_human_n()`'s inputs. Which population is read is set by `human_n_population_basis`. The `"human"` term's `n_input_t` is the `human_n_t` column of `build_human_n()`'s output. The term was called `"urban"` before it was renamed.
- `human_n_population_basis`: `build_human_n()`'s `population_basis`, `"total"` (default: `total_population`, from `build_total_population_grid()`, with the per-inhabitant rate) or `"urban"` (`urban_population` with the per-urban-inhabitant rate). Supplying only the other basis's population aborts. Recorded in `method_human_population` and `method_human_kgn_cap`.
- `human_n_method_residual`: `build_human_n()`'s `method_residual` (`"nearest"` default, `"polity"`, `"keep"` or `"drop"`), for human N the transport step leaves on a source cell with no cropland. Only what that rule leaves stranded reaches `method_unsupported`.
- `human_n_method_local_residual`: `build_human_n()`'s `method_local_residual` (`"room_cap"` default or `"uncapped"`), for the residual a source cell WITH cropland gets back: capped at the cell's own room, the excess placed by `human_n_method_residual`, or kept whole.
- `carbon_balance`: `build_carbon_balance()`'s `"grid"`-resolution output (`lon`, `lat`, `area_code`, `land_use`, `year`, `area_ha`, `son_change_kgn_ha`); this driver requires it supplied directly, it is never computed here. The `"som_mineralization"` term keeps the cropland rows with `son_change_kgn_ha > 0` and takes `son_change_kgn_ha` times `area_ha`, divided by 1000, as `n_input_t`.
- `primary_prod`, `fertilizer`, `crop_patterns`, `type_cropland`, `cell_polity`: the synthetic-fertiliser assembly (country total from `fertilizer`, the `faostat-fertilizer-nutrients` pin, split to crops by the chosen crop-share method, then to cells by `crop_patterns`/`type_cropland`). From `fertilizer` it reads the raw FAOSTAT columns `Element` (`"Agricultural Use"`), `Item` (`"Nutrient nitrogen N (total)"`), `Year`, `Area Code` and `Value`; `Value` becomes the country total `synthetic_n_t`, passed

on as the `n_t` column of `spatialize_country_n_to_crops()`'s `country_totals`, whose output `n_t` is the `"synthetic"` term's `n_input_t`.

- `synthetic_method`: how the synthetic-N country total is split across crops, `"coello"` (default; Coello 2025 rate-weighted, FAOSTAT- conserving) or `"area_share"` (harvested-area shares only).
- `coello_rates`: crop-specific synthetic-N rate table shaped like coello_synthetic_n (`year`, `area_code`, `item_cbs_code`, `kg_n_ha`); defaults to `whep::coello_synthetic_n`. Used only when `synthetic_method = "coello"`.
- `method_unsupported`: the `method_unsupported` argument, supplied on `data` instead, so that `build_nitrogen_balance()` can select it by forwarding its own `data` list.
- `gridded`, `resolution`, `methods`: forwarded to `build_livestock_nutrient_flows()`. `resolution` is the manure engine's own axis, not this function's: it defaults to `"subnational"` at `resolution = "grid"` (cell-level nitrogen needs cell-level manure) and to `"national"` otherwise. A value supplied here is always honoured.
- `residue_destiny_input`, `residue_destiny_method`, `residue_bedding_fraction`: the residue-destiny split `build_nitrogen_balance()` documents. The straw it beds is passed to `build_livestock_nutrient_flows()` as `bedding`, one supply per `year x territory` that `add_manure_bedding()` spreads over the housed litter-using streams, so the bedding N the balance books as leaving the field returns in the manure input. Nothing is bedded when the input is absent or the split beds nothing.

`example` If `TRUE`, return a small fixture instead of assembling real data. Defaults to `FALSE`.

`manure_method` Where the three manure terms come from. `"livestock_intake"` (default) is WHEP's manure engine, `build_livestock_nutrient_flows()` over the realised feed intake. `"faostat"` reads them instead from the FAOSTAT livestock-emissions domain (`data$manure`, the `faostat-emissions-livestock` pin, source `"FAO TIER 1"`, kg N): `"manure_solid"` and `"manure_liquid"` from "Manure applied to soils (N content)" for "All Animals", split with the engine's own solid share per country-year and spread to crops by harvested-area share and to cells by `spatialize_country_n_to_crops()`; `"excreta"` from the per-species "Manure left on pasture (N content)", mapped to species groups by `inst/extdata/livestock_mapping.csv` and spread to cells by `build_gridded_livestock()` with that mapping's spatial proxies. These are national statistics spread onto cells, a cross-check on the engine rather than the default. A country-year the engine gives no solid:liquid split for is booked wholly as solid, with a warning. Recorded in `method_manure`. When `NULL` (default), uses `data$manure_method %||% "livestock_intake"`.

Details

`polity_validity` is forwarded to every builder this function calls that offers it – `build_ag_land_support()`, `build_n_deposition()`, `build_human_n()` and `spatialize_country_n_to_crops()` – and then applied to the assembled output, so one choice governs the whole assembly instead of each builder deciding on its own key space (whep#727). Under `"drop"` the support table

loses those rows too, so a non-item input (deposition, human, SOM mineralization) whose own rows were supplied directly and therefore not dropped can find no cropland support left to allocate over; that aborts in the mass check rather than silently losing nitrogen.

Value

A tibble. At `resolution = "grid"`: `lon`, `lat`, `area_code`, `item_cbs_code`, `year`, `fert_type`, `n_input_t`, `method_recycling_n`, `method_synthetic`, `method_deposition`, `method_deposition_scope`, `method_human_population`, `method_human_kgn_cap`, `method_unsupported`, `method_manure`, `excreta_cpp_share`, `method_excreta_species`, `method_unattributed`. At `resolution = "polity"`: `area_code`, `item_cbs_code`, `year`, `fert_type`, `method_recycling_n`, `method_synthetic`, `method_deposition`, `method_deposition_scope`, `method_human_population`, `method_human_kgn_cap`, `method_unsupported`, `method_manure`, `method_unattributed`, `n_input_t` (summed over cells), `excreta_cpp_share` (N-weighted over cells), `method_excreta_species`. `method_manure` records the source of the three manure terms (`"livestock_intake"`, `"faostat"`, or `"faostat_all_solid"` on applied manure booked as solid for want of an engine split) and is `NA` for every other `fert_type`. `excreta_cpp_share` is the share of an `"excreta"` row's N deposited by cattle, poultry and pigs, the animals of the IPCC 2019 EF3PRP,CPP factor (Vol 4, Ch 11, Table 11.1 (Updated), p. 11.13); sheep and other animals (EF3PRP,SO) deposit the rest. `method_excreta_species` records where the share came from (`"engine_livestock_category"`, `"faostat_species_group"`) or why the row has none (`"fallback_label_no_species"` for manure the engine labelled excreta only for want of a manure-type share, `"no_excreta_n"` for a row with no N). Both are `NA` for every other `fert_type`. `method_recycling_n` records which residue basis the `"recycling"` term used: `"residue_soil_returned"` (residue N net of removal for feed, bedding and burning) or `"total_residue"` (the gross residue N, only on request through `data$recycling_residue_basis`); it is `NA` for every other `fert_type`. `method_synthetic` records the synthetic crop-split basis (`"coello"` or `"area_share"`) on `"synthetic"` rows and is `NA` for every other `fert_type`. `method_deposition` records which deposition product the `"deposition"` term's field came from, read off the supplied `nhx`/`noy` by `build_n_deposition()` (`"hani"` for `read_n_deposition()`'s own rows, `"supplied"` for an injected field carrying no tag of its own), so a corrected field stays visible here. `method_deposition_scope` records which of the polycell's territory the `"deposition"` term was credited with (`"territory"` or `"land"`). Both are `NA` for every other `fert_type`. `method_human_population` and `method_human_kgn_cap` record the `"human"` term's population basis and the denominator of its per-capita rate (see `build_human_n()`); both are `NA` for every other `fert_type`. `method_unsupported` records the rule applied to non-item nitrogen with no allocation support in its own cell, and is the same on every row; read it together with `method_unattributed`, which fixed that support. `method_unattributed` records the `unattributed_method` the whole assembly ran under and is stamped on **every** row, not only on the reallocated ones: under `"exclude"` no reallocated row survives to carry it, and a choice that removes nitrogen has to stay readable from the table. Both grains also carry the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which

territory a row belongs to.

- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_n_inputs(example = TRUE)
```

`build_n_loss_drivers` *Build the IPCC 2019 wet/dry drivers of the nitrogen loss cascade.*

Description

Put the IPCC 2019 Tier 1 wet/dry class of `build_cell_moisture_regime()`, and the Atlantic/Mediterranean drivers mapped from it, on every row of a gridded nitrogen balance, as the two driver tables `build_nitrogen_balance()` joins: `n_balance_drivers` (per balance key and loss `fert_type`, read by `calculate_nh3()`, `calculate_soil_n2o()` and `calculate_indirect_n2o_nh3()`) and `n_balance_leaching_drivers` (per balance key, read by `calculate_n_leaching()`). `build_nitrogen_balance()` calls it itself at grid resolution when neither table is supplied.

A row whose cell, polity, commodity balance item and water regime has a class of its own takes it. A row without one takes its cell's class on precipitation alone (no crop, no

irrigation): a row with no crop item, as the soil organic matter sequestration rows, and a row of an item WHEP holds no land of in that water regime, as grass. A cell lacking CRU climate variables takes them from its nearest CRU land cell (see `build_cell_moisture_regime()`). Every row ends with a class.

`climate` is `"ATL"` on wet rows and `"MED"` on dry rows. `irrig_type`, the n2o_efs_disaggregated key of the Atlantic/Mediterranean options, is `"Flooded"` on every rice row (the commodity balance item of the production items classed `Crop_irrig == "Rice"` in items_prod_full) in both climates and both water regimes, `"Med_average"` on other irrigated MED rows, `"Rainfed"` on other rainfed MED rows and `"Tier_1"` on every other ATL row (the ATL table defines only `"Tier_1"` and `"Flooded"`).

Usage

```
build_n_loss_drivers(
  rows = NULL,
  window = c("trailing_30", "annual", "normal"),
  water = c("precip_irrig", "precip"),
  data = NULL,
  config = list(),
  example = FALSE
)
```

Arguments

`rows`	The balance rows to give drivers: `lon`, `lat`, `area_code`, `item_cbs_code` (`NA` on rows without a crop), `year` and `water_regime` (`"rainfed"` or `"irrigated"`), plus optionally `fert_type`, the loss `fert_type` of each row. With `fert_type`, the loss drivers are built for its non-missing `(key, fert_type)` pairs; without it, for every key and the seven loss `fert_types` (`"Excreta"`, `"Liquid"`, `"Solid"`, `"SOM"`, `"Synthetic"`, `"Human"`, `"Recycling"`). The leaching drivers cover every key. `n_input_t`, the N of each row, is needed wherever an irrigated row has no class of its own: such a row must carry no N.
`window`, `water`, `config`	As in `build_cell_moisture_regime()`. The window also sets the precipitation-only class of rows without a crop.
`data`	Optional list of inputs, passed to `build_cell_moisture_regime()` (`climate`, `crop_depth`, `lpjml_years`, `lpjml_grid`, `regime_areas`, `country_grid`), plus `moisture_regime`, a `build_cell_moisture_regime()` result to use instead of building it. Climate is read once and shared.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Details

Which driver columns each loss method reads, and which this function supplies:

Method	Driver columns	Supplied
`nh3`: `"ipcc"`	`fert_type`	yes

nh3: "manner", "manner_default"	MANNER's, see below	no
n2o: "ipcc2019"	moisture_regime	yes
n2o: "ipcc2019_atl_med"	climate	yes
n2o: "aguilera", "ipcc2006"	climate, irrig_type	yes
n2o_indirect: "ipcc2019"	moisture_regime	yes
n2o_indirect: "ipcc2019_atl_med"	climate, irrig_type	yes
leaching: "ipcc_fracleach"	moisture_regime, water_regime	yes
leaching: "meisinger_drainage"	Meisinger's, see below	no

The two methods it does not supply are later work. MANNER reads `manner_fertiliser`, `soil_ph`, `rate_kg_ha`, `rainfall_mm`, `irrigated`, `temp_c`, `temp_c_annual_mean`, `windspeed_ms`, `technique`, `system`, `incorporation_delay_h` and `species` (see `calculate_nh3()`); Meisinger reads `fert_type`, `climate`, `irrig_cat`, `land_use`, `cn_input`, `tillage`, `som_share` and `drainage_mm` (see `calculate_n_leaching()`). `build_nitrogen_balance()` refuses either before assembling any input unless its driver table is supplied.

`method_moisture_regime` is `build_cell_moisture_regime()`'s stamp. A row that took its cell's precipitation-only class carries the extra field `cell_rainfed_class=no_crop_item` (no crop item) or `cell_rainfed_class=no_regime_area` (an item without regime land of the row's water regime), and a row in a cell that took climate variables from its nearest CRU cell the field `nearest_cru_cell=<col>[+<col>...]`, naming them.

Value

A named list of two tibbles:

- `n_balance_drivers`: one row per balance key (`lon`, `lat`, `area_code`, `item_cbs_code`, `year`, `water_regime`) and `fert_type`, with `moisture_regime` (`"wet"` or `"dry"`), `climate` (`"ATL"` or `"MED"`), `irrig_type`, `method_moisture_regime`, `method_climate` (`"from_moisture_regime"`) and `climate_fill_km`.
- `n_balance_leaching_drivers`: one row per balance key, with `moisture_regime`, `method_moisture_regime` and `climate_fill_km`.

Both are join tables keyed on the balance key; the balance they are joined to carries the polity columns.

Errors

- `whep_n_loss_drivers_rows`: `rows` is missing, empty, lacks a key column, or holds a missing coordinate or year, or a `water_regime` other than `"rainfed"` or `"irrigated"`.
- `whep_n_loss_drivers_unclassified`: a row is still without a class (for example a window-mean PET of zero), named by its stamp.
- `whep_n_loss_drivers_irrigated_cell_class`: an irrigated row carrying N, or one whose N is not supplied, has no class of its own and would take its cell's precipitation-only class. The rainfed/irrigated split gives an irrigated row N only where the item holds irrigated land, so such a row means the balance and the moisture regime disagree on the item's regime land; the error names the count and the items carrying most N.
- Every error of `build_cell_moisture_regime()`, including `whep_absent_input` for an absent climate or irrigation layer.

Source

IPCC (2019), 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Volume 4, Chapter 11, Table 11.1, footnote 4 (p. 11.13).

Examples

```
build_n_loss_drivers(example = TRUE)
```

---

build_n_pathway_exceedance

Build the pathway-mode critical-nitrogen boundary exceedance.

Description

Compares a `build_nitrogen_balance()` output's process-based nitrogen losses against the medium-specific critical loads of Schulte-Uebbing et al. (2022), per grid cell and per crop (Mode B). The air medium compares field ammonia emission (`nh3_n_t`, converted to a per-hectare rate) to the critical ammonia emission; the water medium compares nitrate loss (`no3_n_t`) to the tighter (lower) of the critical groundwater leaching and critical surface-water load, recording which sub-medium binds. Each medium is split into an `exceedance` part (above the load) and a `within` part with the shared exceed-share formula, as a per-hectare intensity and a mass, so `exceedance + within == actual` per medium. `binding_boundary` names the medium (`"air"`, `"water"`, `"both"` on an exact positive tie, or `"none"`) with the highest exceedance share. `resolution = "grid"` keeps the full per-crop grid key and per-medium columns; `"polity"` and `"country"` sum the mass terms over cells to `area_code`, `item_cbs_code`, `year`. Climate / nitrous oxide is not a pathway (the archive ships no critical climate-N load); it stays in the balance's `total_gwp_co2e_kg`, and a climate-N pathway is a documented future hook.

Usage

```
build_n_pathway_exceedance(
  balance,
  critical_loads,
  nh3_source = c("soil", "total_agricultural"),
  resolution = c("grid", "polity", "country"),
  data = list(),
  example = FALSE
)
```

Arguments

`balance`	A `build_nitrogen_balance()` output keyed by `lon`, `lat`, `area_code`, `item_cbs_code`, `year`, carrying `area_ha`, `nh3_n_t` and `no3_n_t`.

`critical_loads`
: A named list of `read_critical_n()` outputs (each `lon`, `lat`, `value` in kg N per hectare) with elements `crit_nh3_emission`, `crit_leaching_gw` and `crit_load_sw`. Every positive-area balance cell must have a non-missing value in all three layers; incomplete coverage aborts instead of silently dropping the cell.

`nh3_source`
: Air-pressure scope: `"soil"` (default, field `nh3_n_t` only, consistent with the surplus boundary) or `"total_agricultural"` (also adds manure housing and storage ammonia from `data$manure_mgmt_nh3_n_t`, keyed to the exact balance grid; if absent, aborts rather than mislabelling soil-only pressure as total agricultural).

`resolution`
: Output grain: `"grid"` (default, per crop per cell) or `"polity"` / `"country"` (per crop per country, summing the mass terms).

`data`
: Optional named list of injected inputs. `manure_mgmt_nh3_n_t` (a tibble keyed to the balance grid with a `manure_mgmt_nh3_n_t` column) supplies the housing and storage ammonia for `nh3_source = "total_agricultural"`. Defaults to `list()`.

`example`
: If `TRUE`, return a small fixture instead of computing. Defaults to `FALSE`.

Value

For `resolution = "grid"`, a tibble keyed `lon`, `lat`, `area_code`, `item_cbs_code`, `year` with `area_ha`, the air-medium columns (`critical_air_kgn_ha`, `actual_air_kgn_ha`, `exceed_share_air`, `exceedance_air_kgn_ha`, `within_air_kgn_ha`, `exceedance_air_n_t`, `within_air_n_t`, `actual_air_n_t`), the water-medium columns (the same set with a `water` suffix, plus `critical_gw_kgn_ha`, `critical_sw_kgn_ha` and `binding_water_medium`), `binding_boundary`, and the `nh3_source` / `method_boundary` stamps. For the aggregate resolutions, the grouping key with the summed per-medium mass terms and the same stamps. Every grain also carries the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See add_polity_code() for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 (area, year) cells – add_polity_code() says so as mapping_status == "backcast_anchor", and polity_coverage_gaps() reports it as gap_kind == "backcast_anchor". These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so reporting_polity_code can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after SUD-1956-2011 ends, and its post-2011 rows carry that code. These columns do not say so: add_polity_code() reports such a row as mapping_status == "out_of_span", and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. polity_coverage_gaps() reports the stand-in rows of a built table, and options(whep.polity_mapping_status = "flag") (or "status") carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_n_pathway_exceedance(example = TRUE)
```

---

build_n_percapita *Build country per-capita anthropogenic reactive nitrogen.*

Description

Aggregates a build_n_inputs() long-format nitrogen-input tibble to the total anthropogenic reactive nitrogen entering each country's agricultural land and divides by population, giving the per-capita reactive nitrogen (kg N/cap/yr) that build_n_boundary_percapita() consumes as its n_percapita input. The default "synthetic_bnf" framing (the Campbell / Global framing) sums the synthetic-fertiliser and biological-nitrogen- fixation input terms using synthetic * syn_tot_agri_ratio + BNF, the locked Campbell / Global framing; recycled or internal terms (manure, deposition, human, soil-organic-matter mineralization) are excluded. Any finer grid key (lon, lat, item_cbs_code) is aggregated away to the country total, and country-years without a matching population row are dropped – in a warning naming those areas and the share of anthropogenic nitrogen that leaves with them, since an area with no denominator is absent from the output rather than wrong in it (#543); options(whep.warn_missing_population = FALSE) silences it. The chosen framing is stamped on every row.

Usage

```
build_n_percapita(
  n_inputs,
  population = NULL,
  framing = c("synthetic_bnf"),
  params = NULL,
  example = FALSE
)
```

Arguments

`n_inputs`	A `build_n_inputs()` long-format output with `fert_type`, `n_input_t` and the `year`, `area_code` keys (finer grid keys such as `lon`/`lat`/`item_cbs_code` are summed away).
`population`	A tibble keyed by `year`, `area_code` with `population` (absolute persons). Defaults to `NULL`, which reads `read_population()` at its own default composition over the years of `n_inputs`; supply a table to use any other source (for instance `read_population(population_source = "pin")`).
`framing`	How the total anthropogenic reactive nitrogen is defined. `"synthetic_bnf"` (default) scales the `"synthetic"` term by `syn_tot_agri_ratio` and adds the `"bnf"` term; other framings can be added.
`params`	Boundary parameters, defaulting to n_boundary_params, used here for `syn_tot_agri_ratio`.
`example`	If `TRUE`, return a small fixture instead of computing from `n_inputs`/`population`. Defaults to `FALSE`.

Value

A tibble keyed by `year`, `area_code` with `n_percapita_kg`, the country total anthropogenic reactive nitrogen per capita (kg N/cap/yr), `framing`, the anthropogenic definition it was computed under, and `method_population`, `"read_population"` when the denominator was read by default or `"supplied"` when the caller passed it, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_n_percapita(example = TRUE)
```

`build_nitrogen_balance`

Build the full nitrogen balance: inputs, outputs, losses and NUE.

Description

Assembles `build_n_inputs()`'s long-format nitrogen inputs into the aggregate `N_input_*` sums, adds the output side (crop production, residue use/burning, grazed weeds, soil organic-matter sequestration), runs the nitrogen-loss cascade (`calculate_nh3()`, `calculate_soil_n2o()`, `calculate_n_leaching()`, `calculate_indirect_n2o_nh3()`), closes the balance (`N_input_full - N_output_full`), applies the N-limitation cap on SOM sequestration, and derives nutrient-use-efficiency (NUE) indicators plus a GWP/CO2e indicator for the nitrous-oxide streams. Ported from the balance and indicator equations of an earlier regional historical reconstruction.

Usage

```
build_nitrogen_balance(
  methods = list(nh3 = "manner", n2o = "ipcc2019", n2o_indirect = "ipcc2019", leaching =
    "meisinger_drainage"),
  resolution = c("grid", "polity"),
  polity_validity = c("keep", "flag", "drop"),
  data = list(),
  gwp = c("ar6", "ar5", "ar4"),
  example = FALSE
)
```

Arguments

`methods`	A named list of method choices: `nh3` (forwarded to `calculate_nh3()`, default `"manner"`), `n2o` (forwarded to `calculate_soil_n2o()`, default `"ipcc2019"`, the IPCC 2019 Tier 1 wet/dry factors of Table 11.1 (Updated) p. 11.13; `"ipcc2019_atl_med"`, `"aguilera"` and `"ipcc2006"` are selectable), `n2o_indirect` (forwarded to `calculate_indirect_n2o_nh3()`,

default "ipcc2019", the wet/dry EF4 of Table 11.3 (Updated) p. 11.26, or "ipcc2019_atl_med"), leaching (forwarded to calculate_n_leaching(), default "meisinger_drainage", or "ipcc_fracleach", the wet/dry FracLEACH-(H) of Section 11.2.2.2 p. 11.25 applied to the N added, or "fracleach_surplus", the same fractions on the surplus) and regime, how each grid row is split into a rainfed and an irrigated part (see Details): "yield_split" (default), "area_split" or "none". The IPCC 2019 wet/dry methods read moisture_regime from the driver tables in data.

resolution "grid" (default) or "polity", as in build_n_inputs(); "polity" sums every term (and re-derives every indicator) over cells.

polity_validity
What to do with a row whose (area_code, year) resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). "keep" (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. "flag" keeps them and adds the per-row logical reporting_polity_out_of_span, marking exactly which rows are stand-ins. "drop" removes them. All three warn; only "drop" changes the numbers. See polity_coverage_gaps(), which reports the same rows for an already-built table.

data Named list of pre-loaded upstream inputs. n_inputs (build_n_inputs()'s output) is used directly when supplied, else build_n_inputs() is called with this SAME data list (so its nested readers, e.g. bnf_input/npp_n_input/livestock_intak propagate through). Also required:

- npp_n_input: shared with build_n_inputs()'s "recycling" term (see calculate_npp_carbon_nitrogen()); used here for prod_n_t.
- ag_land_support: the build_ag_land_support() table, supplying area_ha (the per-hectare boundary denominator) on a physical-land basis. Derived natively when absent and derivable, and shared with build_n_inputs() so both see one table; when neither supplied nor derivable, area_ha falls back to the harvested area the NPP input carries.
- residue_destiny_input: calculate_residue_destinies()'s required input (item_prod_code, residue_dm_t, plus whatever the chosen residue_destiny_method needs; the polity's region_krausmann and region_un_sub are taken from area_code when absent), for used_residue_n_t/bedding_re residue_destiny_method selects the method (default "recovery_regional"), residue_destiny_recovery its recovery-rate variant (default "wirsenius", the package default), residue_bedding the bedding rule (default "wirsenius", 12.4% of the recovered cereal straw) and, under residue_bedding = "fraction", residue_bedding_fraction the share of the recovered non-feed residue used as bedding (default 0; see calculate_residue_destinies()). The same arguments split the residue of npp_n_input for the "recycling" input of build_n_inputs(), which returns to the soil only the residue left on the field (recycling_residue_basis, default "residue_soil_returned"; "total_residue" recycles the gross residue and is recorded in method_recycling_n). An npp_n_input that already carries residue_soil_dm_t keeps it:

the recycling then uses the caller's destinies while the removal outputs use `residue_destiny_input` and these arguments, so the soil-returned and removed residue add up to the gross residue only when both come from the package's own destiny split. The bedded straw also comes back as a manure input, through `build_n_inputs()`, read from the same split as `bedding_residue_n_t`. `residue_feed_share` selects that split's `feed_share` (default `"wirsenius"`).

- `regime_areas`, `regime_ratio`, `regime_production`, `regime_shares`: the rainfed/irrigated split's inputs (see Details), each read or built when absent: the rainfed and irrigated hectares per cell and production item (`lon`, `lat`, `area_code`, `item_prod_code`, `year`, `rainfed_ha`, `irrigated_ha`; built with `build_gridded_landuse()`), the `build_regime_yield_ratio()` table (built on the NPP rows; pass the table `build_carbon_inputs()` reads to make both chains apply one ratio), and the `get_primary_production()` table the yield bounds of `split_regime_yield()` are pooled from. `regime_shares` replaces all three with pre-built shares per grid key (`lon`, `lat`, `area_code`, `item_cbs_code`, `year`, `irrigated_area_share`, `irrigated_yield_share`), which carry no production item: the residue terms then follow the production share.
- `livestock_intake`: shared with `build_n_inputs()`'s manure term; its `"grass"` `feed_quality` rows drive `grazed_weeds_n_t`.
- `carbon_balance`: shared with `build_n_inputs()`'s `"som_ mineralization"` term; its `son_change_kgn_ha` drives `som_ sequestration_n_t`.
- `n_balance_drivers` and `n_balance_leaching_drivers`: the loss drivers. At grid resolution each one left `NULL` is built by `build_n_loss_drivers()` on the split balance's own rows (the IPCC 2019 wet/dry class and the ATL/MED drivers mapped from it; see its table of which methods it supplies), and `method_drivers` says so. That needs the rainfed/irrigated split (`methods$regime` other than `"none"`); `leaching = "meisinger_drainage"` and the MANNER `nh3` methods read drivers it does not supply and are refused before any input is assembled unless their table is supplied. At polity resolution both must be supplied.
- `n_loss_driver_inputs`: optional list passed as `data` to `build_n_loss_drivers()` when it runs (`climate`, `crop_depth`, `lpjml_years`, `lpjml_grid`, `regime_areas`, `moisture_regime`); the balance's own `country_grid` is shared with it.
- `n_balance_drivers`: additional per-`(key, fert_type)` driver columns `calculate_nh3()`/`calculate_soil_n2o()`/ `calculate_indirect_n2o_nh3()` need beyond `n_input_t`/`fert_type` (`moisture_regime` for the IPCC 2019 wet/dry methods, `climate` and `irrig_type` for the ATL/MED ones, plus every MANNER driver column when `methods$nh3 == "manner"`), joined by the balance key AND `fert_type` (the Title-case vocabulary of the loss functions: `"Synthetic"`, `"Solid"`, `"Liquid"`, `"Excreta"`, `"Human"`, `"Recycling"`, `"SOM"`). The excreta rows' species mix, `excreta_cpp_share`, comes from `build_n_inputs()`, N-weighted over the rows pooled into each loss row. A second set of balance-key-only driver columns (`moisture_regime`, `fert_type`, `climate`, `irrig_cat`, `tillage`, `som_share`, `cn_input`, `land_use`) for `calculate_n_leaching()` is read from `n_balance_leaching_driv`

which must hold at most one row per balance key (a many-to-one join aborts on duplicate keys rather than fanning the rows out and misaligning `drainage_mm`). Missing required drivers abort inside the called `calculate_*()` function, naming the exact column; for the MANNER `nh3` methods the check runs on `n_balance_drivers` before any input is assembled, because no function in the package supplies those drivers (only `windspeed_ms` has a reader, `read_lpjml_wind()`, not wired in).

- `drainage_mm`: annual drainage (mm) for `calculate_n_leaching()`, as a numeric vector aligned to the balance-key rows, or already present as a `drainage_mm` column via `n_balance_leaching_drivers`. At grid resolution the rainfed/irrigated split doubles the rows, so an aligned vector only fits with `methods$regime = "none"`; otherwise supply drainage as a column of `n_balance_leaching_drivers`.

`gwp` 100-year global warming potential standard for N2O, `"ar6"` (default), `"ar5"` or `"ar4"`, matching `build_crop_soil_n2o_extension()`.

`example` If `TRUE`, return a small fixture instead of assembling real data. Defaults to `FALSE`.

Details

Rainfed/irrigated split. At grid resolution each crop row is split into a rainfed and an irrigated part before the losses are computed, so regime-specific loss drivers can attach. Two shares per row drive it: the irrigated share of the crop's cell area, from the spatialization chain's rainfed/irrigated hectares (`build_gridded_landuse()`), and the irrigated share of its production, from `split_regime_yield()` on the ratio of `build_regime_yield_ratio()`. Synthetic N and production N split by the production share, on the assumption that N availability per unit of yield is the same in both regimes; every other input, the harvested area, grazed weeds and SOM sequestration split by the area share. The residue terms – the residue and root N returned to soil (`"recycling"`) and the used, bedding and burnt residue N – are evaluated per part: for each production item of the row, the parts' yields keep its production (`Y_r = Y / (1 + s (R - 1))`, `Y_i = R Y_r`), and each part carries the item's residue and root N per hectare times the response of `calculate_crop_residues()` and `calculate_crop_roots()` from the item's mean yield to the part's yield, in the part's water regime, so the model's irrigation terms (a lower residue ratio for the crop groups that respond, a lower root:shoot ratio) apply to the irrigated part. On the default chains, whose NPP input is built with the polity's `region_hanpp` and without `n_input_kg_ha`, the item's residue and root per hectare are the model's at its mean yield, so each part's are the model's at the part's own yield; with any other NPP input they are the item's own density scaled by the model's response. Residue and root are affine in yield, so a high-yield hectare carries less than its yield share. Summing the two parts reproduces the unsplit balance for every additive column except the residue terms and the totals that contain them, which sum to the parts' evaluation instead: with the irrigation terms, the residue and root N of irrigated land fall. The bedded straw the manure input brings back is the parts' bedding: when the balance builds its own inputs and beds any straw, each production item's bedding is scaled by its factor over the two parts before it reaches the manure engine, so the manure columns, too, differ from an unsplit run by the bedding the irrigation terms take off, and the bedded N that leaves the field is still the bedded N that returns. A non-zero

residue term at a key the per-item evaluation does not cover means the balance's inputs and the NPP-N table disagree, and aborts. `"area_split"` uses the area share for everything, residue terms included, and `"none"` returns the unsplit balance. Loss-driver tables without a `water_regime` column apply to both parts.

`polity_validity` is forwarded to `build_n_inputs()` – which forwards it in turn to `build_ag_land_support()`, `build_n_deposition()`, `build_human_n()` and `spatialize_country_n_to_crops()` – and then applied to the balance rows themselves, so one choice governs the whole build (whep#727). A `data$n_inputs` table supplied directly is left alone: it was built by its own call with its own choice, and the balance rows derived from it still get this argument's fate at the tail.

Grazing excreta. The `"excreta"` input is the N grazing animals deposit on pasture, range and paddock, and its N2O is in `n2o_direct_n_t`, `n2o_indirect_nh3_n_t` and `total_gwp_co2e_kg`, as the field loss it is. `calculate_livestock_emissions()` prices the same emission and reports it as `manure_n2o_direct_pasture` and `manure_n2o_indirect_pasture`, and that is the chain WHEP's GHG footprint takes it from. So these balance columns are not to be added to livestock manure N2O: subtract the pasture columns from one of the two first (whep#1524).

Value

A tibble keyed by `year`/`area_code`/`item_cbs_code` (plus `lon`/`lat` at `resolution = "grid"`, and `water_regime`, `"rainfed"` or `"irrigated"`, at grid resolution unless `methods$regime = "none"`) with `area_ha` (each crop's harvested hectares in the cell, summed over cells at `resolution = "polity"`; used downstream to convert tonnes N to a per-hectare rate), the input aggregates (`n_input_full_t`, `n_input_full_nosom_t`, `n_input_std_t`, `n_input_som_t`, `n_input_for_n2o_t`), the output aggregates (`n_output_residues_t`, `n_output_som_t`, `n_output_useful_t`, `n_output_std_t`, `n_output_full_t`), the loss terms (`nh3_n_t`, `n2o_direct_n_t`, `no3_n_t`, `denitrification_n_t`, `retained_profile_n_t`, `n2o_indirect_no3_n_t`, `n2o_indirect_nh3_n_t`; `no3_n_t + denitrification_n_t + retained_profile_n_t` is the surplus, see `calculate_n_leaching()`), the (post-cap) `som_sequestration_n_t`, `n_balance_t`, `surplus_t`, `surplus_share`, the five NUE ratios (`nue_std`, `nue_residues`, `nue_som`, `nue_useful`, `nue_full`), `total_gwp_co2e_kg`, and the `method_nh3`/ `method_soil_n2o`/`method_n2o_indirect_nh3`/`method_leaching` provenance columns (the per-row stamps of the loss functions; a balance row pooling loss rows priced differently lists them joined by `"|"`, and a row with no N2O-relevant input, which emits no direct or NH3-indirect N2O, carries the method chosen suffixed `"_no_loss_row"`, e.g. `"ipcc2019_no_loss_row"`), `method_drivers` (`"whep_built"`, `"supplied"` or `"supplied|whep_built"`), and, when the leaching drivers carry them (as built ones do), the row's wet/dry `moisture_regime`, how it was obtained (`method_moisture_regime`) and `climate_fill_km`, `method_regime` and `method_regime_split` when the split ran (`"yield_ratio"`; `"area_no_production"` where the crop's production could not be expressed in fresh weight and the area share was used for everything; `"regime_shares"` for shares supplied as `data$regime_shares`; or `"no_regime_share"` for a row booked wholly rainfed because the regime layer does not cover it), and `method_residue_regime`, how the row's residue terms were split, the rules of its production items joined by `"|"`: `"part_yield"` (re-evaluated at a rainfed part's yield), `"part_yield_irrigation_terms"` (re-evaluated at an irrigated part's yield, with the model's irrigation terms), `"crop_density"` (the item's per-hectare values on the part's area: no part yield, or `"area_split"`) or `"yield_share"` (the production share, for shares supplied without production items), `NA` on a row with no residue term, plus the polity columns below. When the supplied `n_inputs` carry them, the `method_recycling_n`,

method_synthetic, method_deposition, method_deposition_scope, method_human_population, method_human_kgn_cap, method_unsupported, method_manure and method_unattributed stamps from build_n_inputs() are carried through as well, so a balance names the input conventions that produced it. Gains reporting_polity_out_of_span when polity_validity = "flag".

Loss drivers

Which driver columns each loss method reads, and whether build_n_loss_drivers() supplies them. At grid resolution that function builds each driver table left NULL; at polity resolution both tables must be supplied.

Method	Driver columns	Supplied	Status
nh3: "ipcc"	fert_type	yes	built
nh3: "manner", "manner_default"	MANNER's	no	note 1
n2o: "ipcc2019"	moisture_regime	yes	built
n2o: "ipcc2019_atl_med"	climate	yes	built
n2o: "aguilera", "ipcc2006"	climate, irrig_type	yes	built
n2o_indirect: "ipcc2019"	moisture_regime	yes	built
n2o_indirect: "ipcc2019_atl_med"	climate, irrig_type	yes	built
leaching: "ipcc_fracleach", "fracleach_surplus"	moisture_regime, water_regime	yes	built
leaching: "meisinger_drainage"	Meisinger's	no	note 2

- Note 1: MANNER reads manner_fertiliser, soil_ph, rate_kg_ha, rainfall_mm, irrigated, temp_c, temp_c_annual_mean, windspeed_ms, technique, system, incorporation_delay_h and species (see calculate_nh3()). Drivers must be supplied; global drivers planned.
- Note 2: Meisinger reads fert_type, climate, irrig_cat, land_use, cn_input, tillage, som_share and drainage_mm (see calculate_n_leaching()). Drivers must be supplied; global drivers planned.

build_nitrogen_balance() refuses a method of note 1 or note 2 before assembling any input unless its driver table is supplied.

Polity columns

Every area-keyed output carries the polity its area_code resolves to in that row's year:

- polity_area_code: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use reporting_polity_code to say which territory a row belongs to.
- reporting_polity_code: The polity itself, e.g. ESP-1846-1914. It is year-aware, so the same area_code resolves to different polities in different years, which is the point of the crosswalk.
- reporting_polity_name: Its name. It can differ from the area's own name where the area folds into an aggregate.
- reporting_polity_has_geometry: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. FALSE is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_nitrogen_balance(example = TRUE)
```

`build_nourishment_band`

Build the SJOS-N nourishment band.

Description

Returns the floor and ceiling on **mean per-capita protein supply** that bound the adequate band, the prevalence of inadequacy and of excess, and the number of people on each side. It replaces a flat 46 x 1.35 floor and a flat 63 x 1.35 ceiling, of which only the 46 was ever sourced (whep#753).

Both bounds are one inversion of the WHO/FAO/UNU TRS 935 Box 1 lognormal:

$$S_D = \sqrt{S_I^2 + S_R^2}$$

$$\text{bound} = \text{anchor} \cdot \exp(zS_D + S_I^2/2)/(1-\omega)$$

The **floor** anchors on the demographically weighted *average* requirement with $z = \Phi^{-1}(1 -$ shortfall): the supply below which more than `shortfall` of the population falls short. The **ceiling** anchors on `multiple` times the demographically weighted *safe level* with $z = \Phi^{-1}$(share): the supply above which more than `share` of the population exceeds that limit. `exp(z S_D)` carries a median anchor to the required median intake, and `exp(S_I^2/2)` carries a median to a **mean**, which is what makes either bound comparable with a per-capita supply figure. Omitting the second is not conservative — it sets the required mean equal to the required median, about 5% too low at a typical dispersion.

The two tails do not take the same tolerance, and that is the model's finding rather than a preference. `shortfall` defaults to 2.5%, fixed independently by TRS 935 Figure 7 ("Safe population intake ie. risk<2.5%") and by FAO's stated lowest feasible PoU target. Applying 2.5% to the upper tail as well puts the ceiling *below* the floor for 162 of 167 country-years on the 2010 build, because TRS 935 calls intakes below requirement harmful while calling twice the safe level "unlikely to be associated with any risk" (section 14.2).

`share` is WHEP's own criterion, not a sourced value, and it is the only number in the band that is. It defaults to 0.5, which reads "Over" as *the typical member of this population exceeds the limit* — a definition of over-nourishment, not a measurement of harm. Where `shortfall` is fixed independently by TRS 935 Figure 7 and by FAO's PoU target, nothing external fixes `share`: TRS 935 declines to set a tolerable upper intake at all. It is exposed precisely so it can be varied, the chosen value is stamped in `method_ceiling`, and any published use should carry a sensitivity across it. The `people_over` column exists partly for this reason: reported as a continuous headcount, the result does not rest on where the class boundary was drawn.

`multiple` defaults to 2, which TRS 935 section 13.7 names as "twice the recommended intake, previously identified as a safe upper limit ... likely to be safe". The report's own alternative is 3–4x, which it says "approach the tolerable upper limit and cannot be assumed to be risk-free"; both stay selectable and the choice is stamped in `method_ceiling`.

Protein quality divides BOTH bounds, which is algebraically the diet-side correction TRS 935 section 14.1.5 prefers: comparing `supply x q` against a crude bound and comparing crude supply against `bound / q` are the same inequality, and dividing both keeps the published supply series untouched while moving the floor and the ceiling together. Correcting only the floor would leave the ceiling on crude protein and put the two bounds on different bases. Supply `data$quality` from `build_protein_quality()`; without it the band stays on crude protein and `method_quality` says `"none"`, which is a known understatement of 11-36%.

Supplying `data$supply` adds the prevalences and, where a population is available, the headcounts. A country is not uniformly under or over: with a supply inside the band both tails are populated, and `people_under` / `people_over` are the quantities that say by how much.

Usage

```
build_nourishment_band(
  data = list(),
  shortfall = 0.025,
  ceiling = list(multiple = 2, share = 0.5),
  requirement_sd = 0.12
)
```

Arguments

`data`	Named list of injected inputs. `requirement` (`year`, `area_code`, `requirement_g_cap_day`, optionally `population`), `requirement_safe` (the same from `build_protein_requirement()` with `requirement = "safe"`), `dispersion` (`year`, `area_code`, `sigma`) and `loss_wedge` (`year`, `area_code`, `omega`) are required; optional `supply` (`year`, `area_code`, `protein_g_cap_day`, optionally `population`) adds

the prevalence and headcount columns, and optional `quality` (`year`, `area_code`, `quality`) from `build_protein_quality()` divides both bounds.

`shortfall` Tolerated prevalence of inadequacy, in `(0, 1)`. Defaults to `0.025`.

`ceiling` Named list with `multiple` (of the safe level, positive; defaults to `2`) and `share` (tolerated prevalence of excess, in `(0, 1)`; defaults to `0.5`).

`requirement_sd`
Log-scale SD of the requirement, `S_R`. Defaults to `0.12`, TRS 935 p.38 for adults on a per-kilogram basis. The report notes this captures only about a fifth of observed between-individual variance, so it is exposed for sensitivity rather than fixed.

Value

A tibble keyed by `year`, `area_code` with `floor_g_cap_day`, `ceiling_g_cap_day`, `requirement_g_cap_day`, `safe_g_cap_day`, `sigma_intake`, `sigma_deficit`, `omega`, `quality`, `method_quality`, `method_shortfall` and `method_ceiling`; plus `prevalence_protein_deficit`, `prevalence_protein_excess`, `people_under` and `people_over` when `data$supply` is given; plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()`

reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_nourishment_band(
  data = list(
    requirement = tibble::tribble(
      ~year, ~area_code, ~requirement_g_cap_day,
      2010L, 10L,        32
    ),
    requirement_safe = tibble::tribble(
      ~year, ~area_code, ~requirement_g_cap_day,
      2010L, 10L,        40
    ),
    dispersion = tibble::tribble(
      ~year, ~area_code, ~sigma,
      2010L, 10L,        0.26
    ),
    loss_wedge = tibble::tribble(
      ~year, ~area_code, ~omega,
      2010L, 10L,        0.025
    )
  )
)
```

---

build_polycell_land_uses

Partition each polycell into mutually exclusive land uses.

Description

Split every polycell-year's `land_area_ha` into `cropland`, `grassland`, `urban`, `natural` and `unclassified`, so territorial quantities can be attributed to a land class instead of being assumed agricultural or dropped (issue #423).

The **level** of each agricultural class comes from the statistical record, which is authoritative; LUH2 supplies only the **within-country spatial pattern**. The two provenances are recorded in separate columns (`level_source`, `pattern_source`), and their per-polycell difference is emitted as `statistical_pattern_disagreement_ha` rather than being absorbed into the natural class. That column is a transition instrument: its magnitude measures how much LUH2 is still doing, and it is the criterion for retiring LUH2 as a source.

Inland water and ice are not land uses. They stay on the polycell as their own categories and never enter a class.

Usage

```
build_polycell_land_uses(
  years = NULL,
  grassland_level_source = c("faostat_pasture", "luh2"),
  overfull_method = c("spillover", "cap"),
  data = list(),
  example = FALSE
)
```

Arguments

`years`	Integer vector of calendar years to return, or `NULL` (default) for every year present in the support.
`grassland_level_source`	Grassland level basis, `"faostat_pasture"` (default, FAO Land Use item 6655) or `"luh2"`. The default matches the cropland anchor so both agricultural classes rest on one convention; `"luh2"` remains selectable for sensitivity analysis and is recorded in `level_source`. It is never used as a fallback. Note this differs from `build_grassland_land_extension()`'s own default, which is `"luh2"`: item 6655 excludes temporary meadows and pastures, so the two are a difference in what `grassland` means and not only in provenance. The divergence is deliberate and tracked in whep#759; this function always passes its choice explicitly rather than inheriting a default.
`overfull_method`	How to reconcile a polycell whose anchored agricultural area exceeds its land: `"spillover"` (default) or `"cap"`. See Details.
`data`	Named list of pre-loaded inputs bypassing the readers: `polycell_support` (the `read_polycell_support()` table, interval or year grain), `pattern` (the `read_luh2_landuse()` grid table), `natural_split` (per cell, the non-forested share of natural land), `cropland_level` (the `get_arable_permanent_land()` table), `grassland_level` (the `build_grassland_land_extension()` table) and `temporary_meadows`.
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Details

Per polycell-year:

1. Class **shares** come from the LUH2 `fraction` column - LUH2's share of the whole cell, which is identical on every polycell of a border cell and invariant to the reader's `area_basis`. They are applied to the polycell's own measured `land_area_ha`, so the classes tile the polycell's land by construction rather than to a tolerance.
2. FAO counts temporary meadows and pastures (item 6633) inside arable land, while LUH2 books that ground as grassland. That component is therefore spread over the LUH2 **grassland** pattern but still emitted as `cropland`, because FAO's class definition is the one being anchored.

3. Each agricultural class is rescaled so its polycells sum to the statistical national total for that `(area_code, year)`.
4. Where the anchored area exceeds a polycell's land, it is reconciled by `overfull_method`, never renormalised in silence.
5. `natural` takes the remainder of `land_area_ha`.
6. A polycell with no pattern coverage is `unclassified` in full - never `natural`.

Where a statistical level is absent for a class-country-year, the LUH2 pattern level is used and the row is labelled (`level_source = "luh2"`, `area_source = "pattern_only"`, `allocation_status = "level_missing"`). It is never silently filled. `urban` has no statistical source and is always pattern-only.

Value

A tibble with one row per polycell-year-class: `polycell_id`, `lon`, `lat`, `polity_code`, `area_code`, `year`, `land_use`, `area_ha`, `area_source` (`anchored`, `pattern_only`, `residual` or `unclassified`), `level_source`, `pattern_source`, `allocation_status`, `statistical_pattern_disagreement_` (`NA` where no statistical level applies), `unplaceable_statistical_ha`, `method_overfull`, `spillover_max_ring`, `coverage_status`, and the polity reporting columns carried through from the support.

Pre-1961 levels

FAOSTAT land use starts in 1961, so an unextended level would step at that year. Both agricultural classes are therefore backcast the same way: the FAO 1961 level is carried backwards by LUH2's own national trend, matching FAO exactly at the splice. `get_arable_permanent_land()` already does this for cropland; the pasture backcast here mirrors it, reading the same `luh2-areas` input. A country with no FAO 1961 anchor receives no backcast.

A backcast row is labelled `luh2_backcast_cropland` or `luh2_backcast_pasture` in `level_source` and is **excluded** from the statistical-versus-pattern diagnostic, because comparing a backcast with the pattern compares LUH2 with itself.

Reconciling an overfull polycell

A national statistical total spread by the LUH2 pattern can give a polycell more agricultural land than it has, driven by countries where FAO and LUH2 disagree about how much land is permanent pasture. Measured on this function at 2020: 63.50 Mha, 1.33% of the anchored agricultural area, of which Saudi Arabia is 35.10 Mha and Sudan (former) 14.20 Mha. `overfull_method` selects the treatment and is recorded per row in `method_overfull`:

- `"spillover"` (default) places the excess on same-country neighbours, widening the search ring until it is absorbed, taking non-forested natural land (LUH2 `primn`, `secdn`) first and forest (`primf`, `secdf`) only as a fallback. A neighbour can only receive a class it has a row for, so land the pattern classified nowhere is never quietly credited. The ring each hectare reached is reported in `spillover_max_ring`; at 2020 it places 63.45 of the 63.50 Mha across 3,878 receiving polycells, at a median ring of 2 and a maximum of 22, and names the remaining 42,765 ha (two polycell-classes, in a country whose free land holds neither class) in `unplaceable_statistical_ha`.

- `"cap"` caps the agricultural classes at `land_area_ha` pro rata and leaves the whole 63.50 Mha in `unplaceable_statistical_ha`. It is the sensitivity baseline that quantifies what spillover buys.

The methods are alternatives, never fallbacks. If `"spillover"` cannot place a hectare anywhere in the country, it stays in `unplaceable_statistical_ha` with a warning and the row still reads `method_overfull = "spillover"`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_polycell_land_uses(example = TRUE)
```

`build_polycell_support` *Build the polycell spatial support table*

Description

Produce WHEP's canonical spatial support unit, the **polycell**: a 0.5-degree grid cell intersected with a polity over that polity's validity interval. Each row carries the polity's territory in the cell decomposed into three separately addressable categories, `polity_area_ha = land_area_ha + inland_water_ha + ice_area_ha`, so that aggregating polycells to a polity changes no absolute value and no quantity crosses a border it does not belong to.

Areas are geodesic, from `sf::st_area()` on unprojected WGS84 with spherical (`s2`) geometry, matching WHEP's own spherical convention; `cell_area_ha` keeps the package formula so it stays bit-identical to `build_cell_polity()`. Territory is the union of live real polities; land claimed by no live polity is emitted in the `"unassigned"` attribute rather than renormalised away.

The default grain is **interval-keyed**: one row per polycell per interval over which every area is constant, carrying `start_year` and `end_year`. Supply `years` to expand to one row per polycell-year, which is what `expand_polycell_years()` does on demand. No area varies within an interval, so the interval grain is the form to store.

Usage

```
build_polycell_support(
  years = NULL,
  geometries = NULL,
  water = NULL,
  ice = NULL,
  data = list(),
  aggregates = c("exclude", "overlap_layer")
)
```

Arguments

`years`	Optional integer vector of calendar years. `NULL` (default) returns the interval-keyed grain; a vector expands to one row per polycell-year and adds a `year` column.
`geometries`	An `sf` table of polity geometries with at least `polity_code`, `start_year` and `end_year`; defaults to `get_polity_geometries()`. `start_year` is inclusive; `end_year` is **exclusive at a succession** and **inclusive at the open end**, the convention `polities` is documented under, and neither bound is ever parsed out of `polity_code`. The intervals of one polity must partition time: two that overlap are an error rather than a shape the producer reconciles, and abort with class `whep_pcs_overlapping_interval`. Optional `wiki_status`, `polity_type`, `polygon_status` and `area_code` columns are honoured.

`water` Optional per-cell `tibble` of inland water with `lon`, `lat` and `water_frac`, a fraction of the **whole** cell, as `read_glwd_water()` returns it.

`ice` Optional `sf` polygon layer of glaciated area, as `read_glaciated_areas()` returns it, subtracted per polycell by exact geodesic intersection.

`data` Optional named list of auxiliary layers: `luh2` the validation layer (`lon`, `lat`, `terrestrial_ha`, e.g. `read_luh2_terrestrial()`); `crosswalk` the deployed `build_cell_polity()` table; `producer_crosswalk` a freshly built `build_cell_polity_fraction()` table; and `crosswalk_year`, the year whose polycells the crosswalk's present-day geometry describes (default 2015). The two crosswalks are read **only** by the DA-12 footprint reconciliation: no crosswalk column is carried into the output, and no crosswalk row the intersection did not reproduce is appended to it.

`aggregates` What to do with `polity_type == "aggregate"` rows, which cannot join the partition because an aggregate's polygon covers its members'. `"exclude"` (default) drops them, which is what every published polycell table holds. `"overlap_layer"` clips them too and emits them alongside the partition marked `support_role == "overlap"` – see *The aggregate overlap layer* below.

Value

A `tibble` whose columns are a superset of `polycell_id`, `cell_id`, `lon`, `lat`, `polity_code`, `area_code`, `start_year`, `end_year`, `cell_area_ha`, `polity_area_ha`, `land_area_ha`, `inland_water_ha`, `ice_area_ha`, `geometry_source`, `polygon_status`, `split_method`, `coverage_status`, `support_role`, `area_engine`, `luh2_vintage` and `layers_supplied`, plus `year` when `years` is supplied. `layers_supplied` is the provenance stamp described under *Optional layers are stamped, not inferred* below. `support_role` is `"partition"` on every row unless `aggregates = "overlap_layer"` was asked for. `area_engine` is `"s2"` except on the pieces the spherical engine cannot read back, which are measured with `terra::expanse()` rather than dropped. Diagnostics ride as attributes: `"unassigned"` (the validation-layer disagreement, in both directions: `unassigned_land_ha` where the polities claim less than the layer and `over_claimed_land_ha` where they claim more), `"coverage"` (every live polity interval and why it did or did not produce polycells), `"overlap"` (cells holding more territory than the cell, because two polities were handed the same polygon), `"long_edges"` (polity edges the source stores as one long segment along a parallel, which s2 draws as a bulging great circle), `"terra_measured"` (polycells whose area came from `terra`), `"water_excess"` (inland water clamped to the polycell's territory), `"water_unmatched"` (cells the water layer and the polycells do not share), `"footprints"` and `"footprint_diff"` (the deployed crosswalk, the current producer and the polycell footprint, reconciled at `data$crosswalk_year`).

`"overlap"`, `"terra_measured"`, `"water_excess"`, `"water_unmatched"` and `"unassigned"` are **interval-grain**, like the table itself: they carry `start_year` and `end_year`, and one cell contributes a row per interval. Summing them without first filtering to the interval covering the year of interest counts the same cell once per epoch. On the shipped polities that is the difference between 1,343 clamped polycells over all epochs and 94 in 2015.

Every row is a real polycell: `polity_code`, `polycell_id` and the area columns are populated on all of them, so `sum(land_area_ha)` over the output is the land the intersection measured. The DA-13 transition, which padded the table with the crosswalk rows the inter-

section did not reproduce and carried `polity_frac` alongside, ended with C9; the footprint diagnostics below are where that disagreement is now reported.

Identity is `polity_code`, and only `polity_code`. `area_code` rides along as a label and is not a key: a polity can have no reporting area at all, and two polities can share one. A table whose whole purpose is correct territorial attribution cannot be keyed on a code that merges two countries, so this one is not, and no `reporting_polity_code` or `polity_area_code` is derived here. A consumer joining to a reporting-vocabulary output converts at its own boundary, and **that conversion is where the lossy fold happens** – deliberately visible at the consumer rather than hidden in the support. `build_n_deposition()` refuses an unconverted support instead of converting one silently.

The label is the **reporting** `area_code`, the code space `country_areas` and the national nutrient tables are keyed on – not the coarser `polity_area_code` bucket the matrix workflows aggregate on. Writing the bucket here, which this producer did until whep#907, put 206 (Sudan plus South Sudan) and 999 (Rest of World, absorbing Syria, North Macedonia, Eswatini, Palestine, Equatorial Guinea and French Guiana among 43 territories) in a column named `area_code`, and every `area_code`-keyed consumer dropped those areas' whole national total. A polity with no reporting region of its own – Greenland, Western Sahara, most dependencies – still carries its bucket, because that is the only home the reporting vocabulary gives it.

Land definitions in play

Four definitions of "land" are live in this pipeline and they disagree by up to 10%, so a global area only means something next to the definition it was measured on. At 2015:

Definition	Global area
Whole 0.5-degree cells holding any land	14.3195 Gha
HaNi's own land mask	13.5977 Gha
Union of the live polity polygons	13.4267 Gha
LUH2 terrestrial, `(1 - icwtr) * carea`	12.9931 Gha

`polity_area_ha` carries the third row's territory – the polity polygons, decomposed into land, inland water and ice – but **summing it does not give the third row**. The union is unique ground, which is what makes it comparable with the other three; a sum counts shared ground once per claiming polity. At 2015 `sum(polity_area_ha)` is 13.4599 Gha, exceeding the union by the 0.0332 Gha two live polities both claim. Quote the union for a land definition and the sum for attributed territory, and never read the difference between them as a leak. The first row is the convention this table replaces – a per-hectare rate multiplied by `cell_area_ha` – and it over-counts by 11.0%. The fourth is the DA-5 validation layer: its disagreement with the polygons is emitted in the `"unassigned"` attribute and never silently reconciled, and the polygons exceeding it by about 2.2% is what `inland_water_ha + ice_area_ha` has to account for. The second belongs to the deposition source and governs a different quantity – see `build_n_deposition()`, where WHEP's territory decides *placement* while HaNi's mask decides the *total*.

Only the first and fourth rows are constants of the inputs; `inst/scripts/diagnose_polycell_support.R` re-derives both. The polygon row moves with the polity vintage, so read it back off the table in hand rather than quoting it – `inst/scripts/reconcile_polity_areas.R` measures it.

A fifth land mask is present but deliberately absent from the ladder, because nothing is measured on it: the GLWD water layer carries the CRU mask (67,420 cells against LUH2's 64,493 terrestrial), so cells one carries and the other does not are reported in `"water_unmatched"` rather than dropped by an inner join.

What does not vary historically

`ice_area_ha` comes from `ne_10m_glaciated_areas` (see `read_glaciated_areas()`), a coarse **present-day snapshot**, so it is the same number in 1850 as in 2015: a historical run carries today's ice extent, and land that lay under ice in 1850 is credited to `land_area_ha`. This is accepted only because ice is a **reporting category and not a driver** – nothing in the package divides by `ice_area_ha` or drives a flux with it. If ice ever becomes a driver, the source has to be reopened rather than the caveat restated.

Polity geometry is likewise constant within an interval, and the GLWD file carries a single time step. That is why the default grain is interval-keyed: no area column varies by year, so a per-year grain would repeat identical rows about 173 times.

The aggregate overlap layer

An **aggregate** polity – `BLX-1850-1999` Belgium-Luxembourg, `F249-1918-1990` Yemen, the six residual `"Other"` regions – is a reporting bucket's territory, and its polygon covers its members'. It therefore cannot be a row of the partition: two rows claiming the same ground would hand the cell's land out twice, which is exactly what the rasterised cover this table replaced did, halving Belgium's 1961 cropland (whep#800).

Dropping it outright is not free either. FAOSTAT keys its pre-2000 data on those buckets: Belgium (255) and Luxembourg (256) carry no data before 2000, bucket 15 does, and bucket 15's only territory is `BLX-1850-1999`. Measured against `polity_area_crosswalk` over 1850-1961, **ten** of the 460 polities the pre-1962 resolver reaches are aggregates and **ten** reporting buckets have no other territory in at least one year: 15, 151, 237 (1954-1961 only), 249 and 901-906.

So both granularities are kept, and they are kept apart. With `aggregates = "overlap_layer"`:

- every row carries `support_role`, `"partition"` or `"overlap"`;
- `"partition"` rows are exactly what `"exclude"` emits – same polities, same territory, same land, water and ice, split into more intervals only where an aggregate's validity adds a breakpoint to a cell;
- the cell's inland water is apportioned over the **partition's** territory in that cell, so an aggregate receives what its members receive and the members' share is not diluted by the layer covering them;
- every diagnostic that describes the partition – `"overlap"`, `"unassigned"`, `"water_unmatched"`, `"footprints"` – is measured on the partition alone, so admitting the layer cannot make the polygons look like they over-claim the validation layer.

The consumer contract is the other half: `read_polycell_support()` returns the **partition** unless asked otherwise, so no existing consumer can pick up an overlapping row by accident, and a consumer that wants a bucket's own territory asks for `role = "overlap"` (or `"all"`) and states that it is summing a layer that double-counts by construction. Never aggregate across the two roles.

The layer is not a partition of itself either, and that is not a defect to be fixed by a tolerance: `ROW-1850-2025` Rest of World contains the six regional residuals it is the sum of. Built on the 19 live aggregates of `whep::polities` (779 rows, ingest 2026-08-13) it is 12,644 polycells, and at 2015 it puts more territory in a cell than the cell holds in 2,751 cells shared by `ROW` and `REUR`, 92 by `ROW` and `RAFR`, 8 by `ROW` and `ROCE` and 7 by `ROW` and `RLAM` – plus 2 cells where `CODRU-1922-1960` and `EGYSUD-1934-1956` overlap in 1950, which is a polygon disagreement rather than nesting. So a consumer takes **one polity's** polycells out of the layer – the one its bucket resolves to that year – and never sums the layer as a whole. That is also why the `"overlap"` diagnostic keeps measuring the partition only: an over-full cell means something there, and in this layer it means nothing.

Optional layers are stamped, not inferred

`water` and `ice` are optional, and when either is absent its column is filled with zeros. That is correct for a smoke build and wrong for a published pin, and no identity this table carries can tell the two apart: `polity_area_ha == land_area_ha + inland_water_ha + ice_area_ha` holds to `max |residual| = 0 ha` on an all-zero layer, because zero satisfies it. Two published pins were built that way and passed every check – `20260818T105426Z-a0330` (whep#885) and `20260827T190201Z-f82a2` (whep#1010), the second of them two days after the first was closed and with whep#885's warning already in place. On the second, 2015 land came out 534.9 Mha (+4.1%) high and single cells in Lake Victoria went from 2.9 ha of land to 309,083 ha.

So the output carries `layers_supplied`, a label naming which of the two the build actually consumed: `"ice,water"`, `"water"`, `"ice"` or `"none"`. It records what was consumed rather than what was passed, since an empty layer is dropped before use. A label cannot be satisfied by arithmetic, which is the whole point – `read_polycell_support()` refuses a support whose stamp says a layer was missing, and falls back to asserting the columns are not identically zero on a table published before the stamp existed.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  build_polycell_support(
    years = 2015L,
    geometries = polycell_example_geometries()
  )
}
```

`build_primary_prices` *Build primary item prices*

Description

Compute prices for primary production items. Export trade prices are preferred; when unavailable, production value prices (gross production value divided by quantity) are used as fallback. Gaps are filled via linear interpolation.

Usage

```
build_primary_prices(
  primary_prod,
  value_of_production = NULL,
  trade_prices = NULL,
  example = FALSE
)
```

Arguments

`primary_prod`	A tibble of primary production, as returned by `build_primary_production()` or `get_primary_production()`.
`value_of_production`	A data frame with FAOSTAT Value of Production data. Must contain columns `Item.Code` (or `item_code_prod`), `Element`, `Unit`, `Year`, `Value`, `Area.Code` (or `area_code`). If `NULL`, only trade prices are used.
`trade_prices`	A tibble as returned by `build_trade_prices()`. If `NULL`, it is computed internally.
`example`	Logical. If `TRUE`, return a small example tibble. Default `FALSE`.

Value

A tibble with columns:

- `year`: Integer year.
- `item_prod_code`: Production item code (character).
- `price`: Price in KDollars per tonne.

Examples

```
build_primary_prices(example = TRUE)
```

`build_primary_production` *Build primary production dataset*

Description

Construct the full primary production dataset from raw FAOSTAT inputs. This is a convenience wrapper that chains the pipeline steps:

1. `.read_production()` — read & reformat FAOSTAT data.
2. `.fix_production()` — apply Global-ported corrections.
3. `.dedup_production()` — keep one value per key across sources.
4. `.qc_production()` — flag data-quality anomalies on the surviving (deduplicated) values.

Usage

```
build_primary_production(
  start_year = 1850,
  end_year = 2023,
  smooth_carry_forward = FALSE,
  example = FALSE,
  show_duplicates = FALSE,
  historical_data = NULL,
  federation_land = c("none", "successor_union"),
  land_method = c("present_day", "historical_polity"),
  fodder_split = c("fao_mix", "equal"),
  .raw_data = NULL
)
```

Arguments

`start_year` Integer. First year to include. Default `1850`.

`end_year` Integer. Last year to include. Default `2023`.

`smooth_carry_forward`
Logical. If `TRUE`, carry-forward tails are replaced with a linear trend. Default `FALSE`.

`example` Logical. If `TRUE`, return a small hardcoded example tibble instead of reading remote data. Default `FALSE`.

`show_duplicates`
Logical. If `TRUE`, return only the rows that have competing sources in wide format (one column per source) for diagnostic comparison. Default `FALSE`.

`historical_data`
Optional harmonized historical production rows to add before the LUH2 historical extension. May be a data frame or a path to a parquet/csv file. Required semantic columns are `year`, `item_prod_code`, `unit`, `value`, and one of `area_code` or `polity_area_code`. Names such as `item_prod_name`, `item_cbs_name`, and `source` are used when present; WHEP item and area tables fill canonical names where possible. Observed historical rows are retained, and LUH2 proxy filling can use them as anchors. **Rice supplied here is assumed to be on a paddy (rough-rice) basis** and is multiplied by the paddy-to-milled extraction rate, because WHEP's rice item is milled equivalent throughout; pre-divide by that rate if the series is already milled. Default `NULL`.

`federation_land`
Character. How the pre-1962 LUH2 back-cast reaches an area whose territory is a dissolved federation. LUH2 land use is keyed on present-day ISO3, so 15 Belgium-Luxembourg, 51 Czechoslovakia, 228 USSR and 248 Yugoslav SFR have no land record of their own.

- `"none"` (default, current published behaviour) leaves them unmatched; their pre-1962 production is not back-cast at all and the build warns.

- `"successor_union"` rebuilds each federation's land series as the sum of its successor states' LUH2 land, resolved from the `successor` relation published in polities. This back-casts 14.3% more of the 1961-62 production tonnage and therefore moves published pre-1962 values. Under `land_method = "historical_polity"` it reaches only the federations the historical table has no polygon for, and their rows keep the `LUH2_*` source label, not `LUH2_polity_*`.

`land_method` Character. Which borders the pre-1962 `ha` half of `tonnes = ha * t_ha` is measured on. The yield half is historical either way and is untouched by this argument.

- `"present_day"` (default, current published behaviour) reads the `luh2-areas` pin, which is LUH2 land pre-aggregated to present-day ISO3, so a row labelled with the 1961 entity is measured on the borders that entity has today.
- `"historical_polity"` measures it with `build_historical_land_areas()`: gridded LUH2 summed inside the polygon of the polity `area_code` resolved to in that year. It moves published pre-1962 values, needs `sf` and `terra`, and reads gridded LUH2 for every back-cast year, so it is minutes of extra work.

`fodder_split` Character. How an EU AgriDB fodder area is divided among the FAOSTAT items that share its Eurostat label (seven items share "Other plants harvested green from arable land", four "Other root crops n.e.c."). Either way the items together get exactly the reported area; until whep#654 each item could get all of it.

- `"fao_mix"` (default) splits it in proportion to FAOSTAT's own item areas in that year, or in the nearest years that report them. A label FAOSTAT never gives an item area for in that country is split evenly.
- `"equal"` gives every item sharing the label the same share.

`.raw_data` Optional tibble with the same structure as the output of the internal `.read_production()` step. When supplied, the remote-data read is skipped entirely and the pipeline starts from `.fix_production()`. Columns required: `year`, `area`, `area_code`, `item_prod`, `item_prod_code`, `item_cbs`, `item_cbs_code`, `live_anim`, `live_anim_code`, `unit`, `value`, `source`. `fao_flag` is used when present and completed as `NA` when it is not. Default `NULL`.

Value

A tibble with the columns of `get_primary_production()` plus `fao_flag`: `year`, legacy numeric `area_code`, numeric `polity_area_code`, `reporting_polity_code`, `reporting_polity_name`, `reporting_polity_has_geometry`, `item_prod_code`, `item_cbs_code`, `live_anim_code`, `unit`, `value`, `source`, and `fao_flag`. Item names can be recovered via `add_item_prod_name()` and related helpers. When `show_duplicates = TRUE`, returns a wide tibble with one column per source showing the competing values.

`fao_flag` is FAOSTAT's own observation-status code for the value (`"A"` official, `"E"` estimated, `"I"` imputed, `"M"`, `"X"`), and it describes the number in `value` rather than the row's item or area. It is `NA` wherever the number is not one FAOSTAT published under a flag, which is most rows that are not `unit == "tonnes"` or `unit == "ha"`:

- WHEP's own yields (`t_ha`, `t_LU`, `t_head`) are ratios it computes, so FAOSTAT's separate Yield flag is not a statement about them;
- `LU` and `heads` are livestock-unit conversions summed over an animal's products, and `slaughtered_heads` is one FAOSTAT count split across CBS items by shares;
- a gap-filled, back-cast, imputed or reconstructed value (`fill_linear`, `imputed_yield`, `LUH2_*`, `EuropeAgriDB`, `DM_yield_estimate`, `Estimated`) is WHEP's estimate, not a reported figure – and a `FAOSTAT_prod` row can still be one of these, because `source` is resolved per key while the flag is resolved per quantity;
- a value summed or averaged from parts whose flags disagree is dropped rather than credited to one of them (whep#581).

Measured on a real 2010-2013 build: 45.5% of rows carry a flag, 91.7% of `tonnes` rows and 88.2% of `ha` rows.

Green fodder area is reconstructed, and `source` says how. FAOSTAT reports fodder tonnage only (`faostat-production-old`, to 2012); EU AgriDB reports area and yield (to 2019), and its rows read `"EuropeAgriDB"`. A fodder area held flat past its series' last observation reads `"DM_yield_estimate_carried_forward"`, one held flat before its first `"DM_yield_estimate_carried_backward"`, and one interpolated between two observations `"DM_yield_estimate"`. From 2013 every non-EU fodder area is its 2012 value carried forward: measured on the real inputs, 75.0 of the 100.0 Mha of green fodder area in 2013, against 2.2 of 99.5 Mha in 2012. A fodder series therefore changes what it is made of at 2012/2013 with no change in level (whep#1027). That is why `check_series_jumps()` finds nothing at 2013 in fodder area per country (no area of 100 is flagged), and why it finds the break at once in the yearly share of fodder area whose `source` is a carried one (0.022 to 0.750, a ratio of 33.5).

Examples

```
build_primary_production(example = TRUE)
```

`build_processing_coefs` *Build processing coefficients*

Description

Extract the final calibrated processing coefficients from the CBS building pipeline. These can be used independently for footprint calculations.

Usage

```
build_processing_coefs(
  cbs,
  start_year = 1850,
  end_year = 2023,
  example = FALSE
)
```

Arguments

`cbs`	A tibble of final CBS in long format (one row per `element`), as returned by `build_commodity_balances()`. The legacy wide format, one column per element as returned by `get_wide_cbs()`, is still accepted.
`start_year`	Integer. First year to include. Default `1850`.
`end_year`	Integer. Last year to include. Default `2023`.
`example`	Logical. If `TRUE`, return a small hardcoded dataset for illustration without downloading data. Default `FALSE`.

Value

A tibble with columns: `year`, `area_code`, `item_cbs_code_to_process`, `value_to_process`, `item_cbs_code_processed`, `initial_conversion_factor`, `initial_value_processed`, `conversion_factor_sca` `final_conversion_factor`, `final_value_processed`.

Examples

```
build_processing_coefs(example = TRUE)
```

`build_protein_quality` *Build the protein-quality correction for the nourishment band.*

Description

Returns the diet's protein quality per country and year, the factor the SJOS-N band divides by. WHO/FAO/UNU TRS 935 issues its safe level of protein intake "for proteins with a protein digestibility-corrected amino acid score value of 1.0" (section 14.2), and no real diet reaches that, so an uncorrected band is too low for every country.

`method = "trs935_item"` (default) is **tier 1a**: it uses the measured true digestibility TRS 935 Table 5 publishes for each commodity, and falls back to the tier 1b class rate for the items the report does not measure. Table 5 has 35 rows and prints **no fruit, vegetable, root, tuber or sugar** entry at all, so the fallback is not a corner case — on the 2010 world basket the measured share is **84.5%** of food protein and the rest takes the class rate. `protein_measured_share` reports it per row.

`"digestibility_share"` is **tier 1b**: the protein-weighted mean of **0.95 for animal protein and 0.80 for plant protein**, which is how TRS 935 Table 43 footnote b computes it. The animal/plant split follows FAO's own Food Balance Sheet grouping — Animal Products (item 2941) against Vegetal Products (2903) — so it reconciles against FAOSTAT's published aggregates rather than being WHEP's opinion. `"none"` returns a quality of 1 and leaves the band on crude protein.

Both are a provable lower bound on the full correction, because PDCAAS is `min(1, AAS) x D`, which never exceeds `D`. They are conservative about the **size of the correction**, not about nourishment adequacy — they under-correct, and so classify fewer countries as deficient than the full amino acid score would. `build_protein_score()` is that full score, tier 2; it

is code-complete and validated but needs a composition table WHEP does not have, and it arrives as a new method rather than silently changing this one.

`variant` brackets the one judgement tier 1a makes. Table 5 prints several forms of the same commodity and CBS cannot say which was eaten: wheat whole 0.86, cereal 0.77, flour white 0.96; maize 0.85, corn whole 0.87, corn cereal 0.70; rice polished 0.88, cereal 0.75. **The processing direction is not uniform** — refining raises wheat by removing bran and lowers maize, rice and oats through extrusion and Maillard damage — so there is no single axis to sweep and the bracket is carried per item. `"default"` takes the least-processed form, which is the consistent partner for WHEP's own whole-commodity agronomic nitrogen; `"low"` and `"high"` give the span.

Usage

```
build_protein_quality(
  data = list(),
  method = c("trs935_item", "digestibility_share", "none"),
  variant = c("default", "low", "high")
)
```

Arguments

`data`	Named list of injected inputs. `protein_supply` (`year`, `area_code`, `item_cbs_code`, `protein_t`) is required; `protein_digestibility`, `protein_digestibility_items` and `protein_digestibility_trs935` override the packaged tables.
`method`	`"trs935_item"` (default), `"digestibility_share"` or `"none"`.
`variant`	Which Table 5 row each item takes, for `"trs935_item"`: `"default"` (the least-processed form the report names for the commodity), or `"low"` / `"high"`, the plausible bracket. Ignored by the other methods.

Value

A tibble keyed by `year`, `area_code` with `quality`, `animal_protein_share`, `protein_classified_share`, `protein_measured_share` (the share carrying a measured Table 5 value rather than the class rate) and `method_quality`, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.

- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_protein_quality(
  data = list(
    protein_supply = tibble::tribble(
      ~year, ~area_code, ~item_cbs_code, ~protein_t,
      2010L, 10L,        2731L,          40,
      2010L, 10L,        2511L,          60
    )
  )
)
```

`build_protein_requirement`

Build the population protein requirement.

Description

Weights the WHO/FAO/UNU TRS 935 per-class protein requirements by a population's age and sex structure, giving the mean requirement of an average member of that population in grams of protein per day.

The default `requirement = "average"` uses the class AVERAGE (median) requirement, which is the anchor TRS 935 names for population use. The alternative `"safe"` uses the class safe level, the 97.5th percentile of the individual requirement distribution; it is offered for continuity with analyses built on the safe level, but TRS 935 calls that application to

populations incorrect, and it double-counts the requirement margin whenever a dispersion allowance is also applied downstream.

Population rows are supplied as age groups. Each group is expanded to the single years of age it spans, which are assumed uniformly distributed within the group; on UN WPP 2024 data the difference between five-year groups and single-year data is at most 0.105 g/cap/day (0.3%).

It also returns the population's **amino acid scoring pattern**, weighted from TRS 935 Table 50 by the same age structure. Requirement and protein quality are both age-dependent, so a downstream quality score must be taken against the pattern this population actually requires, not against an adult pattern; scoring separably costs roughly 1.5% in the youngest populations and 0.4% in the oldest, always in the same direction. Note the two outputs are weighted by **different** quantities: the requirement by headcount, the pattern by headcount times protein requirement, because a pattern is a composition per gram of protein rather than an amount.

The age-weighted pattern is WHEP's own construction. It follows from the anchor — TRS 935's requirement is defined against a PDCAAS of 1.0 on its own pattern — but no published study scores a national diet against a demographically weighted pattern, so it should be reported as a WHEP method, not as standard practice.

Usage

```
build_protein_requirement(data = list(), requirement = c("average", "safe"))
```

Arguments

`data`	Named list of injected inputs. `population_age` is required: `year`, `area_code`, `age_start`, `age_span`, `sex` (`"m"` / `"f"`) and `population`. `protein_requirement` and `protein_scoring_pattern` override the packaged coefficient tables.
`requirement`	Which TRS 935 column to weight: `"average"` (default, the class average requirement) or `"safe"` (the class safe level).

Value

A tibble keyed by `year`, `area_code` with `requirement_g_cap_day`, `population`, `method_requirement`, the scoring pattern columns `lysine_mg_g`, `saa_mg_g`, `threonine_mg_g` and `tryptophan_mg_g`, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.

- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_protein_requirement(
  data = list(
    population_age = tibble::tribble(
      ~year, ~area_code, ~age_start, ~age_span, ~sex, ~population,
      2010L, 10L,        0L,         5L,        "m",  1000,
      2010L, 10L,        0L,         5L,        "f",  1000,
      2010L, 10L,        20L,        5L,        "m",  3000,
      2010L, 10L,        20L,        5L,        "f",  3000
    )
  )
)
```

---

build_protein_score *Score a diet's protein against the age-weighted requirement pattern.*

Description

Computes the aggregate protein digestibility-corrected amino acid score (PDCAAS) of a diet, per country and year, following the worked example in WHO/FAO/UNU TRS 935 Table 6 (printed p.100). This is **tier 2** of the protein-quality ladder and the standard the

ladder targets; build_protein_quality() is the tier 1b lower bound that ships when no amino acid composition is available.

Given per-item protein supply P_j, digestibility d_j and amino acid content aa_ij in mg per g protein:

$$P_j^d = P_j d_j \qquad D = \sum_j P_j^d / \sum_j P_j$$

$$A_i = \sum_j P_j^d \, aa_{ij} / \sum_j P_j^d \qquad \text{AAS} = \min_i (A_i / \text{ref}_i)$$

$$q = \min(1, \text{AAS}) \cdot D$$

The amino acid profile is weighted by **digestible** protein, not crude, and the score is truncated **before** multiplying by digestibility, which is the TRS 935 convention rather than FNP 92's. Both choices are load-bearing and are explained at the top of the source file.

Averaging per-item scores is not an approximation of this, it is a different and biased quantity. FAO forbids it twice in words, and because min() is concave the average of item scores is a rigorous lower bound on diet quality and so a rigorous upper bound on the floor.

Four amino acids are enough for most diets — TRS 935 p.99: "in calculating scores it is usually only necessary to use a pattern based on these four amino acids" (lysine, sulfur amino acids, threonine, tryptophan) — and are what both FAO worked examples use. Supplying more is supported: every amino acid present in both amino_acids and pattern is scored.

Usage

```
build_protein_score(data = list())
```

Arguments

data	Named list of injected inputs. protein_supply (year, area_code, item_cbs_code, protein_t), amino_acids (item_cbs_code, amino_acid, mg_per_g_protein) and digestibility (item_cbs_code, digestibility) are required; pattern (amino_acid, mg_per_g_protein) overrides the packaged adult reference pattern.

Value

A tibble keyed by year, area_code with quality, amino_acid_score, digestibility, limiting_amino_acid, protein_scored_share and method_quality, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its area_code resolves to in that row's year:

- polity_area_code: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use reporting_polity_code to say which territory a row belongs to.

- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
# TRS 935 Table 6: wheat, chickpea and milk powder against the adult pattern.
build_protein_score(
  data = list(
    protein_supply = tibble::tribble(
      ~year, ~area_code, ~item_cbs_code, ~protein_t,
      2010L, 10L,        1L,             52.0,
      2010L, 10L,        2L,             22.0,
      2010L, 10L,        3L,             11.9
    ),
    digestibility = tibble::tribble(
      ~item_cbs_code, ~digestibility,
      1L,             0.85,
      2L,             0.80,
      3L,             0.95
    ),
    amino_acids = tibble::tribble(
      ~item_cbs_code, ~amino_acid, ~mg_per_g_protein,
      1L,             "lysine",    25,
      2L,             "lysine",    70,
      3L,             "lysine",    80
    ),
```

```
    pattern = tibble::tribble(
      ~amino_acid, ~mg_per_g_protein,
      "lysine",    45
    )
  )
)
```

`build_regime_yield_ratio` *Build the irrigated:rainfed yield ratio per cell, crop and year.*

Description

Gives each cell-crop-year the ratio `R` of its irrigated yield to its rainfed yield, the weight that splits a crop's production, synthetic nitrogen and harvest removals between its two regimes (issue #1233). `R` combines three sources:

1. **Anchor.** The SPAM2010 v2.0 ratio of irrigated to all-rainfed yield for the item's SPAM crop in the country (see `read_spam_yields()` and regime_yield_crop_mapping). Each yield is the country's production over its harvested area, summed over its SPAM cells, so micro-stands do not weigh more than large ones. The anchor is floored at 1.
2. **Level.** The anchor is carried to year `t` by the country's synthetic nitrogen per hectare of cropland relative to 2010, `R_level = 1 + (R_anchor - 1) * n_t / n_2010`, so the gap closes towards 1 before synthetic fertiliser and widens with it.
3. **Anomaly**, in two parts. The **spatial** part is the cell's LPJmL irrigated:rainfed ratio over 1994-2023 divided by the country's, so drier places get a larger gap. The **temporal** part is the cell's ratio in year `t` divided by its own 1994-2023 ratio, so worse years get a larger gap.

The anomalies scale only the excess of the ratio over 1, so `R` goes to 1 wherever the level does, whatever the anomaly. The long-term ratio is `R_lt = 1 + min(9, (R_level - 1) * spatial)`: it is capped at 10, i.e. its excess over 1 at 9. The ratio of the year is `R = 1 + (R_lt - 1) * temporal`; the temporal part is not capped, so only a bad year takes `R` past 10.

This `R` is returned as `ratio_unbounded`, for diagnosis only: a single-year near-zero LPJmL rainfed yield can make it absurdly large. Only the `ratio` of `split_regime_yield()`, which applies it to a cell's production and areas under an irrigated-yield ceiling and a rainfed-yield floor, is fit to weight anything.

Usage

```
build_regime_yield_ratio(cells, run_dir = NULL, data = NULL, example = FALSE)
```

Arguments

`cells`	A tibble of the cell-crop-years to build, with `lon`, `lat` (0.5-degree cell centres), `area_code` (a WHEP area code; it is resolved to its polity bucket for the national inputs), `item_prod_code` and `year`. Other columns are dropped; duplicated keys are collapsed. The country's LPJmL 1994-2023 ratio pools the cells of `cells` with that `area_code` (any crop), so pass every cell of a country, as the gridded land use holds them, or the spatial part is measured against a partial country.
`run_dir`	LPJmL run directory for the anomaly. `NULL` (default) uses `WHEP_LPJML_RUN_DIR`, as `read_lpjml_regime_yield()` does.
`data`	Optional named list of inputs used instead of reading them, chiefly for testing and for reusing work across calls: • `spam`: `read_spam_yields()` output (SPAM2010). • `fertilizer`: the `faostat-fertilizer-nutrients` table. • `cropland`: `get_arable_permanent_land()` output (`area_code`, `year`, `cropland_ha`). • `luh2`: the `luh2-areas` table (`ISO3`, `Year`, `Land_Use`, `Area_Mha`), for the successors' back-cast cropland. • `faostat_production`: the raw `faostat-production` table (`Area Code`, `Item Code`, `Element`, `Year`, `Value`), for the Linum and Hemp dominance. • `lpjml`: LPJmL crop yields for the years of `cells`, at the crop grain (`lon`, `lat`, `year`, `lpjml_crop`, `yield_rainfed`, `yield_irrigated`, `method_regime_yield`). • `lpjml_window`: the same for the normaliser years, with the stand fractions `stand_frac_rainfed`, `stand_frac_irrigated`. Only rows in 1994-2023 are used. • `lpjml_grid`: the cells (`lon`, `lat`) the LPJmL run's output covers. • `cache`: an environment (`new.env()`) that keeps what the inputs that do not depend on the cells are reduced to: the SPAM2010 country totals, the synthetic nitrogen series, the LUH2 cropland, the Linum and Hemp production, and the LPJmL 1994-2023 stand sums and grid. The first call that needs one reads it and stores it, and a later call with the same `cache` reuses it instead of reading it again, so a build that calls this function once per year reads each of them once. Every call must share `run_dir` and the `spam`, `fertilizer`, `luh2`, `faostat_production`, `lpjml_window` and `lpjml_grid` members, because the cache is not keyed by them. `cropland` and `lpjml` depend on the years of `cells` and are not kept, and neither is the country LPJmL normaliser, which pools the cells of each call: the ratio of a call is the same with or without a `cache`.
`example`	If `TRUE`, return a small fixture instead of building the ratio. Defaults to `FALSE`.

Details

Four implementation choices:

- The 1994-2023 normalisers pool every stand, weighted by stand area (stand fraction times the cell's geometric area; the land fraction is not applied), rather than averaging cell ratios.
- Linum and Hemp dominance pools each country's 1961-2023 production; a tie goes to the fibre.
- The floor at 1 applies to the finished composite, not to its members.
- The yield bounds of `split_regime_yield()` pool every production row of 1961-2023 with positive tonnes and area, whatever its `source`.

The USSR's and Czechoslovakia's cropland before 1961 is their successors' combined LUH2 cropland (annual plus perennial crop types), rescaled to the predecessor's own FAOSTAT cropland in 1961 – the same splicing rule `get_arable_permanent_land()` applies to a single country, which cannot be applied to these two because their codes have no LUH2 country.

Value

A tibble with one row per cell-crop-year of `cells`:

- `lon`, `lat`, `area_code`, `item_prod_code`, `year`: the key.
- `ratio_spam`: the SPAM2010 ratio as computed, before the floor.
- `ratio_anchor`: `max(1, ratio_spam)`.
- `ratio_level`: the anchor carried to `year` by synthetic N.
- `ratio_spatial`: the cell's 1994-2023 LPJmL ratio over the country's.
- `ratio_long_term`: `1 + min(9, (ratio_level - 1) * ratio_spatial)`.
- `ratio_temporal`: the cell-year LPJmL ratio over the cell's 1994-2023 ratio.
- `ratio_anomaly`: `ratio_spatial * ratio_temporal`, for reference.
- `ratio_unbounded`: `1 + (ratio_long_term - 1) * ratio_temporal`; `NA` where the level is. Not fit to weight anything: pass it to `split_regime_yield()`, whose `ratio` is the bounded one.
- `spam_crop_used`: the SPAM crop code(s) the anchor came from (for Linum and Hemp, the product chosen for the country).
- `method_ratio_anchor`: `"spam_country"`, `"spam_global"`, `"spam_composite_country"`, `"spam_composite_global"`, `"spam_dominance_country"`, `"spam_dominance_global"`, `"spam_dominance_world"` or `"spam_none"` (no SPAM ratio at all).
- `method_ratio_trend`: where `n_t` came from: `"faostat"` or `"smil_backcast"` (the country's own), either suffixed `"_predecessor"`, `"_sibling_interval"`, `"_aggregate"` or `"_shared_polity"` (from the polity reporting for it, by the lineage step that found it); or `"pre_synthetic_n"`, `"no_n_reported_pre1961"`, `"n_2010_zero"`, `"no_n_t"`, `"no_cropland"` (a reporting polity was found but its cropland is missing) or `"no_n_2010"`.
- `method_ratio_n_2010`: `"own"`, `"successors"`, `"none"` or `"not_needed"` (before 1913).
- `method_ratio_cropland`: the cropland `n_t` is divided by: `"own"`, `"successors_luh2_backcast"`, `"none"` or `"not_needed"`.
- `method_dominance`: `"dominance_raw_faostat"` for Linum and Hemp, `"not_applicable"` otherwise.

- `method_ratio_spatial`: `"lpjml"`, `"no_lpjml_cell"` or `"no_cell_normal"`.
- `method_ratio_temporal`: `"lpjml"`, `"lpjml_recycled_climate"` (years before 1901), `"no_lpjml_cell"`, `"no_cell_ratio"` or `"no_cell_normal"`.
- `method_regime_yield`: the adjustments applied, joined by `";"`: `"anchor_floor"` (SPAM ratio below 1), `"level_cap"` (long-term ratio above 10 before the cap), or `"none"`.

Plus the polity columns below.

Anchor

How `spam_crop` is read is fixed by `spam_basis` in regime_yield_crop_mapping:

- A single crop, or crops joined by `+` with basis `"direct"` (millet, coffee): harvested area and production are summed over the crops and the ratio is taken of the sums.
- `"composite_weighted"` (the forage crops): the mean of the member crops' ratios weighted by each member's SPAM harvested area (irrigated plus rainfed) in the country. A member with no irrigated or no rainfed yield there is dropped and the weights of the others renormalised; with no member left, the global composite is used.
- `"product_dominance"` (Linum 772, Hemp 776): `ooil` where the country's seed production (linseed 333, hempseed 336) exceeds its fibre production (flax 771 "Flax, raw or retted", true hemp 777), otherwise `ofib`, both summed over 1961-2023. A country with neither product takes the world's dominant product and its global ratio. The four products are read from the raw `faostat-production` pin (`method_dominance` `"dominance_raw_faostat"`), the same item codes WHEP's primary production books them on, so ranking two products needs no 1961-2023 production build.

A country with no irrigated or no rainfed yield for the crop in SPAM takes the crop's global ratio (the ratio of the world's sums). Countries are matched by ISO3 code onto WHEP's polity buckets, never by name.

Level

`n_t` is the country's synthetic N over its cropland in year `t`: FAOSTAT's agricultural use of nitrogen (the `faostat-fertilizer-nutrients` pin) from 1961, back-cast to 1913 with the Smil (2001) global series scaled by the country's 1961-1965 share (smil_2001_synthetic_n_global), and zero before 1913, when there was no synthetic nitrogen. The share's divisor is the Smil series interpolated over 1961-1965 (15.6 Mt), the same divisor the spatialization scripts' `prepare_nitrogen_inputs()` uses (issue #1303). The cropland is `get_arable_permanent_land()` (FAOSTAT from 1961, LUH2 back-cast before).

A country that reports no N in year `t` takes the N per hectare of the polity that reported fertiliser for its territory that year, found by walking the `predecessor` edges of polities with `resolve_polity_lineage()`: Russia before 1992 takes the USSR's N over the USSR's cropland. A historical polity with no 2010 value of its own (the USSR) takes its successors' combined 2010 N over their combined cropland. Before 1961, a polity with a back-cast N but no back-cast cropland (the USSR and Czechoslovakia, whose codes have no LUH2 country) takes its successors' combined LUH2 cropland, rescaled to its own FAOSTAT cropland of 1961 (`method_ratio_cropland` `"successors_luh2_backcast"`); and a territory for which nothing reported N in 1961-1965 had no synthetic N before 1961, so its

level is 1 ("no_n_reported_pre1961"). method_ratio_trend names the path ("faostat", "faostat_predecessor", ...) and method_ratio_n_2010 the 2010 basis. A country with no synthetic nitrogen in 2010 keeps R_level = 1. What no path reaches gets no ratio (NA) rather than a guessed one.

Anomaly

Each item takes the LPJmL crop functional type of regime_yield_crop_mapping (lpjml_cft), including the "others" stand. The cell ratio is the irrigated over the rainfed per-stand yield of read_lpjml_regime_yield(). The cell's and the country's 1994-2023 ratios are each a ratio of pooled yields: each regime's yield is the sum of yield times stand area over the sum of stand area, over the cell's (or the country's cells') stands and the window years, the LPJmL counterpart of the SPAM anchor. Both parts are 1 in a cell the LPJmL run does not cover at all (its land mask misses some coastal and island cells), stamped "no_lpjml_cell". The spatial part is 1 where the cell has no 1994-2023 ratio; the temporal part is 1 where the cell has no ratio that year (a regime without a stand, or no rainfed harvest) or no 1994-2023 ratio. ratio_anomaly is their product, the cell-year ratio over the country's wherever all three ratios exist.

Polity columns

Every area-keyed output carries the polity its area_code resolves to in that row's year:

- polity_area_code: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use reporting_polity_code to say which territory a row belongs to.
- reporting_polity_code: The polity itself, e.g. ESP-1846-1914. It is year-aware, so the same area_code resolves to different polities in different years, which is the point of the crosswalk.
- reporting_polity_name: Its name. It can differ from the area's own name where the area folds into an aggregate.
- reporting_polity_has_geometry: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. FALSE is a documented gap upstream, not an error.

Rows whose area_code resolves to no polity keep the columns with NA rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See add_polity_code() for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 (area, year) cells – add_polity_code() says so as mapping_status == "backcast_anchor", and polity_coverage_gaps() reports it as gap_kind == "backcast_anchor". These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so reporting_polity_code can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after SUD-1956-2011 ends, and its post-2011 rows carry that code. These columns do not say so: add_polity_code() reports such a row as mapping_status == "out_of_span", and that column is dropped

here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

Yu, Q. et al. (2020). A cultivated planet in 2010 – Part 2: The global gridded agricultural-production maps. Earth System Science Data 12, 3545-3572. doi:10.5194/essd1235452020. LPJmL 6.1.1 band harvests as in `read_lpjml_regime_yield()`. FAOSTAT Fertilizers by Nutrient, Land Use and Crops and livestock products. Smil, V. (2001) *Enriching the Earth*, MIT Press.

Examples

```
build_regime_yield_ratio(example = TRUE)
```

`build_residue_bedding_supply` *Build the bedding straw supply reaching the managed manure chain.*

Description

Converts the bedding destiny of crop residues (`calculate_residue_destinies()`'s `residue_bedding_dm_t`) into the dry matter, carbon and nitrogen that `add_manure_bedding()` adds to the housed manure streams. Composition comes from the same `bio_coefs` residue coefficients `calculate_npp_carbon_nitrog` uses for the residue that stays on the field (`residue_c_kgdm`, `residue_n_kgdm`, joined on `item_prod_code`), so the straw that goes through the yard and the straw that does not are described by one table.

The supply is aggregated over crops, because bedding is not traceable to the crop it came from once it is in the heap: the manure chain keys on `year x territory` (and `sub_territory` when the input carries one).

Usage

```
build_residue_bedding_supply(x)
```

Arguments

`x`	A tibble from `calculate_residue_destinies()` with `item_prod_code`, `residue_bedding_dm_t`, `year` and `territory`, and optionally `sub_territory`.

Value

A tibble with one row per `year x territory` (x `sub_territory`) and columns `bedding_dm_t`, `bedding_c_t` and `bedding_n_t`.

Examples

```
tibble::tibble(
  year = 2020L, territory = "203", item_prod_code = "15",
  residue_bedding_dm_t = 1000
) |>
  build_residue_bedding_supply()
```

---

build_sjos_n_footprint

Build the embodied-nitrogen trade footprint.

Description

Trace a per-crop nitrogen category through the FABIO footprint framework and return the consumption-side embodied nitrogen split into domestic and traded flows. The `category` selects which nitrogen mass is carried as the footprint intensity (see `build_n_exceedance_extension()`): `"exceedance"` (default), `"within_boundary"`, or `"production"` (the crop's harvest removal: harvested product plus used residue plus grazed forage).

The pipeline is `build_n_exceedance_extension()` to assemble the extension, `build_footprint()` to trace it through the multi-regional input-output model, then a domestic-versus-traded relabelling of the tidy flows: a flow is `"Domestic consumption"` when the nitrogen is emitted and consumed in the same area (`origin_area == target_area`) and `"Traded"` otherwise. Two outputs are returned, mirroring Global's `FP_all_N` and `FP_food_all_N`: `fp_all` is the embodied nitrogen across all final-demand categories, `fp_food` is the subset consumed as food (`target_fd == "food"`). Signed crop attributions are traced as separate positive and negative linear extensions and recombined. Explicit undefined-attribution residuals are rejected by `build_n_exceedance_extension()` before tracing.

Usage

```
build_sjos_n_footprint(
  exceedance = NULL,
  io = NULL,
  category = c("exceedance", "within_boundary", "production"),
  years = NULL,
  data = list(),
  example = FALSE
)
```

Arguments

`exceedance`	A `build_n_boundary_exceedance()` country-resolution output passed straight to `build_n_exceedance_extension()`. Not needed when `example = TRUE`.
`io`	Optional pre-built `build_io_model()` result reused across extensions. When `NULL` (default), `build_footprint()` builds it for `years` from the package inputs (the real-data path, an integration wiring step).

`category` Which per-crop nitrogen mass to trace: `"exceedance"` (default), `"within_boundary"`, or `"production"`. Validated with `rlang::arg_match()`.

`years` Years to trace. Defaults to the years present in the extension; ignored when `io` is supplied.

`data` Optional named list of injected inputs. `data$fp_flows` supplies pre-traced tidy footprint flows (as from `build_footprint()`) directly, bypassing the model build, for testing the split logic in isolation. `data$origin_classes` may supply producer classifications keyed by `year`, `area_code`, `item_cbs_code` (for example `classify_sjos_n()` output). `data$target_classes` may supply consumer classifications, one row per country-year keyed by `year` and `area_code`, carrying `nourish` (for example `normalize_nourishment()` output) and optionally a country-year `boundary_side` and `sjos_class`. They join on `target_area` and `year` as `target_nourish`, `target_boundary_side` and `target_sjos_class`. Its `area_code` must be in the same code space as `target_area`, which is the IO model's `fd_labels$area_code`: the commodity balances' `polity_area_code` bucket (see `get_wide_cbs()`), not the source FAOSTAT area. A consumer bucket with no row in the table is reported as unclassified; a table keyed in another code space can match the wrong country wherever the two numberings share a code, so key it on the bucket. This function classifies nothing: a country-year boundary class is the caller's to classify after aggregation, and a table with more than one class per country-year (a crop-level `classify_sjos_n()` output, for instance) aborts rather than duplicating flows.

`example` If `TRUE`, return a small hardcoded fixture instead of running the pipeline. Defaults to `FALSE`.

Value

A named list with two tibbles:

- `fp_all`: embodied nitrogen by `year`, producer `origin_area` / `origin_item`, consumer `target_area` / `target_item`, `target_fd`, `origin` (`"Domestic consumption"` or `"Traded"`), `item_cbs_code` (an alias of `target_item`) and `impact_u` (tonnes N), stamped with the traced `category`, optional producer classes and, when `data$target_classes` is supplied, the `target_*` consumer classes.
- `fp_food`: `fp_all` restricted to food consumption (`target_fd == "food"`).
- `target_class_diag`: only when `data$target_classes` is supplied. One row per output table (`table`, `"fp_all"` or `"fp_food"`) and every `year` in `fp_all`, zero-filled where `fp_food` has no flows that year: the flow and consumer-area counts, `impact_u` (tonnes N), and how many flows, consumer areas and tonnes N went to a country-year with no `nourish` class (`n_flows_unclassified`, `n_target_areas_unclassified`, `impact_u_unclassified`). Those flows keep `NA` in `target_nourish` and stay in the tables; a warning of class `whep_sjos_fp_unclassified_target` names the consumer areas.

Examples

```
build_sjos_n_footprint(example = TRUE)
```

`build_sjos_nitrogen` *Assemble the end-to-end SJOS-N output tables.*

Description

Composes the Safe and Just Operating Space for nitrogen (SJOS-N) modules into a named list of analysis-output tables from one coherent set of inputs. The gridded soil-surface nitrogen surplus (`calculate_n_surplus()`) is compared to the Schulte-Uebbing critical nitrogen layer for the surplus-mode boundary (`build_n_boundary_exceedance()`, at grid and country resolution) and the same balance's process-based losses are routed to their medium-specific critical loads for the pathway boundary (`build_n_pathway_exceedance()`). The nourishment axis (`build_food_supply()` then `normalize_nourishment()`) is crossed with the country-aggregated exceedance into the 2-way classification (`classify_sjos_n()`) and, via the per-capita anthropogenic reactive nitrogen (`build_n_percapita()`), into the boundary-versus-nourishment scatter (`build_n_boundary_percapita()`). The country exceedance finally becomes an embodied-nitrogen trade footprint (`build_sjos_n_footprint()`), which carries the producer's classes (`origin_classes`, the per-crop classification) and the consuming country's nourishment class (`target_nourish`, joined on `target_area` and `year` from the same nourishment table). The driver joins no consumer boundary class: that is a country-year class decided after aggregation, which a caller supplies to `build_sjos_n_footprint()` directly.

The same nitrogen balance feeds the surplus and the pathway boundaries, the same nourishment feeds the classification and the scatter, and the one country exceedance feeds the classification, the footprint extension and the footprint: consistency is enforced by construction. When `example = TRUE`, a single coherent fixture set drives the whole chain without any real data.

Usage

```
build_sjos_nitrogen(
  data = list(),
  surplus_method = "harvest_removal",
  boundary_land_use = "ara",
  grassland_split = c("image_density", "none"),
  nh3_source = "soil",
  footprint_category = "exceedance",
  nourishment_thresholds = c("composed", "flat"),
  nourishment_band = list(),
  negative_critical = c("keep", "clamp"),
  country_table = FALSE,
  beyond_share_cut = 0.5,
  include = c("pathway", "scatter", "footprint"),
  example = FALSE
)
```

Arguments

`data` Named list of injected module inputs. When `example = FALSE` it must carry a `balance` (`build_nitrogen_balance()` output), a `critical` (`read_critical_n()` critical surplus), a `critical_loads` list (the three medium critical loads for the pathway boundary), `cbs_food` and `n_inputs`, and optionally `population` (read with `read_population()` at its own default composition over the years of `cbs_food` and `n_inputs` when absent; inject a table to use any other source), `biomass_coefs` / `items_full` for the food supply, `manure_mgmt_nh3_n_t` for the pathway boundary when `nh3_source = "total_agricultural"`, `critical_binding` (a `build_critical_n_binding()` table for the `boundary_land_use` scope, whose `binding_threshold` is then carried into the grid boundary; absent, the column is `NA`), and either an `io` model or `fp_flows` for the footprint. A real call without either source aborts rather than fabricating a domestic-only footprint. `grassland` is the `build_n_boundary_exceedance()` grassland-split input, only used and only optional when `boundary_land_use = "all"` and `grassland_split = "image_density"` (see `grassland_split`); a real call without it and without `critical` to build it from aborts rather than guessing a var/threshold to match. Defaults to `list()`.

`surplus_method`
Surplus definition passed to `calculate_n_surplus()`, `"harvest_removal"` (default) or `"full_balance"`.

`boundary_land_use`
Land-use scope stamp passed to `build_n_boundary_exceedance()`, `"ara"` (default, the robust historical comparison) or `"all"` (cropland and intensive grassland compared like for like against the critical allowance, extensive grassland against IMAGE's 2010 budget; see `grassland_split`) (issue #1285).

`grassland_split`
Grassland treatment passed to `build_n_boundary_exceedance()`, used only when `boundary_land_use = "all"`: `"image_density"` (default) splits each cell into a managed and an extensive component, `"none"` compares one cell pressure with the deposited `"all"`-scope allowance. Under `"image_density"`, `data$grassland` is used when supplied (its four elements, see `build_n_boundary_exceedance()`); otherwise it is built from `build_grassland_intensity_classes()`, the IMAGE 2010 extensive budget, and the `"ara"`/`"igl"` critical layers matched to `data$critical`'s own var and threshold. Ignored (no extra reads) when `boundary_land_use` is not `"all"`.

`nh3_source` Air-pressure scope passed to `build_n_pathway_exceedance()`, `"soil"` (default) or `"total_agricultural"`.

`footprint_category`
Which per-crop nitrogen mass the footprint traces, `"exceedance"` (default), `"within_boundary"` or `"production"`.

`nourishment_thresholds`
Which band the "just" axis classifies against: `"composed"` (default) builds it per country and year from `build_nourishment_band()`'s four sourced

terms, or "flat" restores the retired 62.1 / 85.05 pair. "flat" survives for continuity and sensitivity only: of its five underlying numbers only the 46 g/cap/day floor was ever sourced, and the 1.35 multiplier behind both bounds was a preliminary presentation figure (whep#753).

nourishment_band

Named list of options for the composed band, ignored when nourishment_thresholds = "flat". quality_method and quality_variant select the protein-quality tier and its bracket (build_protein_quality()); wedge_method and wedge_coverage select the loss wedge (build_loss_wedge()); shortfall, ceiling and requirement_sd go to build_nourishment_band() itself, and ceiling is the sensitivity knob the band's own documentation asks callers to sweep. An option this list does not name **aborts** rather than being ignored, so a mistyped knob cannot silently run the default and be reported as a sensitivity. Defaults to list(), which leaves every builder on its own default.

negative_critical

Treatment of cells whose critical surplus is below zero, passed to build_n_boundary_exceedance("keep" (default, as the source) or "clamp" (zero allowance, a declared departure from Schulte-Uebbing et al. 2022). It reaches the grid and country boundary and through them the classification and the footprint, and is stamped as negative_critical in both boundary tables, in sjos_class and in both footprint tables.

country_table

If TRUE, add the country_table element (see Value): the grid exceedance summarised to a country-year table by build_n_boundary_country(), with the country boundary side crossed with the nourishment class. It needs the agricultural land area, taken from data$ag_land (a build_ag_land_support() table) or, when absent, read with build_ag_land_support() for the years of the balance. Defaults to FALSE, which leaves every other element as it was.

beyond_share_cut

Share of a country's positive surplus above which it is on the "Exceedance" side, passed to build_n_boundary_country(); used only when country_table = TRUE. Defaults to 0.5, a WHEP criterion.

include

Which optional elements to return: any of "pathway" (boundary_pathway), "scatter", "footprint" and "band" (nourishment_band). Defaults to the first three. An element left out is absent from the result, and for the first three its inputs are not needed: without "pathway" no data$critical_loads, without "scatter" no data$n_inputs, and without "footprint" no data$io or data$fp_flows. "band" returns the composed band the nourishment axis was classified against, which is built either way, and needs nourishment_thresholds = "composed". The surplus boundary, the nourishment axis and the classification are always built, and are the same whichever elements are included.

example

If TRUE, drive the whole chain from the coherent fixture set instead of data. Defaults to FALSE.

Value

A named list of SJOS-N output tables: `surplus` (per-crop gridded surplus), `boundary_surplus` (a list with the `grid` and `country` surplus-mode exceedance), `boundary_pathway` (the pathway-mode exceedance with `binding_boundary`), `nourishment` (per-capita food supply with the normalized adequacy score and class), `scatter` (the per-capita boundary versus nourishment points; it and `nourishment` carry `method_population`, `"read_population"` or `"supplied"`), `sjos_class` (the 2-way classification) and `footprint` (a list with the `fp_all` and `fp_food` embodied-nitrogen footprints, both carrying `target_nourish`, and `target_class_diag`, the per-year count of flows whose consumer country-year has no nourishment class). The boundary tables, `sjos_class` and both footprint tables carry `negative_critical`. With `country_table = TRUE` the list also holds `country_table`, the `build_n_boundary_country()` result (a list with the `country` and `diagnostics` tables). With `"band"` in `include` it holds `nourishment_band`, the `build_nourishment_band()` table the nourishment axis used (floor, ceiling, prevalences and the `people_under` and `people_over` headcounts per country and year). Elements left out of `include` are absent.

Examples

```
build_sjos_nitrogen(example = TRUE)
```

---

build_soil_carbon_inputs

Assemble soil carbon inputs per cell, crop and year.

Description

Builds the carbon returned to soil from crop residues, crop roots, weeds and applied manure, as the carbon-input layer the soil-organic-carbon turnover models (`calculate_soc_dynamics()`) consume. Soil-returned residue carbon (residue net of the fraction removed for feed, fuel and burning), root carbon and weed (spontaneous-grass) carbon come from `calculate_npp_carbon_nitrogen()` (per polity, crop and year); manure carbon comes from `build_livestock_nutrient_flows()`'s `applied` stream. The four component carbon masses are converted to megagrams of carbon (1 tonne = 1 Mg), gridded to cells in proportion to each crop's harvested area, and divided by the cell-crop area to give Mg C per hectare per year. A carbon-weighted humification fraction is computed per cell-year from residue_humification, with the weed carbon humified at the weed (spontaneous-grass) coefficient.

The weed stream is structurally present but ZERO on the default path: the turnkey chain runs `calculate_crop_npp()` and `calculate_npp_carbon_nitrogen()` without `calculate_crop_npp_component` which is the only producer of `weed_ag_dm_t`, so weed carbon is zero for every crop, polity and year unless `data$npp` is supplied from a chain that includes it. Whether to wire it into the default path is open (whep#806); until then the zero is reported rather than passed off as computed.

At `"polity"` resolution the component carbon masses are summed back to `(area_code, item_prod_code, year)` and the per-hectare values and humified fraction re-derived from the polity totals.

Usage

```
build_soil_carbon_inputs(
  resolution = c("grid", "polity"),
  data = list(),
  years = NULL,
  method_unspatialized = c("fodder_pattern", "reallocate", "drop"),
  method_crop_weights = c("spatialized", "static"),
  method_manure_placement = c("livestock", "crop_area"),
  example = FALSE
)
```

Arguments

`resolution`	`"grid"` (default, per cell) or `"polity"` (aggregated to `area_code`).
`data`	Optional named list of pre-loaded inputs, each falling back to its reader when absent: `npp` (soil-returned residue, root and weed carbon per `area_code`, `item_prod_code`, `year`, columns `residue_soil_c_t`, `root_c_t` and `weed_npp_c_t`, tonnes C); `manure` (the `applied` tibble of `build_livestock_nutrient_flows()`, with `crop` either an existing `item_prod_code` or an `item_prod` name from items_prod_full (matched case-insensitively), and `territory` a stringified `area_code` – an `iso3c` literal is still resolved but deprecated, see `estimate_n_excretion()`); `country_grid`, the polycell support resolved to one row per cell and `area_code` (`lon`, `lat`, `area_code`, `cell_area_frac`, the polycell's share of the cell's land), refused when a cell-`area_code` group is duplicated or `NA` (DA-23); `crop_patterns` (the static per-cell `crop_area_ha` layer, read only under `method_crop_weights = "static"`); `gridded_crops` (the crop-level `build_gridded_landuse()` output on the same `country_grid` support, `lon`, `lat`, `area_code`, `item_prod_code`, `year`, `rainfed_ha`, `irrigated_ha`, read only under `"spatialized"` and built from the pinned spatialization inputs when absent); `harvested_area` (the FAOSTAT national harvested area per `area_code`, `item_prod_code`, `year` in a `faostat_area_ha` column, used to renormalize each polity-crop-year's spatialized cell area to the national total so per-hectare densities are the national density and carbon mass is conserved; defaults to the same `get_primary_production()` table the NPP reader uses, and is skipped when a hand-supplied `npp` keeps the pipeline offline unless supplied here); `residue_humification` (defaults to residue_humification); `fodder_patterns` (only read under `method_unspatialized = "fodder_pattern"`: per-cell fodder crop area in the `crop_patterns` schema, defaulting to the pooled Monfreda et al. (2008) forage layer from the `spatialize-fodder-patterns` pin, or rebuilt from the rasters under `WHEP_MONFREDA_DIR` when that is set).
`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the inputs cover. Threaded into the default NPP and manure readers so they slice to the requested years; ignored for inputs supplied via `data`.
`method_unspatialized`	What happens to a polity-crop whose crop has no hectares in the (time-

invariant) `crop_patterns` layer. `"fodder_pattern"` (default) first places the fodder crops (FAOSTAT items 636-649, 651 and 655) on the pooled 16 forage layers of Monfreda, Ramankutty and Foley (2008, doi:10.1029/2007GB002947), which the crop-pattern pin omits, gridded exactly as a crop-pattern crop is. The layers are pooled because the per-item ones follow the circa-2000 reporting vocabulary, not where each item is grown later. Whatever the layer still cannot place is reallocated as under `"reallocate"`. The layer is read from the `spatialize-fodder-patterns` pin (built by `inst/scripts/prepare_fodder_patterns.R`); setting `WHEP_MONFREDA_DIR` to the `HarvestedAreaYield175Crops_Geotiff` folder of the Monfreda archive (`inst/scripts/download/download_monfreda.R`) rebuilds it from the rasters instead, and `data$fodder_patterns` replaces it. It aborts rather than falling back to uniform reallocation when the layer is empty or a set `WHEP_MONFREDA_DIR` is wrong. The layer codes are matched on the layer names, as the archive's metadata gives no FAO name for them (assumed, unverified against a Monfreda table). At 2010 it moves 83.6 Mt C between cells relative to `"reallocate"`, with polity totals unchanged. `"reallocate"` spreads that carbon over the polity's crop-pattern cropland cells in proportion to each cell's total cropland area, and puts it on the crop's FAOSTAT national harvested area, so the polity's gridded carbon mass equals its national mass; this is the rule the sibling nitrogen spatialization already applies to the same gap (`spatialize_country_n_to_crops()`), and was the default before whep#1118. `"drop"` discards it, which is what the package did before and what a caller who prefers a hole to a smear should ask for. Either way the mass and the affected crops are reported, and the choice is recorded in `method_unspatialized`. Reallocation needs a national area to put the carbon on, so a polity-crop with no `harvested_area` row – and a polity with no cell at all in the support, which no rule here can reach (see whep#1002) – is dropped under all three methods, reported separately. Under either `method_crop_weights`, the fodder layer only fills the polity-crops that year's crop weights do not place, and the 83.6 Mt C figure above was measured on the static weights.

`method_crop_weights`

Where each crop's within-polity cell weights come from. `"spatialized"` (default) takes them from the spatialization engine's crop-level output, `build_gridded_landuse()` run on the pinned spatialization inputs and the same cell support, year by year: the crop's placement follows that year's cropland, its irrigated share and the engine's per-cell capacity ceiling, so the carbon lands where the gridded land use puts the crop. `data$gridded_crops` supplies that output pre-built. `"static"` uses the time-invariant `crop_patterns` layer (`harvest_fraction` times each cell's cropland averaged over every year of the gridded-cropland pin), one map for every year, which is what the package did before (whep#1002). Measured on the pins, the L1 distance between the two cell-share vectors of a polity-crop (0 = same placement, 2 = disjoint) has an area-weighted mean of 0.30 in 1961, 0.44 in 2010 and 0.47 in 2020. Polity totals are the same under both; only where the carbon sits moves. Recorded in `method_crop_weights`.

`method_manure_placement`
: Where the cropland manure carbon is put. `"livestock"` (default) places it where the animals are, the grain the gridded nitrogen balance uses for the same manure (whep#1300): the feed intake is spread to cells by gridded livestock heads on `country_grid` (the local grain of `build_feed_intake_local()`), and `build_livestock_nutrient_flows()` runs at `"subnational"`, so each cell's collected manure goes to that cell's crops up to the 170 kg N/ha ceiling and the excess is trucked to neighbouring cells with room. Its crop layer is that year's cell-crop area, the hectares the carbon is divided by. Manure with no crop – trucked in, or over-applied above the cap – is spread over its cell's crops by area, as the nitrogen balance spreads it, and manure on a cell with no crop over its polity's cropland; only a polity with no cropland cell loses it, with a warning. Measured at 2010, it places 322.1 Tg C of cropland manure where `"crop_area"` places 346.9 Tg C, because the cell-grain intake caps grazing at each cell's grass; 41.5% of the manure carbon sits on a different cell. `"crop_area"` allocates the polity's manure to its crops by harvested area and fans each polity-crop out to cells by crop area, as residue and roots are, which is what the package did before (whep#1307); it drops the over-cap manure the national allocation leaves with no crop (whep#805). A supplied `data$manure` must be keyed on cells (`sub_territory` a `"lon_lat"` cell id) under `"livestock"`; a national stream aborts. Recorded in `method_manure_placement`.

`example`
: If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble keyed by `(lon, lat, area_code, item_prod_code, year)` at `"grid"` resolution (or `(area_code, item_prod_code, year)` at `"polity"`), with `residue_c_mgc_ha_yr`, `root_c_mgc_ha_yr`, `weed_c_mgc_ha_yr`, `manure_c_mgc_ha_yr`, `total_c_input_mgc_ha_yr`, `input_cn` (the C:N of the components whose nitrogen is known), `input_n_mgn_ha_yr` (their nitrogen, Mg N per hectare per year, so `input_cn` times it is their carbon), `residue_n_mgn_ha_yr` and `root_n_mgn_ha_yr` (the soil-returned residue and root nitrogen within it, `NA` where unknown), `humified_fraction`, `method_c_input`, `method_unspatialized`, `method_crop_weights`, `method_manure_placement` and `crop_area_ha` – the crop's area at that grain on the basis the densities are computed on: the FAOSTAT-renormalised cell area where a national harvested area was supplied, the spatialized area otherwise, so a consumer can recover the carbon mass without re-deriving it – plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.

- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_soil_carbon_inputs(example = TRUE)
```

---

build_supply_use *Supply and use tables*

Description

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

Usage

```
build_supply_use(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

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.
  - `animal_draught`: Annual useful work energy generated by draft animals.
  - `slaughtering`: Slaughter of live animals into meat, edible offals, raw animal fats, and hides/skins.
  - `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, except for field co-products where the process uses the field aggregate item. For example, rice production uses the rice code, while seed cotton production uses the seed cotton code and supplies both cottonseed and cotton lint.
  - `husbandry`: The code of the farmed animal, e.g. bees for beekeeping, non-dairy cattle for non-dairy cattle husbandry, etc.
  - `slaughtering`: The code of the live animal being slaughtered, e.g. non-dairy cattle for cattle slaughter.
  - `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 the item's own unit. Most items are measured in tonnes; live animals are measured in heads; animal draught is measured as annual useful work energy (TJ).

Examples

```
build_supply_use(example = TRUE)
```

`build_total_population_grid` *Build gridded total population from UN WPP and HYDE.*

Description

Downscales UN World Population Prospects 2024 country totals (`read_wpp_population()`, `by = "total"`) to WHEP's 0.5-degree polycells by HYDE's total-population (`popc`) pattern. Each polycell's share of its country's HYDE population is linearly interpolated between the two nearest HYDE snapshots (decadal before 2001, annual 2001-2017 in the HYDE 3.x baseline release) and held at the last snapshot for later years, and WPP sets the level. So every country's cells sum to exactly its WPP total in every year.

This is the population `build_human_n()` reads under its default `population_basis = "total"`, where it is paired with the per-capita rate human_kgn_cap_total_reference, whose denominator is its calibration WPP total population, so the level and the rate share one basis.

Usage

```
build_total_population_grid(
  years = NULL,
  polity_validity = c("keep", "flag", "drop"),
  hyde_dir = NULL,
  data = list(),
  example = FALSE
)
```

Arguments

`years`	Integer vector of calendar years. Required. A year before the first HYDE snapshot, or one the WPP table does not carry (WPP 2024 starts in 1950), aborts with class `whep_total_population_uncovered`.

`polity_validity`
: What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`hyde_dir`
: Directory holding the HYDE `"{year}AD_pop.zip"` archives, whose file names set the available snapshot years. Defaults to `Sys.getenv("WHEP_HYDE_DIR")`. Ignored when `data$hyde` is supplied.

`data`
: Named list of inputs: `cell_polity` (required; `lon`, `lat`, `area_code`, optional `polity_frac`, as `build_cell_polity()` emits it, with the numeric WHEP area code), `wpp` (optional; `year`, `area_code`, `population`, bypassing `read_wpp_population()`) and `hyde` (optional; `lon`, `lat`, `year`, `popc` on the 0.5-degree grid, bypassing the HYDE archives; its years are the snapshot years).

`example`
: If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Details

WPP is keyed on the `polity_area_code` bucket, so shares are taken within each bucket (the one `build_cell_polity(area_key = "polity_area")` assigns) and the output keeps the grid `area_code` of `data$cell_polity`. A bucket with a WPP total but no HYDE population, and one with HYDE population but no WPP total, are each reported in a warning and in the `coverage` attribute rather than dropped silently.

The `coverage` attribute is a list recording what the numbers are: `population_basis` (`"total"`), `level_source`, `pattern_variable` (`"popc"`), the `exact`, `interpolated` and `held` years with `held_at`, the snapshots used, the per-year `plan`, and the `unplaced`, `unmatched_hyde` and `share_gaps` tables.

Value

A tibble with `lon`, `lat`, `area_code`, `year` and `population` (persons), one row per populated polycell-year, plus the polity columns below, and the `coverage` attribute described in Details.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.

- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_total_population_grid(example = TRUE)
```

`build_trade_prices` *Build global trade prices*

Description

Compute global prices of traded items from FAOSTAT trade data. For each item and element (import/export), the price is `KDollars / tonnes` aggregated across all countries.

Usage

```
build_trade_prices(raw_trade = NULL, example = FALSE)
```

Arguments

`raw_trade`	A data.table or tibble of FAOSTAT bilateral trade data with columns `year`, `item_trade`, `item_code_trade`, `unit`, `element`, and `value`. Must include both quantity (`"tonnes"`) and value (`"1000 US$"`) rows. If `NULL` (default), the data is read from the `"faostat-trade-bilateral"` pin.
`example`	Logical. If `TRUE`, return a small example tibble. Default `FALSE`.

Value

A tibble with columns:

- `year`: Integer year.
- `item_trade`: Trade item name.
- `item_code_trade`: Numeric FAOSTAT trade item code.
- `element`: `"import"` or `"export"`.
- `kdollars`: Total trade value in thousand US dollars.
- `tonnes`: Total trade quantity in tonnes.
- `price`: Price in KDollars per tonne.

Examples

```
build_trade_prices(example = TRUE)
```

---

build_water_balance *Build a gridded soil water balance from LPJmL hydrology.*

Description

Combines LPJmL monthly hydrology outputs into an annual per-cell water balance that closes as `water_input_mm = aet_mm + runoff_mm + drainage_mm + soil_water_change_mm`. Actual evapotranspiration (`aet_mm`) is the sum of the transpiration, evaporation and interception components (LPJmL has no direct AET, PET or temperature output). Water input is precipitation plus irrigation. `drainage_mm` is LPJmL deep seepage, the leaching driver downstream; `runoff_mm` is the runoff that does not pass through that seepage, which leaves the cell laterally and is not a leaching term but must appear in the budget. LPJmL's own total runoff (`mrunoff.nc`) is not used as `runoff_mm` directly, because it already contains the seepage: the seepage recharges groundwater and LPJmL adds the groundwater outflow to runoff, so adding seepage on top of it books that water twice. Drainage defaults to native seepage; the `"residual"` method instead reconstructs it as `water_input - aet - runoff - soil_water_change`, for use only when the seepage file is absent. Evapotranspiration is split into a blue (irrigation-sourced) and green (rain-sourced) part. The result is returned per grid cell, or aggregated to polity totals when `resolution = "polity"`.

The output also exposes the footprint-relevant terms folded into the budget: `prec_mm` (precipitation) and `irrig_mm` (applied irrigation, the gross blue-water volume), which satisfy `water_input_mm = prec_mm + irrig_mm`; `blue_consump_mm` and `green_consump_mm`, the LPJmL-native consumptive blue and green water (the per-CFT `cft_consump_water_b` / `cft_consump_water_g` cubes when supplied, summed over the bands `bands` selects, otherwise the blue and green AET); and `cft_nir_mm`, the net irrigation requirement (LPJmL `cft_nir`), the net blue-water demand, summed to cell level when `data$cft_nir` is supplied and `NA` otherwise. Potential evapotranspiration (`pet_mm`) comes from the CRU climate forcing that drives the LPJmL run and is `NA` until that forcing is wired (see `data$pet`); no PET formula is fabricated here.

Usage

```
build_water_balance(
  method = list(),
  resolution = c("grid", "polity"),
  polity_validity = c("keep", "flag", "drop"),
  data = list(),
  bands = NULL,
  example = FALSE
)
```

Arguments

`method` Named list selecting the estimation method for each term: `aet` (`"components"`, the only method), `drainage` (`"seepage"` default, LPJmL native seepage, or `"residual"`, a seepage reconstruction from the budget residual usable only when the seepage file is absent), `runoff` (`"total_minus_seepage"` default, LPJmL total runoff less the seepage it already contains, or `"surface_lateral"`, the sum of LPJmL's surface and lateral runoff outputs; see *Runoff* below) and `blue_green` (`"cft_native"` default, per-crop blue/green consumptive water, or `"irrig_share"`, the irrigation share of water input). Members left out take their default. `"cft_native"` warns when the per-CFT cubes report consumptive blue water on a *rainfed* band, which no irrigation can supply: that is the signature of an LPJmL 6.x run without the green/blue fix, whose split is unusable (the numbers are still returned, so the warning changes nothing; `"irrig_share"` does not read those cubes).

`partial_year` is the one member that is a policy rather than a method: what to do when a cell-year does not carry all twelve months, so that summing it to an annual flux would return a total over eleven. `"abort"` (default) refuses, naming the absent cell-months; `"warn"` returns the short sums; `"drop"` excludes the incomplete cell-years, so the year is absent rather than wrong. It is stamped into `method_water` as `partial:<choice>` only when it is not the default, because `"abort"` and `"warn"` return byte-identical numbers and only `"drop"` removes rows a reader could not otherwise account for.

Runoff. The two `runoff` methods are two ways of removing the seepage from LPJmL's total runoff, and they differ in what else they keep. `"total_minus_seepage"` keeps every other runoff component LPJmL routes (surface, lateral, snow, TOPMODEL baseflow and the lateral-water overflow), and treats this year's seepage as this year's groundwater outflow. Where groundwater storage changes within the year that difference lands in `runoff_mm`, which can then be negative (2.1% of cells in 2009 and 2.2% in 2010 on the LPJmL 6.1.1 run). `"surface_lateral"` reads `mrunoff_surf.nc` and `mrunoff_lat.nc`, never negative, but LPJmL does not write its baseflow or overflow, so they fall out of the budget. On that run, summed over 58,795 cells, the budget residual is +3.6% (2009) and +1.3% (2010) of water input with `"total_minus_seepage"`, and +6.8% and +4.2% with `"surface_lateral"`, against -19.2% and -21.4% when the seepage was counted twice. `drainage = "residual"` requires

"surface_lateral": "total_minus_seepage" needs the seepage that the residual method exists to do without. Because it then also absorbs the baseflow and overflow, the residual exceeds seepage (by 30% in 2009 and 19% in 2010 on that run).

`resolution` "grid" (per cell, default) or "polity" (aggregated to year and area_code).

`polity_validity`
What to do with a row whose (area_code, year) resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). "keep" (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. "flag" keeps them and adds the per-row logical reporting_polity_out_of_span, marking exactly which rows are stand-ins. "drop" removes them. All three warn; only "drop" changes the numbers. See polity_coverage_gaps(), which reports the same rows for an already-built table.

`data` Optional named list of pre-loaded inputs to avoid NetCDF reads: hydrology tibbles transp, evap, interc, prec, irrig, runoff (LPJmL total runoff), runoff_surf and runoff_lat (read only for runoff = "surface_lateral") and seepage (each lon, lat, year, value; annual-summed automatically when a month column is present), swc (lon, lat, year, month, layer, value fractional saturation), optional per-crop consumptive water cft_consump_water_b and cft_consump_water_g (each lon, lat, year, value mm/yr), an optional cft_nir net-irrigation-requirement input (lon, lat, year, value mm/yr, summed to cell level when supplied; exposed as cft_nir_mm, else NA) and a cell_polity crosswalk (lon, lat, area_code, polity_frac, cell_area_ha). Each falls back to read_lpjml_hydrology() when absent, except cft_nir (see Details), pet and the consumptive-water inputs. Read the latter with read_lpjml_hydrology("cft_consu monthly = FALSE), which names their CFT bands so bands can select among them.

Supplying any per-CFT cube also requires stand_frac (lon, lat, year, band_name or band, value), the per-CFT stand area fractions, read by default with read_lpjml_hydrology("stand_frac", monthly = FALSE). Every per-CFT cube is a density per square metre of ITS OWN STAND, so its bands must be weighted by these fractions before they are summed to a cell. Without them the aggregation aborts rather than returning an unweighted sum, which is not a whole-cell total at all: it overstates by 1 / (managed fraction of the cell).

`bands` Optional character vector of LPJmL crop-functional-type band names restricting which bands the per-CFT consumptive-water and net irrigation terms are summed over, e.g. "rainfed grassland" to charge a grazing footprint the grassland water alone. NULL (default) sums every band, the whole-cell total. Bands are matched on the band_name the file itself carries, so an unknown name aborts rather than silently returning the whole-cell total; the band index is never used, because which crop a given index denotes is a property of how the run was configured. Only blue_consump_mm, green_consump_mm and cft_nir_mm

are per-CFT, so this leaves the water budget itself untouched: `aet_mm`, `runoff_mm`, `drainage_mm`, `soil_water_change_mm` and the blue/green AET split (`aet_blue_mm`, `aet_green_mm`) are whole-cell quantities and take the same values whatever `bands` asks for. The split in particular is computed from the all-band consumptive totals, never the selected ones, because the fraction of a cell's evapotranspiration that came from irrigation cannot depend on which crop the caller asked about.

`bands` is also the only route to per-crop water today: read the per-CFT cubes once into `data`, then call once per band. Nothing in WHEP consumes the per-crop numbers yet, and on an LPJmL 6.x run without the green/blue fix their blue/green split is unusable (see `method`), so treat a per-band blue-water series as provisional.

`example` If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble. For `resolution = "grid"`: `lon`, `lat`, `area_code`, `year`, `water_input_mm`, `prec_mm`, `irrig_mm`, `pet_mm`, `aet_mm`, `aet_blue_mm`, `aet_green_mm`, `blue_consump_mm`, `green_consump_mm`, `cft_nir_mm`, `drainage_mm`, `runoff_mm`, `soil_water_change_mm` and `method_water`. For `resolution = "polity"`: the same terms aggregated to `year` and `area_code`. Both resolutions carry the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
build_water_balance(example = TRUE)
```

---

calculate_bnf *Estimate total biological nitrogen fixation.*

Description

Sums the three BNF components: symbiotic crop legumes, symbiotic weeds/cover crops, and non-symbiotic free-living fixation, by running `calculate_crop_bnf()`, `calculate_weed_bnf()` and `calculate_nonsymbiotic_bnf()`. When a `climate_type` column is present, the climate-specific parameters from `bnf_climate_params` override the relevant defaults per climate type.

Usage

```
calculate_bnf(
  x,
  symbiotic_params = list(),
  nonsymbiotic_params = list(),
  soil_params = list()
)
```

Arguments

`x`	A tibble carrying the required columns of all three component functions, optionally with a `climate_type` column.
`symbiotic_params`, `nonsymbiotic_params`, `soil_params`	Named lists passed to the component functions (see those functions).

Details

The weed component uses the `weed_npp_n_t` already present in `x`. In the standard crop-NPP chain this is non-zero only when `calculate_crop_npp_components()` has been run, or when callers supply weed NPP directly; `calculate_npp_carbon_nitrogen()` treats missing weed biomass as zero.

Value

The input tibble with all component columns plus `fert_type` (`"BNF"`) and `bnf_t` (total BNF).

Examples

```
calculate_bnf(
  tibble::tibble(
    item_prod_code = "176", crop_npp_n_t = 10, product_n_t = 5,
    weed_npp_n_t = 4, land_use = "Cropland", legumes_seeded = 0,
    seeded_cover_crop_share = 0, area_ha = 40
  )
)
```

---

calculate_cohorts_systems

Calculate cohort and production system distribution.

Description

Distributes national herd totals across GLEAM-defined cohorts and production systems using `gleam_livestock_categories` and regional weight data.

GLEAM supplies only the taxonomy here: which systems and cohorts each species has. It supplies no herd shares. `gleam_livestock_categories` has no share column, and the herd-parameter tables of the GLEAM 2.0 and 3.0 supplements give demographic rates and live weights, not a dairy/meat or layer/broiler split of the herd (whep#1194).

Where FAOSTAT already reports the herd split into the items `build_primary_production()` carries (`"Cattle, dairy"` / `"Cattle, non-dairy"`, `"Pigs"` / `"Hogs"` for market / breeding swine, and `"Chickens, layers"` / `"Chickens, broilers"`), the whole herd goes to the system its item names and no share is applied. Every other herd (buffalo, sheep, goats, ducks, turkeys, geese, and any aggregate cattle, swine or poultry label) is split by WHEP's default shares, which are **assumed, unverified** placeholders with no source. The `method_system_share` column says which of the two applied to each row.

Within a system, `method_cohorts` says how the heads reach cohorts. A `"Cattle, dairy"` head is a cow producing milk: FAOSTAT's dairy cattle are the Milk Animals element of cow milk, and its non-dairy cattle are the rest of the herd, "cattle minus dairy cattle" (Tubiello et al. 2015, *Estimating Greenhouse Gas Emissions in Agriculture*, FAO, ISBN 978-92-5-108674-2, pp. 45-46). So by default those heads all go to the dairy system's `"Adult Female"` cohort, and the dairy herd's bulls, heifers and calves are counted where FAOSTAT counts them, in the non-dairy herd (whep#1127). Every other herd is split equally across its system's cohorts, which is **assumed, unverified**: GLEAM supplies no cohort shares. The `method_cohort_share` column says which applied to each row.

Usage

```
calculate_cohorts_systems(
  data,
  system_shares = NULL,
  method_cohorts = c("milk_animals", "uniform")
)
```

Arguments

`data`	Dataframe with `species`, `heads`, and optionally `iso3` or `region`.
`system_shares`	Optional dataframe with `species_gen`, `system`, `system_share` columns. If `NULL`, a herd whose commodity names a production system goes wholly to it, and every other herd uses WHEP's assumed, unverified default shares. Supplying this overrides both, so the supplied shares are used verbatim.
`method_cohorts`	How heads are split across the cohorts of a system: • `"milk_animals"` (default): a dairy-cattle herd (`"Cattle, dairy"`, the FAOSTAT Milk Animals count) goes wholly to the dairy system's `"Adult Female"` cohort; every other herd is split equally across its system's cohorts. • `"uniform"`: every herd, dairy cattle included, is split equally across its system's cohorts, the behaviour before whep#1127. It books five sixths of the milking cows as bulls, heifers and calves, and is kept for sensitivity analysis.

Value

Dataframe expanded to cohort level with `cohort`, `system`, `cohort_heads`, and `cohort_fraction` columns, plus `method_system_share`: `"reported"` when the commodity itself names the system, `"assumed"` when WHEP's unsourced default split was applied, or `"supplied"` when `system_shares` was given, and `method_cohort_share`: `"milk_animals"` for a dairy-cattle herd routed to its milking cows, `"uniform"` for an equal split across cohorts, or `NA` for a species with no cohorts. A `milk_yield_kg_day` column (per head of the input row) is moved onto the milked cohort, the `"Dairy"` system's `"Adult Female"`, at the yield that keeps the herd's milk unchanged; every other cohort gets 0. `method_milk_yield` says which: `"milked_cohort"`, `"not_milked_cohort"`, or `"whole_herd"` for a species with no cohorts. A herd with milk but no milked cohort aborts (class `whep_milk_without_milked_cohort`).

Examples

```
tibble::tibble(
  species = "Cattle", heads = 10000,
  iso3 = "DEU"
) |>
  calculate_cohorts_systems()
```

`calculate_critical_n` *Calculate critical-nitrogen allowances with the Schulte-Uebbing method.*

Description

Computes the critical nitrogen input and surplus of every 0.5-degree cell with the equations of Schulte-Uebbing et al. (2022, Nature 610, 507-512, doi:10.1038/s41586-022-05158-2, Supplementary Information Eqs. 1-36), instead of reading the fixed 2010 layers of their archive (doi:10.5281/zenodo.6395016) as `read_critical_n()` does by default. The critical input is the fertiliser and manure that keeps each of three impacts at its threshold, plus the constant fixation and the deposition it implies:

- `"de"`: deposition at the critical load of the cell's biome (5 to 20 kg N/ha of cell area, SI Supplementary Table 2);
- `"sw"`: 5 mg N/l in runoff to surface water (SI Eq. 12);
- `"gw"`: 11.6 mg NO3-N/l in water leaching from agricultural soil (SI Eq. 29);
- `"mi"`: the lowest of the three, land use by land use (SI Eqs. 31-32).

Fertiliser and manure change in their current proportion (SI Eq. 3); fixation, the non-agricultural sources and extensive grassland stay at their current values. Where the non-agricultural sources alone exceed a threshold, fertiliser and manure are set to zero (`critical_rule "non_agricultural_floor"`). Where the threshold allows more than crops can use, the input is cut off at the input needed for the regional yield potential at current nitrogen use efficiency, capped at 0.8 (Methods Eqs. 1-2; `"yield_potential_cap"`). The critical surplus is input minus uptake (SI Eqs. 5-6).

With the archive's own 2010 inputs the result reproduces the deposited layers closely but not exactly, because the source does not print every rule it applied (see the section below). Supplying `inputs` from another year or model applies the same method to them.

Usage

```
calculate_critical_n(
  inputs = NULL,
  dir = NULL,
  verify_source = TRUE,
  example = FALSE
)
```

Arguments

`inputs`	Optional tibble with one row per cell and the IMAGE-GNM quantities listed by `whep:::.critn_input_specs()`: `cell_id`, `lon`, `lat`, areas in hectares (`area_total_ha`, `area_arable_ha`, `area_intensive_ha`, `area_extensive_ha`, `area_natural_ha`), the IMAGE `biome` and `image_region` codes, `runoff_l` (litres per year) and the nitrogen flows in kg N per cell per year. Fertiliser and manure are net of NH3, as the archive deposits

	them. When `NULL` (default) the 2010 inputs are read from the archive's `Input_files`.
`dir`	Optional archive directory, resolved as in `read_critical_n()`. Ignored when `inputs` is supplied.
`verify_source`	If `TRUE` (default), the archive's input rasters are checked against the package's content manifest before they are read. Ignored when `inputs` is supplied.
`example`	If `TRUE`, return the result for a small fixture of four cells instead of reading data. Defaults to `FALSE`.

Value

A tibble with one row per cell, threshold and land-use scope: `cell_id`, `lon`, `lat`, `image_region`, `critical_threshold` (`"de"`, `"sw"`, `"gw"` or `"mi"`), `critical_land_use` (`"ara"`, `"igl"` or `"all"`), `area_ha` (the hectares of that scope), the critical `critical_n_input_kgn_ha` and `critical_n_surplus_kgn_ha`, the current `current_n_input_kgn_ha` and `current_n_surplus_kgn_ha` (all kg N per hectare per year), `critical_rule` (`"environmental_threshold"`, `"non_agricultural_floor"` or `"yield_potential_cap"`; `NA` for `"all"`) and `method_critical_n = "reproduced"`. A cell whose land use has no current fertiliser or manure, no crop uptake, or no agricultural leaching has no critical value there, as in the archive.

Rules recovered from the deposited layers

The SI prints the equations for cells with one reducible agricultural land use and states that cells combining arable land with grassland follow "slightly different" formulas. The rules for those cells, and several values the text leaves out, were recovered by recomputing the archive's outputs from its inputs: fertiliser and manure enter gross of their NH3 emission; the groundwater limit is 11.6 mg NO3-N/l (the article's Methods print 11.3); ice cells get 5 kg N/ha of critical deposition (Supplementary Table 2 prints "n.a."); the fertiliser share is clipped to [1e-4, 1 - 1e-4]; the regional yield-gap ratios carry three decimals (each rounds to the two printed in Supplementary Table 5). The cut-off binds where uptake at the critical input (with the deposition of every land use in the cell) reaches uptake at yield potential; with NUE above 0.8 this raises the input to the cut-off input. In cells with both arable land and intensive grassland, the NH3 the critical deposition leaves room for is shared by their current NH3 emission, and both land uses are scaled by one factor for the surface-water threshold. For groundwater there, each land use is held to its area share of the critical leaching and the one that must fall further is solved first; this rule is a reconstruction, and those cells carry the remaining differences.

One departure is deliberate: `"mi"` is the lowest of the three thresholds as SI Eq. 31 defines it. In 57 arable cells where deposition and groundwater both sit at the non-agricultural floor, the deposited all-impacts layer carries the higher surface-water value instead.

Examples

```
calculate_critical_n(example = TRUE)
```

calculate_crop_bnf *Estimate symbiotic biological nitrogen fixation by crop legumes.*

Description

Estimates symbiotic BNF from leguminous crops via two methods (an NPP-based estimate and the Anglade product-based estimate). Environmental modifiers (nitrogen inhibition, temperature, water) are applied only when their driver columns are present in `x`; an absent driver leaves that modifier at 1.

Usage

```
calculate_crop_bnf(x, symbiotic_params = list())
```

Arguments

`x`	A tibble with `item_prod_code`, `crop_npp_n_t` and `product_n_t`. Optional driver columns activate the modifiers: `n_synth_kg_ha` / `n_org_kg_ha` (nitrogen inhibition), `temp_c` (temperature), `water_input_mm` (or `precip_mm` + `irrig_mm`) and `pet_mm` (water).
`symbiotic_params`	Named list overriding the symbiotic-BNF parameters `k_n_synth`, `k_n_org`, `t_opt`, `t_sigma`, `ai_threshold`.

Value

The input tibble with the environmental factors, `ndfa_adj`, `crop_bnf_t` (NPP method), `crop_bnf_anglade_t` (Anglade method) and the per-product ratios `bnf_product_ratio_npp` / `bnf_product_ratio_anglade`.

Examples

```
calculate_crop_bnf(
  tibble::tibble(item_prod_code = "176", crop_npp_n_t = 10, product_n_t = 5)
)
```

calculate_crop_npp *Estimate total crop net primary production.*

Description

Assembles total crop net primary production (product plus residue plus root dry matter) by running `calculate_crop_residues()` then `calculate_crop_roots()`.

Usage

```
calculate_crop_npp(
  x,
  residue_method = "ensemble",
  root_method = "ensemble",
  weights = list(w_ipcc = 0.5, w_ref = 0.5),
  root_irrigation = "except_paddy"
)
```

Arguments

x	A tibble with item_prod_code, production_t and area_ha, plus any optional adjustment columns used by the residue and root steps.
residue_method	Residue method passed to calculate_crop_residues().
root_method	Root method passed to calculate_crop_roots().
weights	Named list of ensemble weights; w_ipcc for residues and w_ref for roots (each 0-1, default 0.5).
root_irrigation	Irrigated root:shoot scope passed to calculate_crop_roots() as irrigation.

Value

The input tibble with product_dm_t, yield_dm_t_ha, residue_dm_t, root_dm_t, crop_npp_dm_t, method_residue and method_root.

Examples

```
calculate_crop_npp(
  tibble::tibble(item_prod_code = "15", production_t = 100, area_ha = 40)
)
```

---

calculate_crop_npp_components

Estimate cropland NPP components including weeds.

Description

Assembles full cropland net primary production: scales weed biomass from potential NPP and the weed_npp_scaling factors, then partitions crop and weed biomass into dry matter, carbon and nitrogen via calculate_npp_carbon_nitrogen().

Usage

```
calculate_crop_npp_components(
  x,
  .by = NULL,
  potential = list(method = "lpjml")
)
```

Arguments

`x`	A tibble with `item_prod_code`, `area_ha`, `year`, `product_dm_t`, `residue_dm_t` and `root_dm_t` (e.g. the output of `calculate_crop_npp()`). When `npp_potential_dm_t_ha` is absent it is computed via `calculate_potential_npp()` (which, for the default `lpjml` method, needs `lon`/`lat`).
`.by`	Optional character vector of grouping columns used to fill missing weed-scaling factors with the group mean.
`potential`	A named list selecting the potential-NPP source: `method` (default `"lpjml"`) and `lpjml` options.

Details

The `weed_npp_scaling` table is taken from an earlier regional historical reconstruction and is flagged `to_be_revised`: it is Spain-specific and not validated for WHEP's global scope. A `weed_scaling_to_be_revised` column records this on the output and a one-time warning is emitted.

Value

The input tibble with weed dry matter, the dry-matter / nitrogen / carbon partition for crop, weeds and total, and `weed_scaling_to_be_revised`.

Examples

```
tibble::tibble(
  item_prod_code = "15", production_t = 100, area_ha = 40,
  year = 2000, npp_potential_dm_t_ha = 5
) |>
  calculate_crop_npp() |>
  calculate_crop_npp_components()
```

`calculate_crop_residues`

Estimate crop above-ground residue biomass.

Description

Estimates crop residue dry matter from production and area using an ensemble of an IPCC 2019 linear model and a `bio_coefs` residue:product ratio. The irrigation and modern-variety adjustments activate only when their driver columns are present in `x`.

Usage

```
calculate_crop_residues(
  x,
  method = c("ensemble", "ipcc", "ratio"),
  weights = list(w_ipcc = 0.5)
)
```

Arguments

`x`	A tibble with `item_prod_code`, `production_t` (fresh matter) and `area_ha`. Optional `water_regime` enables the irrigation adjustment; optional `year` plus `region_hanpp` enable the modern-variety adjustment.
`method`	Residue method: `"ensemble"` (default, weighted IPCC + ratio), `"ipcc"` (IPCC linear only) or `"ratio"` (`bio_coefs` ratio only).
`weights`	Named list; `w_ipcc` is the IPCC weight in the ensemble (0-1, default 0.5), used only when `method = "ensemble"`.

Value

The input tibble with `product_dm_t`, `yield_dm_t_ha`, `residue_dm_t`, `method_residue` and `method_residue_group`. The latter is `"ipcc_group"` for an item with an IPCC crop group, and `"unmapped_uncorrected"` for one without: that item gets no irrigation or modern-variety residue correction (both are keyed on the crop group), which is not the same as a correction that was computed and found to be 1.

Examples

```
calculate_crop_residues(
  tibble::tibble(item_prod_code = "15", production_t = 100, area_ha = 40)
)
```

`calculate_crop_roots` *Estimate crop below-ground (root) biomass.*

Description

Estimates root dry matter from above-ground biomass using an ensemble of an IPCC root:shoot ratio and a reference below-ground biomass per hectare. The nitrogen-input and irrigation adjustments activate only when their driver columns are present in `x`.

Usage

```
calculate_crop_roots(
  x,
  method = c("ensemble", "root_shoot", "reference"),
  weights = list(w_ref = 0.5),
  irrigation = c("except_paddy", "all_crops", "none")
)
```

Arguments

`x`	A tibble with `item_prod_code`, `product_dm_t`, `residue_dm_t` and `area_ha`. Optional `n_input_kg_ha` enables the nitrogen adjustment; optional `water_regime` enables the irrigation adjustment.

`method` Root method: `"ensemble"` (default, weighted root:shoot + reference), `"root_shoot"` (root:shoot ratio only) or `"reference"` (reference below-ground biomass only).

`weights` Named list; `w_ref` is the reference weight in the ensemble (0-1, default 0.5), used only when `method = "ensemble"`.

`irrigation` Which irrigated crops take the root:shoot factor of `irrigation_adj` (0.85 irrigated), used only when `x` has `water_regime`: `"except_paddy"` (default) applies it to every crop but paddy rice, which is flooded whether irrigated or not; `"all_crops"` applies it to every crop, as afsetools does; `"none"` applies no irrigation factor. Water stress raises root:shoot across crops (Eziz et al. 2017, doi:10.1002/ece3.3630; Kou et al. 2022, doi:10.3389/fpls.2022.1085409; Seidel et al. 2024, doi:10.1016/j.scitotenv.2024.176738), implying an irrigated factor of about 0.70 to 0.88 (whep#1536).

Value

The input tibble with `root_dm_t` and `method_root`, plus `method_root_irrigation` when `x` has `water_regime`.

Examples

```
calculate_crop_roots(
  tibble::tibble(
    item_prod_code = "15",
    product_dm_t = 87.9,
    residue_dm_t = 135.75,
    area_ha = 40
  )
)
```

`calculate_enteric_ch4`

Calculate enteric methane emissions.

Description

Wrapper that selects Tier 1 or 2 for enteric CH4 based on data availability.

Usage

```
calculate_enteric_ch4(data, tier = NULL, options = list())
```

Arguments

`data` Dataframe with `species`, `heads`. For Tier 2, also needs `cohort`, `weight`, and `diet_quality`. For Tier 1, `iso3` is used to select regional emission factors.

`tier` Integer 1 or 2. If `NULL` (default), auto-selects based on data completeness.

`options` A named list of manure-engine options. All but nine defaults reproduce the behaviour in force before whep#949. The exceptions are `enteric_ef_source`, `ef3_source` and `slurry_crust`, which since whep#601 read the 2019 Refinement's Tier 1 enteric factors and direct manure N2O EF3 and move Tier 1 enteric CH4 and both tiers' direct manure N2O, `mcf_source`, which moved from the shipped table to the 2019 Refinement in whep#1022 and does move Tier 2 manure CH4, `mms_shares`, which moved from the unsourced placeholder table to the GLEAM 2.0 ingest in whep#958 and does move both tiers' manure N2O, `pasture_bo`, which since whep#1137 pairs the 2019 pasture MCF with its published `Bo` and moves Tier 2 manure CH4, and `tier2_uncovered`, which since whep#1028 gives species with no Tier 2 method their Tier 1 values instead of `NA`, `indirect_n2o_source`, which since whep#1245 reads the 2019 Refinement's leaching factors and moves both tiers' indirect manure N2O, and `indirect_n2o_fractions`, which since whep#1365 weights the Table 10.22 fractions over the manure-management split and moves it again.

`indirect_n2o_source` selects the edition of indirect_n2o_ef the indirect manure N2O reads:

- `"ipcc_2019"` (default): EF5 0.011 and FracLEACH-(H) 0.24, from Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26 of the 2019 Refinement, the current IPCC guidance and the EF5 the nitrogen balance already uses.
- `"ipcc_2006"`: EF5 0.0075 and FracLEACH-(H) 0.30, from Table 11.3, p. 11.24 of the 2006 Guidelines. These are the values WHEP shipped before whep#1245 under a 2019 citation, kept selectable so earlier figures stay reproducible.

EF4 (0.010) is the same under both. The edition also selects the Ch 11 FracGASM (0.21 in 2019, 0.20 in 2006) the pasture stream is priced at.

`indirect_n2o_fractions` selects which nitrogen fractions the indirect manure N2O volatilises and leaches:

- `"per_mms"` (default): IPCC 2019 Refinement, Vol 4, Ch 10, Eq. 10.26 and 10.27 (Updated), pp. 10.76-10.77. `FracGasMS` and `FracLeachMS` per animal category and system, from Table 10.22 (Updated), pp. 10.96-10.97, weighted over the row's own manure-management split, the same split the direct EF3 is weighted over. Pasture/range/paddock N, which Table 10.22 does not cover, is priced at the Ch 11 Table 11.3 FracGASM and FracLEACH-(H) (Eq. 11.9 and 11.10), as the IPCC prices grazing deposition; no other WHEP chain prices it, and the direct path prices the same N at EF3, so it is counted once. The managed-system fractions are the 2019 table's under either edition, as in `apply_management_losses()`: the 2006 Table 10.22 leaves most pairs blank.
- `"single"`: one FracGasMS 0.20 (assumed, unverified) and one FracLEACH-(H) on all excreted N, the form before whep#1365. Kept so earlier figures stay reproducible.

`method_manure_n2o` records both choices, e.g. `indirect_ipcc_2019_per_mms` or `indirect_ipcc_2006_single`.

`enteric_ef_source` selects the Tier 1 enteric CH4 emission factors. Although it sits among the manure-engine options, it governs the enteric path only:

- `"ipcc_2019"` (default): ipcc_enteric_ef_tier1, the 2019 Refinement's Tables 10.10 and 10.11 (Updated) as published: cattle and buffalo by IPCC region, and sheep, goats and swine at the productivity system the Table 10.10 footnote assigns the region (high in North America, Europe and Oceania, low elsewhere). A row the edition prices at nothing keeps the as-shipped factor, stamped `enteric_ef_as_shipped_fallback`.
- `"as_shipped"`: ipcc_2019_enteric_ef_cattle and ipcc_2019_enteric_ef_other, the 2006 Guidelines' factors with four cells in neither edition, read before whep#601. Kept so earlier figures stay reproducible.

`method_enteric` records which (`enteric_ef_ipcc_2019` or `enteric_ef_as_shipped`). On FAOSTAT 2020 heads the default raises Tier 1 enteric CH4 from 110.8 to 125.3 Tg. Taking high productivity everywhere instead would give 134.2 Tg, low everywhere 124.2 Tg.

`ef3_source` selects the direct manure N2O emission factors, at both tiers: `"ipcc_2019"` (default) or `"as_shipped"`, the two editions of ipcc_manure_ef3.

`slurry_crust` sets the share of liquid slurry with a natural crust, which mixes the 2019 Table 10.21 (Updated) EF3 with a crust (0.005) and without one (0) into the engine's single `"Liquid/Slurry"` label: `"half"` (default, 0.0025, GLEAM 2.0's stated assumption for its pig and chicken slurry, extended to the other species), `"none"` (0) or `"all"` (0.005). It has no effect under `ef3_source = "as_shipped"`, whose 0.002 is a single row.

`method_manure_n2o` records the EF3 choice (`ef3_ipcc_2019; slurry_crust_half` or `ef3_as_shipped`). On FAOSTAT 2020 heads the default lowers Tier 1 direct manure N2O from 1.523 to 1.178 Tg.

`mms_shares` selects which half of regional_mms_distribution the split is read from: `"gleam_2_0"` (default) is the GLEAM 2.0 Supplement S1 Tab. 4.2-4.11 ingest, `"placeholder"` the unsourced table it replaced in whep#958. The placeholder stays selectable so the values WHEP published before that ingest remain reproducible and the sensitivity to it stays measurable; it is not a defensible alternative estimate.

`mms_region` selects how the manure-management split in regional_mms_distribution is keyed:

- `"as_available"` (default): a row uses its own region when the frame already carries a `region` column, and the `region == "Global"` split otherwise. Tier 1 resolves a region for the (sourced) per-head N-excretion table and so takes the region-specific split; Tier 2 carries no region and so takes the Global one.
- `"resolve"`: the IPCC region is resolved from `iso3`, `area_code` or `polity_area_code` where it is missing, which makes the table's region-specific rows live on the Tier 2 path too. Opt-in because it changes which rows of the table apply, not because the rows are doubtful: since whep#958 they are the GLEAM 2.0 ingest.

- `"global"`: every row takes the `region == "Global"` split, whatever region column it carries.

`mcf_source` selects which methane conversion factor table the Tier 2 manure CH4 weighting reads:

- `"ipcc_2019"` (default): the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 (Updated) of the 2019 Refinement, which is the current IPCC guidance.
- `"ipcc_2006"`: the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 of the 2006 Guidelines.
- `"as_shipped"`: climate_mcf, whose live rows are predominantly the 2006 Guidelines Table 10.17 but with six cells that match no published IPCC value (whep#601, whep#1022). Kept selectable so an older run can be reproduced; it is no longer the default, because values whose provenance could not be established should not be what ships.

Neither edition publishes one number per Cool/Temperate/Warm zone for every system, so both as-published tables apply a stated collapse rule; see climate_mcf_ipcc. Both rules are WHEP's, not the IPCC's, and the default makes them live. `method_manure_ch4` records the table used.

`pasture_bo` selects the methane potential (`Bo`) the Tier 2 manure CH4 prices the pasture/range/paddock stream at. The 2019 Refinement publishes its single 0.47 percent pasture MCF as half of a pair that "must always be used in conjunction with a B0 value of 0.19" (Vol 4, Ch 10, Table 10.17 (Updated) footnote 2, p. 10.70; Section 10.4.2, p. 10.66); the pair is the `paired_bo_m3_kg_vs` column of climate_mcf_ipcc (whep#1137).

- `"paired"` (default): a stream whose MCF row carries a paired `Bo` is priced at it, every other stream at the animal-category `Bo` of ipcc_tier2_bo_values. Only `mcf_source = "ipcc_2019"` publishes a pair, so under the other two sources this changes nothing.
- `"species"`: every stream takes the animal-category `Bo`, the behaviour before whep#1137. Under `"ipcc_2019"` that is the hybrid the Refinement rejects; kept so the earlier figures stay reproducible and the sensitivity to the pairing stays measurable.

`method_manure_ch4` records per row which applied (`pasture_bo_paired` or `pasture_bo_species`) wherever the row has manure on a stream that carries a published pair.

`climate_source` selects where the climate zone the methane conversion factors in the MCF table are read at comes from. A `climate_zone` a row already carries is always used and stamped `climate_from_data`; the option governs only the rows left without one, whether that is a hole in a supplied column or a wholly absent column.

- `"assumed"` (default): fill with `assumed_climate_zone`.
- `"from_data"`: abort instead of assuming.

`assumed_climate_zone` is the zone `"assumed"` fills in: `"Cool"`, `"Temperate"` (default) or `"Warm"`. It is an assumption, not a measurement; `method_manure_ch4`

records per row which of the sources applied, and this argument exists so the sensitivity to the assumption can be measured (whep#949).

`tier2_uncovered` says what the Tier 2 path does with a species WHEP has no Tier 2 method for. The energy balance needs the maintenance and activity coefficients of Tables 10.4 and 10.5, which ipcc_tier2_energy_coefs ships for cattle, buffalo, sheep and goats only; the 2019 Refinement itself suggests Tier 1 for camels, horses, mules and asses and swine and has no enteric method for poultry (Vol. 4 Ch. 10, Table 10.9 (Updated)), and its Tier 2 manure equations for swine and poultry need a country-specific dry-matter intake (Equation 10.32A) that WHEP does not hold. Although it sits among the manure-engine options, it governs the enteric path too.

- `"tier1"` (default): those species take the Tier 1 enteric CH4, manure CH4 and manure N2O, written into the Tier 2 output columns and stamped `"IPCC_2019_Tier1"` in `method_enteric`, `method_manure_ch4` and `method_manure_n2o`, with a message naming them. This is the IPCC's own suggested method for them, so a Tier 2 inventory keeps the whole herd rather than silently covering fewer animals than Tier 1.
- `"leave_na"`: they keep `NA` emissions, the behaviour before whep#1028, with a warning naming them. Kept so a ruminant-only Tier 2 figure stays reproducible.
- `"abort"`: any such species aborts, naming it.

Value

Dataframe with all input columns preserved, plus:

- `method_enteric`: tracking label (`"IPCC_2019_Tier1"` or `"IPCC_2019_Tier2"`). At Tier 2 a species with no Tier 2 method carries `"IPCC_2019_Tier1"` and its Tier 1 value (see `tier2_uncovered` under `options`).
- Tier 1: `enteric_ef_kgch4` (emission factor), `enteric_ch4_tier1` (total kg CH4).
- Tier 2: `gross_energy`, `ym_factor`, `enteric_ch4_per_head` (kg CH4/head/yr), `enteric_ch4_tier2` (total kg CH4).

Examples

```
tibble::tibble(
  species = "Cattle", heads = 1000, iso3 = "DEU"
) |>
  calculate_enteric_ch4(tier = 1)
```

`calculate_food_gini` *Between-country population-weighted Gini of per-capita food supply.*

Description

Reduces a per-capita nourishment series to a per-year, between-country, population-weighted Gini coefficient, a diagnostic of how unequally per-capita supply is shared across countries (it does not rescale the per-country nourishment score). Within each `year`, countries are sorted in descending per-capita supply; each country's population fraction `pop_frac` and its share of the total supply mass `value_frac` (per-capita supply times population, over the world total) give a score `value_frac * (pop_frac + 2 * richer_frac)`, where `richer_frac` is the cumulative population fraction of the strictly richer countries. The Gini is `1 - sum(score)` over the countries in that year, `0` under a perfectly equal distribution and rising toward `1` as supply concentrates. Protein per-capita supply is the SJOS-N nourishment axis, so it is the default; passing a different `value_col` (for example `energy_kcal_cap_day`) takes the Gini of that axis instead. Rows with a missing supply or population value are dropped before the computation.

Usage

```
calculate_food_gini(x, value_col = protein_g_cap_day, pop_col = population)
```

Arguments

`x`	A tibble with a `year` column, the per-capita supply column named by `value_col` and the population column named by `pop_col` (for example a `build_food_supply()` output).
`value_col`	The unquoted per-capita supply column whose between-country inequality is measured. Defaults to `protein_g_cap_day`.
`pop_col`	The unquoted population column used as the inequality weight. Defaults to `population`.

Value

A tibble keyed by `year` with the `gini` coefficient.

Examples

```
calculate_food_gini(
  tibble::tribble(
    ~year, ~area_code, ~protein_g_cap_day, ~population,
    2000L, 10L, 40, 100,
    2000L, 20L, 40, 100,
    2000L, 30L, 40, 100
  )
)
```

`calculate_indirect_n2o_nh3` *Estimate indirect N2O from volatilised ammonia.*

Description

Converts the ammonia-N already volatilised (`calculate_nh3()`'s `nh3_n_t`) into indirect nitrous oxide. `"ipcc2019"` (the default) is the IPCC 2019 Tier 1 wet/dry `EF4`, read from ipcc_2019_soil_n_ef on each row's `moisture_regime`: 0.014 wet and 0.005 dry (IPCC 2019 Refinement Vol 4 Ch 11 Table 11.3 (Updated) p. 11.26), stamped `"ipcc2019_wet_dry"`. `"ipcc2019_atl_med"` is the Atlantic/Mediterranean option: Atlantic rows use the flat EF4 wet factor (`ef4_nh3_to_n2o_atl`, 0.014) and touch no emission-factor lookup; Mediterranean rows use the disaggregated n2o_efs_disaggregated `ef` on `(irrig_type, climate)` alone (`NH3_MgN * N2O_EF`), WITHOUT the fertiliser_n2o_modifiers `mf` that `calculate_soil_n2o()`'s `method = "aguilera"` applies to direct N2O. It is stamped `"ipcc2019_atl_med"`.

Usage

```
calculate_indirect_n2o_nh3(
  x,
  method = c("ipcc2019", "ipcc2019_atl_med"),
  example = FALSE
)
```

Arguments

`x`	A tibble with `nh3_n_t`. `method = "ipcc2019"` requires `moisture_regime` (`"wet"` or `"dry"`); `method = "ipcc2019_atl_med"` requires `climate` and, for MED rows, the `irrig_type` column n2o_efs_disaggregated is keyed on.
`method`	`"ipcc2019"` (default, IPCC 2019 Tier 1 wet/dry) or `"ipcc2019_atl_med"` (ATL flat EF4, MED disaggregated by irrigation).
`example`	If `TRUE`, return a small fixture instead of computing from `x`. Defaults to `FALSE`.

Details

`method = "ipcc2019"` aborts on an absent, missing or invalid `moisture_regime` exactly as `calculate_soil_n2o()` does.

Value

`x` with `n2o_indirect_nh3_n_t` and `method_n2o_indirect_nh3` appended.

Source

IPCC (2019) 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26.

Examples

```
calculate_indirect_n2o_nh3(example = TRUE)
```

---

calculate_livestock_emissions

Calculate all livestock emissions.

Description

Main dispatcher that runs the full IPCC 2019 livestock emissions pipeline: energy demand (Tier 2), enteric CH4, manure CH4, and manure N2O.

Selects tier automatically: Tier 2 when cohort-level data (weight, diet) are available; Tier 1 otherwise.

WHEP's Tier 2 energy balance covers cattle, buffalo, sheep and goats, the species ipcc_tier2_energy_coefs ships coefficients for. The IPCC 2019 Refinement (Vol. 4 Ch. 10, Table 10.9 (Updated)) suggests Tier 1 for camels, horses, mules and asses and swine, and develops no enteric method for poultry. At Tier 2 those species take Tier 1 by default, with a message and `"IPCC_2019_Tier1"` in each row's `method_*` columns, so a Tier 2 run covers the same animals as a Tier 1 run. `options$tier2_uncovered` selects `"leave_na"` or `"abort"` instead.

Usage

```
calculate_livestock_emissions(data, tier = NULL, options = list())
```

Arguments

`data`	Dataframe with at minimum `species` and `heads`. For Tier 2, also needs `cohort`, `weight` (or `iso3`), `diet_quality`, and production columns.
`tier`	Integer 1 or 2. If `NULL` (default), auto-selects based on data completeness.
`options`	A named list of manure-engine options. All but nine defaults reproduce the behaviour in force before whep#949. The exceptions are `enteric_ef_source`, `ef3_source` and `slurry_crust`, which since whep#601 read the 2019 Refinement's Tier 1 enteric factors and direct manure N2O EF3 and move Tier 1 enteric CH4 and both tiers' direct manure N2O, `mcf_source`, which moved from the shipped table to the 2019 Refinement in whep#1022 and does move Tier 2 manure CH4, `mms_shares`, which moved from the unsourced placeholder table to the GLEAM 2.0 ingest in whep#958 and does move both tiers' manure N2O, `pasture_bo`, which since whep#1137 pairs the 2019 pasture MCF with its published `Bo` and moves Tier 2 manure CH4, and `tier2_uncovered`, which since whep#1028 gives species with no Tier 2 method their Tier 1 values instead of `NA`, `indirect_n2o_source`, which since whep#1245 reads the 2019 Refinement's leaching factors and moves both tiers' indirect manure N2O, and `indirect_n2o_fractions`, which since whep#1365 weights the Table 10.22 fractions over the manure-management split and moves it again. `indirect_n2o_source` selects the edition of indirect_n2o_ef the indirect manure N2O reads:

- `"ipcc_2019"` (default): EF5 0.011 and FracLEACH-(H) 0.24, from Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26 of the 2019 Refinement, the current IPCC guidance and the EF5 the nitrogen balance already uses.
- `"ipcc_2006"`: EF5 0.0075 and FracLEACH-(H) 0.30, from Table 11.3, p. 11.24 of the 2006 Guidelines. These are the values WHEP shipped before whep#1245 under a 2019 citation, kept selectable so earlier figures stay reproducible.

EF4 (0.010) is the same under both. The edition also selects the Ch 11 FracGASM (0.21 in 2019, 0.20 in 2006) the pasture stream is priced at.

`indirect_n2o_fractions` selects which nitrogen fractions the indirect manure N2O volatilises and leaches:

- `"per_mms"` (default): IPCC 2019 Refinement, Vol 4, Ch 10, Eq. 10.26 and 10.27 (Updated), pp. 10.76-10.77. `FracGasMS` and `FracLeachMS` per animal category and system, from Table 10.22 (Updated), pp. 10.96-10.97, weighted over the row's own manure-management split, the same split the direct EF3 is weighted over. Pasture/range/paddock N, which Table 10.22 does not cover, is priced at the Ch 11 Table 11.3 FracGASM and FracLEACH-(H) (Eq. 11.9 and 11.10), as the IPCC prices grazing deposition; no other WHEP chain prices it, and the direct path prices the same N at EF3, so it is counted once. The managed-system fractions are the 2019 table's under either edition, as in `apply_management_losses()`: the 2006 Table 10.22 leaves most pairs blank.
- `"single"`: one FracGasMS 0.20 (assumed, unverified) and one FracLEACH-(H) on all excreted N, the form before whep#1365. Kept so earlier figures stay reproducible.

`method_manure_n2o` records both choices, e.g. `indirect_ipcc_2019_per_mms` or `indirect_ipcc_2006_single`.

`enteric_ef_source` selects the Tier 1 enteric CH4 emission factors. Although it sits among the manure-engine options, it governs the enteric path only:

- `"ipcc_2019"` (default): ipcc_enteric_ef_tier1, the 2019 Refinement's Tables 10.10 and 10.11 (Updated) as published: cattle and buffalo by IPCC region, and sheep, goats and swine at the productivity system the Table 10.10 footnote assigns the region (high in North America, Europe and Oceania, low elsewhere). A row the edition prices at nothing keeps the as-shipped factor, stamped `enteric_ef_as_shipped_fallback`.
- `"as_shipped"`: ipcc_2019_enteric_ef_cattle and ipcc_2019_enteric_ef_other, the 2006 Guidelines' factors with four cells in neither edition, read before whep#601. Kept so earlier figures stay reproducible.

`method_enteric` records which (`enteric_ef_ipcc_2019` or `enteric_ef_as_shipped`). On FAOSTAT 2020 heads the default raises Tier 1 enteric CH4 from 110.8 to 125.3 Tg. Taking high productivity everywhere instead would give 134.2 Tg, low everywhere 124.2 Tg.

`ef3_source` selects the direct manure N2O emission factors, at both tiers: `"ipcc_2019"` (default) or `"as_shipped"`, the two editions of ipcc_manure_ef3.

`slurry_crust` sets the share of liquid slurry with a natural crust, which mixes the 2019 Table 10.21 (Updated) EF3 with a crust (0.005) and without one (0) into the engine's single `"Liquid/Slurry"` label: `"half"` (default, 0.0025, GLEAM 2.0's stated assumption for its pig and chicken slurry, extended to the other species), `"none"` (0) or `"all"` (0.005). It has no effect under `ef3_source = "as_shipped"`, whose 0.002 is a single row.

`method_manure_n2o` records the EF3 choice (`ef3_ipcc_2019; slurry_crust_half` or `ef3_as_shipped`). On FAOSTAT 2020 heads the default lowers Tier 1 direct manure N2O from 1.523 to 1.178 Tg.

`mms_shares` selects which half of regional_mms_distribution the split is read from: `"gleam_2_0"` (default) is the GLEAM 2.0 Supplement S1 Tab. 4.2-4.11 ingest, `"placeholder"` the unsourced table it replaced in whep#958. The placeholder stays selectable so the values WHEP published before that ingest remain reproducible and the sensitivity to it stays measurable; it is not a defensible alternative estimate.

`mms_region` selects how the manure-management split in regional_mms_distribution is keyed:

- `"as_available"` (default): a row uses its own region when the frame already carries a `region` column, and the `region == "Global"` split otherwise. Tier 1 resolves a region for the (sourced) per-head N-excretion table and so takes the region-specific split; Tier 2 carries no region and so takes the Global one.
- `"resolve"`: the IPCC region is resolved from `iso3`, `area_code` or `polity_area_code` where it is missing, which makes the table's region-specific rows live on the Tier 2 path too. Opt-in because it changes which rows of the table apply, not because the rows are doubtful: since whep#958 they are the GLEAM 2.0 ingest.
- `"global"`: every row takes the `region == "Global"` split, whatever region column it carries.

`mcf_source` selects which methane conversion factor table the Tier 2 manure CH4 weighting reads:

- `"ipcc_2019"` (default): the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 (Updated) of the 2019 Refinement, which is the current IPCC guidance.
- `"ipcc_2006"`: the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 of the 2006 Guidelines.
- `"as_shipped"`: climate_mcf, whose live rows are predominantly the 2006 Guidelines Table 10.17 but with six cells that match no published IPCC value (whep#601, whep#1022). Kept selectable so an older run can be reproduced; it is no longer the default, because values whose provenance could not be established should not be what ships.

Neither edition publishes one number per Cool/Temperate/Warm zone for every system, so both as-published tables apply a stated collapse rule; see climate_mcf_ipcc. Both rules are WHEP's, not the IPCC's, and the default makes them live. `method_manure_ch4` records the table used.

`pasture_bo` selects the methane potential (`Bo`) the Tier 2 manure CH4 prices the pasture/range/paddock stream at. The 2019 Refinement publishes its single 0.47 percent pasture MCF as half of a pair that "must always be used in conjunction with a B0 value of 0.19" (Vol 4, Ch 10, Table 10.17 (Updated) footnote 2, p. 10.70; Section 10.4.2, p. 10.66); the pair is the `paired_bo_m3_kg_vs` column of climate_mcf_ipcc (whep#1137).

- `"paired"` (default): a stream whose MCF row carries a paired `Bo` is priced at it, every other stream at the animal-category `Bo` of ipcc_tier2_bo_values. Only `mcf_source = "ipcc_2019"` publishes a pair, so under the other two sources this changes nothing.
- `"species"`: every stream takes the animal-category `Bo`, the behaviour before whep#1137. Under `"ipcc_2019"` that is the hybrid the Refinement rejects; kept so the earlier figures stay reproducible and the sensitivity to the pairing stays measurable.

`method_manure_ch4` records per row which applied (`pasture_bo_paired` or `pasture_bo_species`) wherever the row has manure on a stream that carries a published pair.

`climate_source` selects where the climate zone the methane conversion factors in the MCF table are read at comes from. A `climate_zone` a row already carries is always used and stamped `climate_from_data`; the option governs only the rows left without one, whether that is a hole in a supplied column or a wholly absent column.

- `"assumed"` (default): fill with `assumed_climate_zone`.
- `"from_data"`: abort instead of assuming.

`assumed_climate_zone` is the zone `"assumed"` fills in: `"Cool"`, `"Temperate"` (default) or `"Warm"`. It is an assumption, not a measurement; `method_manure_ch4` records per row which of the sources applied, and this argument exists so the sensitivity to the assumption can be measured (whep#949).

`tier2_uncovered` says what the Tier 2 path does with a species WHEP has no Tier 2 method for. The energy balance needs the maintenance and activity coefficients of Tables 10.4 and 10.5, which ipcc_tier2_energy_coefs ships for cattle, buffalo, sheep and goats only; the 2019 Refinement itself suggests Tier 1 for camels, horses, mules and asses and swine and has no enteric method for poultry (Vol. 4 Ch. 10, Table 10.9 (Updated)), and its Tier 2 manure equations for swine and poultry need a country-specific dry-matter intake (Equation 10.32A) that WHEP does not hold. Although it sits among the manure-engine options, it governs the enteric path too.

- `"tier1"` (default): those species take the Tier 1 enteric CH4, manure CH4 and manure N2O, written into the Tier 2 output columns and stamped `"IPCC_2019_Tier1"` in `method_enteric`, `method_manure_ch4` and `method_manure_n2o`, with a message naming them. This is the IPCC's own suggested method for them, so a Tier 2 inventory keeps the whole herd rather than silently covering fewer animals than Tier 1.
- `"leave_na"`: they keep `NA` emissions, the behaviour before whep#1028,

with a warning naming them. Kept so a ruminant-only Tier 2 figure stays reproducible.

- `"abort"`: any such species aborts, naming it.

Value

Dataframe with all emission columns, method tracking, and original data columns preserved.

Examples

```
tibble::tibble(
  species = "Dairy Cattle",
  cohort = "Adult Female",
  heads = 1000,
  weight = 600,
  diet_quality = "High",
  milk_yield_kg_day = 20
) |>
  calculate_livestock_emissions() |>
  dplyr::select(species, cohort, heads,
    enteric_ch4_tier2, manure_ch4_tier2,
    manure_n2o_total)
```

---

calculate_lmdi *Calculate LMDI decomposition.*

Description

Performs LMDI (Log Mean Divisia Index) decomposition analysis with flexible identity parsing, automatic factor detection, and support for multiple periods and groupings. Supports sectoral decomposition using bracket notation for both summing and grouping operations.

Usage

```
calculate_lmdi(
  data,
  identity,
  identity_labels = NULL,
  time_var = year,
  periods = NULL,
  periods_2 = NULL,
  .by = NULL,
  rolling_mean = 1,
  output_format = "clean",
  verbose = TRUE,
  zero_method = c("limit", "small_value")
)
```

Arguments

`data`	A data frame containing the variables for decomposition. Must include all variables specified in the identity, time variable, and any grouping variables.
`identity`	Character. Decomposition identity in format `"target:factor1*factor2*..."`. The target appears before the colon, factors after, separated by asterisks. Supports explicit ratios with `/` and structural decomposition with `[]`.
`identity_labels`	Character vector. Custom labels for factors to use in output instead of variable names. The first element labels the target, and subsequent elements label each factor in order. Default: NULL uses variable names as-is.
`time_var`	Unquoted name of the time variable column in the data. Default: `year`. Must be numeric or coercible to numeric.
`periods`	Numeric vector. Years defining analysis periods. Each consecutive pair defines one period. Default: NULL uses all available years.
`periods_2`	Numeric vector. Additional period specification for complex multi-period analyses. Default: NULL.
`.by`	Character vector. Grouping variables for performing separate decompositions. Default: NULL (single decomposition for all data).
`rolling_mean`	Numeric. Window size for rolling mean smoothing applied before decomposition. Default: 1 (no smoothing).
`output_format`	Character. Format of output data frame. Options: `"clean"` (default) or `"total"`.
`verbose`	Logical. If TRUE (default), prints progress messages during decomposition.
`zero_method`	Character. How zero values are handled, since the logarithmic mean weight is undefined at zero. Missing values (including rows added when balancing the panel) are treated as zeros by both methods. One of: • `"limit"` (default): the analytical limit of the small-value strategy as the replacement value tends to zero (Ang and Liu, 2007). Zeros are kept as zeros and each affected sub-category's change is attributed exactly, so the additive decomposition is perfect and does not depend on an arbitrary constant. See the *Zero values* section. • `"small_value"`: the previous behaviour. Every zero is replaced by `1e-12` before decomposing (the small-value strategy of Ang, Zhang and Choi, 1998; the constant itself is the package's historical choice, assumed, unverified against a source), and for identities whose factors are all plain columns the target is re-derived as the product of the factors. Kept for comparison and sensitivity analysis. The chosen method is recorded in the `method_zero_handling` column.

Details

The LMDI method decomposes changes in a target variable into contributions from multiple factors using logarithmic mean weights. This implementation supports:

Flexible identity specification:

- Automatic factor detection from identity string.
- Support for ratio calculations (implicit division).
- Sectoral aggregation with `[]` notation.
- Sectoral grouping with `{}` notation.

Period analysis: The function can decompose changes over single or multiple periods. Periods are defined by consecutive pairs in the `periods` vector.

Grouping capabilities: Use `.by` to perform separate decompositions for different groups (e.g., countries, regions) while maintaining consistent factor structure.

Value

A tibble with LMDI decomposition results containing:

- Time variables and grouping variables (if specified).
- `additive`: Additive contributions (sum equals total change in target).
- `multiplicative`: Multiplicative indices (product equals target ratio).
- `multiplicative_log`: Log of multiplicative indices.
- `method_zero_handling`: the `zero_method` used.
- Period identifiers and metadata.

Identity Syntax

The identity parameter uses a special syntax to define decomposition:

Basic format: `"target:factor1*factor2*factor3"`

Simple decomposition (no sectors):

- Basic: `"emissions:gdp*(emissions/gdp)"`
- Complete: `"emissions:(emissions/gdp)*(gdp/population)*population"`

Understanding bracket notation:

Square brackets `[]` specify variables to sum across categories, enabling structural decomposition. The bracket aggregates values BEFORE calculating ratios.

Single-level structural decomposition:

- `"emissions:activity*(activity[sector]/activity)*(emissions[sector]/activity[sector])"`
- Creates 3 factors: Activity level, Sectoral structure, Sectoral intensity.

Multi-level structural decomposition:

- Two levels: `"emissions:activity*(activity[sector]/activity)*(activity[sector+fuel]/activity[s`
- Creates 4 factors: Activity level, Sector structure, Fuel structure, Sectoral-fuel intensity.

Data Requirements

The input data frame must contain:

- All variables mentioned in the identity.
- The time variable (default: "year").
- Grouping variables if using `.by`.
- No missing values in key variables for decomposition periods.

The panel is balanced (and smoothed, with `rolling_mean`) over its key columns: every non-numeric column, the bracket selectors and the `.by` columns. Any other numeric column is carried along as a value and is not a key, so a numeric identifier that must keep its series apart belongs in `.by` or a selector.

Zero values

With `zero_method = "limit"`, a zero is treated as a quantity $\delta \to 0$. Every aggregate a factor is built from is then of the form $c\delta^p$ (a sum is of order δ only when all its terms are zero; a ratio subtracts the orders). For a sub-category with target change ΔV and factor k of order q_{k0} at the start and q_{kT} at the end, the additive contribution tends to:

- the usual LMDI-I term when neither target value is zero;
- $\Delta V \Delta q_k / \sum_j \Delta q_j$, with $\Delta q_k = q_{kT} - q_{k0}$, when exactly one of them is zero: the whole change of that sub-category goes to the factor(s) that reach or leave zero, split by their order when several do;
- zero when the target is zero at both ends.

These sum exactly to the change in the target, which is what `"small_value"` approaches only slowly (its error decays like $1/\log(1/\epsilon)$): with `1e-12` a factor that is zero at the start still leaves a few percent of the change on the other factors. Ang and Liu (2007) show that the small-value results converge as the replacement value tends to zero; the limits above are derived here from that convergence with one common replacement value for every zero (their tables were not accessible when this was written, so the term-by-term match with them is assumed, unverified). A sub-category whose target stays positive while one of its factors goes to zero and another to infinity (for example emissions with no activity), or whose target reaches or leaves zero while none of its factors does (data breaking the identity), has no finite decomposition: its contributions are returned as `NA` with a warning. The multiplicative form is undefined when the aggregate target is zero at either end, so its factor indices are then `NA`.

References

Ang, B. W., Zhang, F. and Choi, K. (1998). Factorizing changes in energy and environmental indicators through decomposition. *Energy*, 23(6), 489-495. doi:10.1016/S0360-5442(98)000164

Ang, B. W. and Liu, N. (2007). Handling zero values in the logarithmic mean Divisia index decomposition approach. *Energy Policy*, 35(1), 238-246. doi:10.1016/j.enpol.2005.11.001

Ang, B. W. (2015). LMDI decomposition approach: A guide for implementation. *Energy Policy*, 86, 233-238. doi:10.1016/j.enpol.2015.07.007

Examples

```
# In these examples, 'activity' is a measure of scale
# (e.g., GDP in million USD) and 'intensity' is the target
# variable per unit activity (e.g., emissions per million USD).
# The units are illustrative; adapt to your context.
# --- Shared sample data ---
data_simple <- tibble::tribble(
  ~year, ~activity, ~intensity, ~emissions,
  2010,  1000,      0.10,       100,
  2011,  1100,      0.12,       132,
  2012,  1200,      0.09,       108,
  2013,  1300,      0.10,       130
)

# --- 1. Year-over-year decomposition (default) ---
# Decompose annual emission changes into activity and intensity effects.
# The additive column sums to the total change in emissions each period.
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 2. Single baseline-to-end period ---
# Pass a two-element periods vector to get a single cumulative period
# instead of year-over-year results.
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  time_var = year,
  periods = c(2010, 2013),
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 3. Year-over-year AND one cumulative summary period ---
# Use periods_2 to append an extra comparison period alongside the
# year-over-year results.
```

```
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  time_var = year,
  periods = c(2010, 2011, 2012, 2013),
  periods_2 = c(2010, 2013),
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 4. Per-country decomposition with .by ---
# Separate LMDI runs per country; results are stacked with a country column.
data_countries <- tibble::tribble(
  ~year, ~country, ~activity, ~intensity, ~emissions,
  2010, "ESP", 1000, 0.10, 100,
  2011, "ESP", 1100, 0.11, 121,
  2012, "ESP", 1200, 0.10, 120,
  2010, "FRA", 2000, 0.05, 100,
  2011, "FRA", 2200, 0.05, 110,
  2012, "FRA", 2400, 0.05, 120
)

calculate_lmdi(
  data_countries,
  identity = "emissions:activity*intensity",
  time_var = year,
  .by = "country",
  verbose = FALSE
) |>
  dplyr::select(
    country,
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 5. Ratio notation ---
# Express factors as explicit ratios (e.g. intensity = emissions/activity).
# Factor labels in the output preserve the ratio form for clarity.
calculate_lmdi(
  data_simple,
  identity = "emissions:(emissions/activity)*activity",
  time_var = year,
  verbose = FALSE
) |>
```

```
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 6. Structural (sectoral) decomposition with [] notation ---
# Decomposes emissions into:
#   total_activity * sector_structure * sector_intensity
# [] sums the bracketed variable across sector before forming ratios,
# enabling proper structural decomposition.
data_sectors <- tibble::tribble(
  ~year, ~sector, ~activity, ~emissions,
  2010, "industry", 600, 60,
  2010, "transport", 400, 40,
  2011, "industry", 700, 63,
  2011, "transport", 500, 55
) |>
  dplyr::group_by(year) |>
  dplyr::mutate(total_activity = sum(activity)) |>
  dplyr::ungroup()

calculate_lmdi(
  data_sectors,
  identity = paste0(
    "emissions:",
    "total_activity*",
    "(activity[sector]/total_activity)*",
    "(emissions[sector]/activity[sector])"
  ),
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 7. Custom factor labels ---
# Replace raw variable names with readable labels for reporting.
# Supply one label per term (target first, then each factor in order).
calculate_lmdi(
  data_simple,
  identity = "emissions:activity*intensity",
  identity_labels = c(
    "Total Emissions",
    "Activity Effect",
    "Intensity Effect"
```

```
  ),
  time_var = year,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )

# --- 8. Rolling mean smoothing before decomposition ---
# A 3-year rolling mean reduces noise in volatile series before
# computing LMDI weights. Edge years use partial windows (fewer
# than k observations) so no periods are lost.
data_smooth <- tibble::tibble(
  year      = 2010:2020,
  activity  = seq(1000, 2000, length.out = 11),
  intensity = rep(0.1, 11),
  emissions = seq(1000, 2000, length.out = 11) * 0.1
)

calculate_lmdi(
  data_smooth,
  identity = "emissions:activity*intensity",
  time_var = year,
  rolling_mean = 3,
  verbose = FALSE
) |>
  dplyr::select(
    period,
    component_type,
    factor_label,
    additive,
    multiplicative
  )
```

`calculate_manner_nh3` *Estimate ammonia-N volatilisation with the MANNER process-based model.*

Description

Ports the MANNER (Nicholson et al. 2013) process-based ammonia volatilisation model. Dispatches on `fertiliser` to one of two independent paths: a synthetic-fertiliser path (`"Urea"`, `"AN"`, `"CAN"`, `"AS"`) driven by soil pH, application rate, rainfall and temperature, and an organic-manure path (`"cattle_slurry"`, `"pig_slurry"`, `"FYM"`, `"poultry_manure"`, `"urban"`) driven by rainfall, temperature, wind speed, application technique, system (arable/grassland) and incorporation delay.

Usage

```
calculate_manner_nh3(
  n_applied_t = NULL,
  fertiliser = NULL,
  drivers = list(),
  example = FALSE,
  temp_method = c("misselbrook_2004", "source_port")
)
```

Arguments

`n_applied_t`	Numeric, nitrogen applied (t).
`fertiliser`	One of `"Urea"`, `"AN"`, `"CAN"`, `"AS"` (synthetic path) or `"cattle_slurry"`, `"pig_slurry"`, `"FYM"`, `"poultry_manure"`, `"urban"` (organic path).
`drivers`	A named list of driver values. Synthetic path: `soil_ph` (numeric soil pH), `rate_kg_ha` (numeric N application rate, kg N/ha), `rainfall_mm` (numeric period precipitation, mm), `irrigated` (logical), `temp_c` (numeric application-period temperature, deg C), `temp_c_annual_mean` (numeric annual mean temperature, deg C; not used for Urea). Organic path: `rainfall_mm`, `irrigated`, `windspeed_ms` (numeric wind speed, m/s), `technique` (one of the six manner_params `technique` keys), `system` (`"Arable"` or `"Grassland"`), `temp_c`, `incorporation_delay_h` (numeric hours between surface application and soil incorporation, or `Inf`/`NA` for no incorporation), `species` (optional; one of the eight manure_inorganic_n species, used only to look up `inorganic_n_fraction`; when omitted it falls back to the manure type's default species, and it is ignored entirely for `"urban"`).
`example`	If `TRUE`, return a small fixture instead of computing from drivers. Defaults to `FALSE`.
`temp_method`	Synthetic path only: which temperature-equation form AN and AS use, `"misselbrook_2004"` (default) or `"source_port"`. See Details.

Details

The organic path's `inorganic_n_fraction` lookup (manure_inorganic_n) maps `manure_type` to its ammoniacal `manure_stream` (`"Liquid"` for `"cattle_slurry"`/`"pig_slurry"`, `"Solid"` for `"FYM"`/`"poultry_manure"`) and reads that stream's fraction for the actual `species` driver supplied by the caller. This matches the source implementation, which maps every species' solid stream to the FYM MANNER class yet looks the inorganic-N fraction up per real species. When `species` is omitted it falls back to the manure type's default species (Cattle for `"cattle_slurry"`/`"FYM"`, Pigs for `"pig_slurry"`, Poultry for `"poultry_manure"`). `"urban"` bypasses this lookup entirely: it fixes `inorganic_n_fraction = 0.5` regardless of species, matching the source implementation. For the AG availability and incorporation factors, `"urban"` maps to the FYM manure class, as the source coefficient table does, including the 0.4 Org_ef correction.

The synthetic path's temperature factor (Misselbrook et al. 2004, doi:10.1079/SUM2004280) is an exponential scaled on UK monthly temperatures. It is capped at 1, as the UK ammonia inventory that runs this model caps it (Misselbrook et al. 2015, Inventory of Ammonia

Emissions from UK Agriculture 2014). Without the cap, warm application months returned emission factors above 1. With it, `ef` never exceeds the fertiliser's `max_nh3` in manner_params.

Two forms of that factor exist. Urea always takes the absolute form, `exp(0.1386 (temp_c - 8.4)) / 3`; CAN always takes the anomaly form, `exp(0.2197225 (temp_c - temp_c_annual_mean)) / 3`. AN and AS depend on `temp_method`:

- `"misselbrook_2004"` (default): AN takes the anomaly form on every soil, and AS takes the anomaly form on non-calcareous soils and the absolute form on calcareous soils (`soil_ph > 7`). This is the assignment of Misselbrook et al. (2004, p. 367): eq. 3 (anomaly) "for AN and other N to all soils and AS/DAP to non-calcareous soils", eq. 4 (absolute) "for urea and UAN to all soils and AS/DAP to calcareous soils". The UK inventory agrees for AS, which it treats on calcareous soils "as for urea" (Misselbrook et al. 2015, notes to Table A11).
- `"source_port"`: the forms of the regional implementation this model was ported from. AN takes the absolute form, and AS takes the anomaly form on every soil.

Value

A tibble with `n_applied_t`, `ef` (realised emission factor), `nh3_n_t` and `method_manner`. The synthetic path also returns `method_manner_temp`, the `temp_method` used.

Examples

```
calculate_manner_nh3(example = TRUE)
```

---

calculate_manner_nh3_default

Estimate ammonia-N volatilisation with MANNER's gross-default technique and incorporation-delay blend.

Description

A thin wrapper around `calculate_manner_nh3()`'s organic-manure path that fills in `technique` and `incorporation_delay_h` from manner_default_technique_mix instead of requiring them as caller- supplied drivers. It calls `calculate_manner_nh3()` once per row of manner_default_technique_mix and combines the results into a single share-weighted emission factor. Use this only where real per-cell/per-era application-technique survey data does not exist (see Details).

Usage

```
calculate_manner_nh3_default(
  n_applied_t = NULL,
  fertiliser = NULL,
  drivers = list(),
  example = FALSE
)
```

Arguments

`n_applied_t`	Numeric, nitrogen applied (t).
`fertiliser`	One of `"cattle_slurry"`, `"pig_slurry"`, `"FYM"`, `"poultry_manure"`, `"urban"` (the `calculate_manner_nh3()` organic-manure path; the gross default only applies there).
`drivers`	A named list of driver values, as in `calculate_manner_nh3()`'s organic path, EXCEPT `technique` and `incorporation_delay_h` are neither required nor used (they come from manner_default_technique_mix instead).
`example`	If `TRUE`, return a small fixture instead of computing from drivers. Defaults to `FALSE`.

Details

manner_default_technique_mix is a deliberate, permanent gross- assumption default, not a region/era-specific survey. It fixes `technique = "Broadcast"` on every row, matching the production MANNER run of the earlier regional historical reconstruction this model is ported from, which itself applies the Broadcast application-technique factor unconditionally to its whole national run, with no region/era variation. For incorporation delay, it blends four of manner_incorporation_factor's `delay_bin` categories in equal 25% shares: 25% of applied nitrogen assumed never incorporated (`incorporation_delay_h = NA`, matching how `calculate_manner_nh3()` treats a missing delay as `"No incorporation"`), 25% incorporated within 2 hours (`incorporation_delay_h = 2`), 25% within 12-24 hours (`incorporation_delay_h = 24`) and 25% within 1-2 days (`incorporation_delay_h = 48`). Only `ef` and `nh3_n_t` are share-weighted across the four blend calls; `n_applied_t` is carried through as-is, not re-weighted, since the full `n_applied_t` is assumed split across the four incorporation-delay scenarios rather than quadruplicated.

Value

A tibble with `n_applied_t`, `ef` (share-weighted realised emission factor), `nh3_n_t` and `method_manner`.

Examples

```
calculate_manner_nh3_default(example = TRUE)
```

---

calculate_manure_emissions

Calculate manure emissions (CH4 + N2O).

Description

Wrapper that selects Tier 1 or 2 for manure CH4 and computes N2O (Tier 2 only; skipped for Tier 1).

Usage

```
calculate_manure_emissions(data, tier = NULL, options = list())
```

Arguments

`data`	Dataframe with `species`, `heads`. For Tier 2, also needs `cohort`, `weight`, and `diet_quality`. For Tier 1, `iso3` is used to select regional emission factors.
`tier`	Integer 1 or 2. If `NULL` (default), auto-selects based on data completeness.
`options`	A named list of manure-engine options. All but nine defaults reproduce the behaviour in force before whep#949. The exceptions are `enteric_ef_source`, `ef3_source` and `slurry_crust`, which since whep#601 read the 2019 Refinement's Tier 1 enteric factors and direct manure N2O EF3 and move Tier 1 enteric CH4 and both tiers' direct manure N2O, `mcf_source`, which moved from the shipped table to the 2019 Refinement in whep#1022 and does move Tier 2 manure CH4, `mms_shares`, which moved from the unsourced placeholder table to the GLEAM 2.0 ingest in whep#958 and does move both tiers' manure N2O, `pasture_bo`, which since whep#1137 pairs the 2019 pasture MCF with its published `Bo` and moves Tier 2 manure CH4, and `tier2_uncovered`, which since whep#1028 gives species with no Tier 2 method their Tier 1 values instead of `NA`, `indirect_n2o_source`, which since whep#1245 reads the 2019 Refinement's leaching factors and moves both tiers' indirect manure N2O, and `indirect_n2o_fractions`, which since whep#1365 weights the Table 10.22 fractions over the manure-management split and moves it again. `indirect_n2o_source` selects the edition of indirect_n2o_ef the indirect manure N2O reads: • `"ipcc_2019"` (default): EF5 0.011 and FracLEACH-(H) 0.24, from Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26 of the 2019 Refinement, the current IPCC guidance and the EF5 the nitrogen balance already uses. • `"ipcc_2006"`: EF5 0.0075 and FracLEACH-(H) 0.30, from Table 11.3, p. 11.24 of the 2006 Guidelines. These are the values WHEP shipped before whep#1245 under a 2019 citation, kept selectable so earlier figures stay reproducible. EF4 (0.010) is the same under both. The edition also selects the Ch 11 FracGASM (0.21 in 2019, 0.20 in 2006) the pasture stream is priced at. `indirect_n2o_fractions` selects which nitrogen fractions the indirect manure N2O volatilises and leaches: • `"per_mms"` (default): IPCC 2019 Refinement, Vol 4, Ch 10, Eq. 10.26 and 10.27 (Updated), pp. 10.76-10.77. `FracGasMS` and `FracLeachMS` per animal category and system, from Table 10.22 (Updated), pp. 10.96-10.97, weighted over the row's own manure-management split, the same split the direct EF3 is weighted over. Pasture/range/paddock N, which Table 10.22 does not cover, is priced at the Ch 11 Table 11.3 FracGASM and FracLEACH-(H) (Eq. 11.9 and 11.10), as the IPCC

prices grazing deposition; no other WHEP chain prices it, and the direct path prices the same N at EF3, so it is counted once. The managed-system fractions are the 2019 table's under either edition, as in `apply_management_losses()`: the 2006 Table 10.22 leaves most pairs blank.

- `"single"`: one FracGasMS 0.20 (assumed, unverified) and one FracLEACH-(H) on all excreted N, the form before whep#1365. Kept so earlier figures stay reproducible.

`method_manure_n2o` records both choices, e.g. `indirect_ipcc_2019_per_mms` or `indirect_ipcc_2006_single`.

`enteric_ef_source` selects the Tier 1 enteric CH4 emission factors. Although it sits among the manure-engine options, it governs the enteric path only:

- `"ipcc_2019"` (default): ipcc_enteric_ef_tier1, the 2019 Refinement's Tables 10.10 and 10.11 (Updated) as published: cattle and buffalo by IPCC region, and sheep, goats and swine at the productivity system the Table 10.10 footnote assigns the region (high in North America, Europe and Oceania, low elsewhere). A row the edition prices at nothing keeps the as-shipped factor, stamped `enteric_ef_as_shipped_fallback`.
- `"as_shipped"`: ipcc_2019_enteric_ef_cattle and ipcc_2019_enteric_ef_other, the 2006 Guidelines' factors with four cells in neither edition, read before whep#601. Kept so earlier figures stay reproducible.

`method_enteric` records which (`enteric_ef_ipcc_2019` or `enteric_ef_as_shipped`). On FAOSTAT 2020 heads the default raises Tier 1 enteric CH4 from 110.8 to 125.3 Tg. Taking high productivity everywhere instead would give 134.2 Tg, low everywhere 124.2 Tg.

`ef3_source` selects the direct manure N2O emission factors, at both tiers: `"ipcc_2019"` (default) or `"as_shipped"`, the two editions of ipcc_manure_ef3.

`slurry_crust` sets the share of liquid slurry with a natural crust, which mixes the 2019 Table 10.21 (Updated) EF3 with a crust (0.005) and without one (0) into the engine's single `"Liquid/Slurry"` label: `"half"` (default, 0.0025, GLEAM 2.0's stated assumption for its pig and chicken slurry, extended to the other species), `"none"` (0) or `"all"` (0.005). It has no effect under `ef3_source = "as_shipped"`, whose 0.002 is a single row.

`method_manure_n2o` records the EF3 choice (`ef3_ipcc_2019; slurry_crust_half` or `ef3_as_shipped`). On FAOSTAT 2020 heads the default lowers Tier 1 direct manure N2O from 1.523 to 1.178 Tg.

`mms_shares` selects which half of regional_mms_distribution the split is read from: `"gleam_2_0"` (default) is the GLEAM 2.0 Supplement S1 Tab. 4.2-4.11 ingest, `"placeholder"` the unsourced table it replaced in whep#958. The placeholder stays selectable so the values WHEP published before that ingest remain reproducible and the sensitivity to it stays measurable; it is not a defensible alternative estimate.

`mms_region` selects how the manure-management split in regional_mms_distribution is keyed:

- `"as_available"` (default): a row uses its own region when the frame already carries a `region` column, and the `region == "Global"` split otherwise. Tier 1 resolves a region for the (sourced) per-head N-excretion table and so takes the region-specific split; Tier 2 carries no region and so takes the Global one.
- `"resolve"`: the IPCC region is resolved from `iso3`, `area_code` or `polity_area_code` where it is missing, which makes the table's region-specific rows live on the Tier 2 path too. Opt-in because it changes which rows of the table apply, not because the rows are doubtful: since whep#958 they are the GLEAM 2.0 ingest.
- `"global"`: every row takes the `region == "Global"` split, whatever region column it carries.

`mcf_source` selects which methane conversion factor table the Tier 2 manure CH4 weighting reads:

- `"ipcc_2019"` (default): the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 (Updated) of the 2019 Refinement, which is the current IPCC guidance.
- `"ipcc_2006"`: the matching `edition` rows of climate_mcf_ipcc, read off Table 10.17 of the 2006 Guidelines.
- `"as_shipped"`: climate_mcf, whose live rows are predominantly the 2006 Guidelines Table 10.17 but with six cells that match no published IPCC value (whep#601, whep#1022). Kept selectable so an older run can be reproduced; it is no longer the default, because values whose provenance could not be established should not be what ships.

Neither edition publishes one number per Cool/Temperate/Warm zone for every system, so both as-published tables apply a stated collapse rule; see climate_mcf_ipcc. Both rules are WHEP's, not the IPCC's, and the default makes them live. `method_manure_ch4` records the table used.

`pasture_bo` selects the methane potential (`Bo`) the Tier 2 manure CH4 prices the pasture/range/paddock stream at. The 2019 Refinement publishes its single 0.47 percent pasture MCF as half of a pair that "must always be used in conjunction with a B0 value of 0.19" (Vol 4, Ch 10, Table 10.17 (Updated) footnote 2, p. 10.70; Section 10.4.2, p. 10.66); the pair is the `paired_bo_m3_kg_vs` column of climate_mcf_ipcc (whep#1137).

- `"paired"` (default): a stream whose MCF row carries a paired `Bo` is priced at it, every other stream at the animal-category `Bo` of ipcc_tier2_bo_values. Only `mcf_source = "ipcc_2019"` publishes a pair, so under the other two sources this changes nothing.
- `"species"`: every stream takes the animal-category `Bo`, the behaviour before whep#1137. Under `"ipcc_2019"` that is the hybrid the Refinement rejects; kept so the earlier figures stay reproducible and the sensitivity to the pairing stays measurable.

`method_manure_ch4` records per row which applied (`pasture_bo_paired` or `pasture_bo_species`) wherever the row has manure on a stream that carries a published pair.

`climate_source` selects where the climate zone the methane conversion factors in the MCF table are read at comes from. A `climate_zone` a row already carries is always used and stamped `climate_from_data`; the option governs only the rows left without one, whether that is a hole in a supplied column or a wholly absent column.

- `"assumed"` (default): fill with `assumed_climate_zone`.
- `"from_data"`: abort instead of assuming.

`assumed_climate_zone` is the zone `"assumed"` fills in: `"Cool"`, `"Temperate"` (default) or `"Warm"`. It is an assumption, not a measurement; `method_manure_ch4` records per row which of the sources applied, and this argument exists so the sensitivity to the assumption can be measured (whep#949).

`tier2_uncovered` says what the Tier 2 path does with a species WHEP has no Tier 2 method for. The energy balance needs the maintenance and activity coefficients of Tables 10.4 and 10.5, which ipcc_tier2_energy_coefs ships for cattle, buffalo, sheep and goats only; the 2019 Refinement itself suggests Tier 1 for camels, horses, mules and asses and swine and has no enteric method for poultry (Vol. 4 Ch. 10, Table 10.9 (Updated)), and its Tier 2 manure equations for swine and poultry need a country-specific dry-matter intake (Equation 10.32A) that WHEP does not hold. Although it sits among the manure-engine options, it governs the enteric path too.

- `"tier1"` (default): those species take the Tier 1 enteric CH4, manure CH4 and manure N2O, written into the Tier 2 output columns and stamped `"IPCC_2019_Tier1"` in `method_enteric`, `method_manure_ch4` and `method_manure_n2o`, with a message naming them. This is the IPCC's own suggested method for them, so a Tier 2 inventory keeps the whole herd rather than silently covering fewer animals than Tier 1.
- `"leave_na"`: they keep `NA` emissions, the behaviour before whep#1028, with a warning naming them. Kept so a ruminant-only Tier 2 figure stays reproducible.
- `"abort"`: any such species aborts, naming it.

Value

Dataframe with all input columns preserved, plus:

- `method_manure_ch4`: tracking label.
- `method_mms`: which half of regional_mms_distribution was read and how it was keyed, `"<shares>/<keying>"` (e.g. `"gleam_2_0/region_specific"`).
- Tier 1: `manure_ef_kgch4`, `manure_ch4_tier1`.
- Tier 2: `volatile_solids`, `methane_potential`, `weighted_mcf`, `manure_ch4_per_head`, `manure_ch4_tier2`.
- N2O (both tiers): `method_manure_n2o`, `n_excretion`, `manure_n2o_direct`, `manure_n2o_indirect`, `manure_n2o_total`. Tier 1 uses default per-head excretion rates; Tier 2 uses the energy/nitrogen balance.

- `manure_n2o_direct_pasture`, `manure_n2o_indirect_pasture`: the part of `manure_n2o_direct` and `manure_n2o_indirect` (kg N2O) from the N deposited by grazing animals on pasture, range and paddock, priced at EF3PRP and the Ch 11 FracGASM and FracLEACH-(H). Both are already inside `manure_n2o_total`, never added to it. The IPCC reports this N under managed soils (2019 Refinement Vol 4 Ch 11, F_PRP), not manure management, and `build_nitrogen_balance()` prices the same N as a field loss on its excreta rows. WHEP's GHG footprint takes it from this chain only (`build_livestock_ghg_extension()`; `build_crop_soil_n2o_extension()` leaves F_PRP out), so subtract these columns from `manure_n2o_total` before adding any managed-soils figure that includes grazing N, or for a manure-management-only total comparable to an inventory's.

Examples

```
tibble::tibble(
  species = "Cattle", heads = 1000, iso3 = "DEU"
) |>
  calculate_manure_emissions(tier = 1)
```

`calculate_n_leaching` *Estimate nitrate leaching, topsoil denitrification and indirect N2O.*

Description

Two methods for partitioning a nitrogen surplus into leached nitrate and topsoil-denitrified nitrogen. `"meisinger_drainage"` (the default) is the full cascade of the source implementation: bins annual drainage and soil organic matter share, looks up a topsoil denitrification share from meisinger_denitrification, applies subsoil NO3 reduction (subsoil_no3_reduction) and a carbon-to-nitrogen leaching attenuation, then re-derives `denitrification_n_t` as the residual of `n_surplus_t` minus the computed `no3_n_t` (the raw denitrification share is only an intermediate; see Details). `"ipcc_fracleach"` is the IPCC 2019 Tier 1 leached fraction, keyed on `moisture_regime` and `water_regime`, with every factor read from ipcc_2019_soil_n_ef: `FracLEACH-(H)` is 0.24 on wet rows and on dry irrigated rows and 0 on dry rainfed rows (IPCC 2019 Refinement Vol 4 Ch 11 Section 11.2.2.2 p. 11.25: 0.24 "for wet climates or in dry climate regions where irrigation (other than drip irrigation) is used"), and `EF5` is 0.011 (Table 11.3 (Updated) p. 11.26). All irrigation is treated as non-drip, since no layer says which is drip: an upper-bound assumption for dry irrigated land. On wet and dry irrigated rows the rest of the surplus is booked as `denitrification_n_t`, as in the Meisinger cascade. On dry rainfed rows, where FracLEACH is 0, it is booked as `retained_profile_n_t`, nitrogen left in the soil profile, and `denitrification_n_t` is 0: FracLEACH says that surplus does not leach, not that it denitrifies, and no method here estimates its fate. Each row is stamped `"ipcc_fracleach_wet"`, `"ipcc_fracleach_dry_irrigated"` or `"ipcc_fracleach_dry_rainfed_retai`
Eq. 11.10 and Table 11.3 define `FracLEACH-(H)` per kg of N added (`F_SN + F_ON + F_PRP + F_CR + F_SOM`), so it multiplies `n_input_for_n2o_t`, the same input set that prices direct N2O in `build_nitrogen_balance()`, not the surplus. `"fracleach_surplus"` applies the same fractions to `n_surplus_t` instead, the form before whep#1535: a surplus-based deviation from the IPCC definition, kept selectable for comparison, stamped `"fracleach_surplus_wet"`

and so on. Under every method `no3_n_t + denitrification_n_t + retained_profile_n_t` equals `n_surplus_t`.

Usage

```
calculate_n_leaching(
  x,
  drainage_mm = NULL,
  method = c("meisinger_drainage", "ipcc_fracleach", "fracleach_surplus"),
  example = FALSE
)
```

Arguments

`x`	A tibble with `n_surplus_t`. `method = "meisinger_drainage"` requires `fert_type`, `climate`, `irrig_cat`, `land_use`, `cn_input` (may be `NA`), `tillage` (checked only for synthetic rows) and `som_share`; `method = "ipcc_fracleach"` requires `moisture_regime` (`"wet"` or `"dry"`), on dry rows `water_regime` (`"rainfed"` or `"irrigated"`), and `n_input_for_n2o_t`, the N added in IPCC Eq. 11.10; `"fracleach_surplus"` requires the first two.
`drainage_mm`	A numeric vector aligned to `x`'s rows giving annual drainage (mm), or a single string naming a column of `x` to use instead. Kept as a separate argument (rather than a static `x` column) because in the full pipeline it flows in from Module A's gridded water balance.
`method`	`"meisinger_drainage"` (default), `"ipcc_fracleach"` or `"fracleach_surplus"`.
`example`	If `TRUE`, return a small fixture instead of computing from `x`. Defaults to `FALSE`.

Details

For `method = "meisinger_drainage"`, `denitrification_n_t` is computed twice: first as `n_surplus_t * denit_share` (the raw Meisinger share) to derive `no3_n_t`, then overwritten as `n_surplus_t - no3_n_t`, as in the source implementation. The RETURNED `denitrification_n_t` is this second, residual value, not the raw share product; this is a deliberate two-step sequence in the source, not a redundant computation to simplify away. Drainage and soil organic matter bins are matched with the source's strictly-open `s_min < s < s_max` filter: a value exactly on a shared bin edge, or outside the covered range, matches no bin and aborts via the unmatched-row check (the source drops it), rather than being pulled into an adjacent or ceiling bin.

Manure/organic rows (`fert_cat == "Manure"`, i.e. every `fert_type` other than `"Synthetic"`) always join the Meisinger table's `tillage == "Not_specified"` row regardless of `x$tillage`. Synthetic rows join on `x$tillage`, but only `"Tillage"` exists in the source's synthetic block: a synthetic row with `tillage == "No_tillage"` aborts.

For `method = "ipcc_fracleach"`, a missing or invalid `moisture_regime` aborts as in `calculate_soil_n2o()`, and a dry row without a `water_regime` aborts (`whep_missing_water_regime` for an absent column, `whep_water_regime_value` for a missing or unknown value, naming the count): dry irrigated and dry rainfed land leach 0.24 and 0 of their surplus, so a dry row of unknown

regime has no defensible fraction. Wet rows need no `water_regime`. `"fracleach_surplus"` checks the same.

Under `"ipcc_fracleach"` leaching is capped at `n_surplus_t`, so it never exceeds the nitrogen left after crop uptake. IPCC Tier 1 has no such cap; rows where it binds carry the suffix `"_capped_surplus"` on their stamp (e.g. `"ipcc_fracleach_wet_capped_surplus"`). A missing `n_input_for_n2o_t` column, or a missing value in it, aborts (`whep_missing_n_added`, `whep_n_added_na`): read as zero it would leach nothing while the partition still closed.

Value

`x` with `no3_n_t`, `denitrification_n_t`, `retained_profile_n_t` (0 under `"meisinger_drainage"`, which books every non-leached tonne as denitrified), `n2o_indirect_no3_n_t` and `method_leaching` appended.

Source

IPCC (2019) 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol 4, Ch 11, Section 11.2.2.2, p. 11.25 (`FracLEACH-(H)`) and Table 11.3 (Updated), p. 11.26 (`EF5`).

Examples

```
calculate_n_leaching(example = TRUE)
```

---

calculate_n_surplus *Calculate the gridded soil-surface nitrogen surplus.*

Description

Derives the nitrogen surplus from a `build_nitrogen_balance()` output. The default `"harvest_removal"` method is the standard soil-surface nitrogen balance: net inputs (`n_input_std_t`) minus the nitrogen exported in harvest (crop product, used or exported residue and grazed forage), matching the basis of the Schulte-Uebbing et al. (2022) critical nitrogen surplus. Field-burnt residue nitrogen is not subtracted (it volatilises in place, a loss inside the surplus, not an export), nor is recycled residue (returned to the field, internal). The `"full_balance"` method instead returns the balance's post-loss `n_balance_t`, a sensitivity alternative. The surplus may be negative (a nitrogen deficit) and is not clamped here, as clamping is a boundary or leaching concern. The full grid key (`lon`, `lat`, `area_code`, `item_cbs_code`, `year`) and every balance column are preserved.

Usage

```
calculate_n_surplus(
  balance,
  method = c("harvest_removal", "full_balance"),
  example = FALSE
)
```

Arguments

balance	A build_nitrogen_balance() output tibble. For "harvest_removal" it must carry n_input_std_t, prod_n_t, used_residue_n_t and grazed_weeds_n_t; for "full_balance" it must carry n_balance_t. When an area_ha column (each crop's harvested hectares in the cell) is present, the per-hectare surplus surplus_kgn_ha is also emitted.
method	Surplus definition: "harvest_removal" (default, n_input_std_t - (prod_n_t + used_residue_n_t + grazed_weeds_n_t)) or "full_balance" (the balance's n_balance_t, a sensitivity).
example	If TRUE, return a small fixture instead of computing from balance. Defaults to FALSE.

Value

The balance tibble with surplus_n_t (tonnes N, may be negative), method_surplus and, when area_ha is present, surplus_kgn_ha (kg N per hectare). An area-keyed balance also gains the polity columns below; one without an area_code is returned without them.

Polity columns

Every area-keyed output carries the polity its area_code resolves to in that row's year:

- polity_area_code: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use reporting_polity_code to say which territory a row belongs to.
- reporting_polity_code: The polity itself, e.g. ESP-1846-1914. It is year-aware, so the same area_code resolves to different polities in different years, which is the point of the crosswalk.
- reporting_polity_name: Its name. It can differ from the area's own name where the area folds into an aggregate.
- reporting_polity_has_geometry: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. FALSE is a documented gap upstream, not an error.

Rows whose area_code resolves to no polity keep the columns with NA rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See add_polity_code() for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 (area, year) cells – add_polity_code() says so as mapping_status == "backcast_anchor", and polity_coverage_gaps() reports it as gap_kind == "backcast_anchor". These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so reporting_polity_code can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after SUD-1956-2011 ends, and its post-2011 rows carry that code. These columns do not say so: add_polity_code() reports such a row as mapping_status == "out_of_span", and that column is dropped

here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
calculate_n_surplus(example = TRUE)
```

---

calculate_nh3 *Estimate ammonia-N volatilisation from applied nitrogen.*

Description

Three independent methods for the fraction of applied nitrogen volatilised as ammonia. `"ipcc"` (IPCC 2019 Tier 1) needs only `fert_type` and applies a single global fraction. `"manner"` (the default) dispatches each row through the process-based `calculate_manner_nh3()` MANNER model (Task C4), which requires far more driver detail (see Details); this asymmetry in input requirements is intentional, not an oversight. `"manner_default"` dispatches each row through `calculate_manner_nh3_default()` instead, the same process-based organic-manure model but with `technique`/`incorporation_delay_h` filled in from a documented gross-assumption blend rather than required as driver columns.

Usage

```
calculate_nh3(x, method = "manner", example = FALSE)
```

Arguments

`x`	A tibble with `n_input_t` (numeric, tonnes N) and `fert_type`. `method = "manner"` additionally requires `manner_fertiliser` and the driver columns listed in Details. `method = "manner_default"` additionally requires `manner_fertiliser` and the driver columns listed in Details, but NOT `technique`/`incorporation_delay_h`.
`method`	`"manner"` (default, process-based, per-row), `"ipcc"` (Tier 1, global fraction) or `"manner_default"` (process-based organic path with a gross-assumption technique/incorporation-delay blend, no `technique`/`incorporation_delay_h` columns required).
`example`	If `TRUE`, return a small fixture instead of computing from `x`. Defaults to `FALSE`.

Details

`method = "manner"` requires `x` to already carry a `manner_fertiliser` column holding the exact `calculate_manner_nh3()` `fertiliser` key (`"Urea"`, `"AN"`, `"CAN"`, `"AS"`, `"cattle_slurry"`, `"pig_slurry"`, `"FYM"`, `"poultry_manure"` or `"urban"`) plus every driver column that key's path needs: `soil_ph`, `rate_kg_ha`, `rainfall_mm`, `irrigated`, `temp_c`, `temp_c_annual_mean` for the synthetic path; `rainfall_mm`, `irrigated`, `windspeed_ms`, `technique`, `system`,

`temp_c`, `incorporation_delay_h`, `species` (unless `manner_fertiliser == "urban"`) for the organic path. This function does not infer `manner_fertiliser` from `fert_type` (e.g. which synthetic sub-type `"Synthetic"` maps to is not determined by `fert_type` alone) and does not silently fall back to `"ipcc"` or invent driver values: a missing required column aborts naming exactly which column is absent. Each row is dispatched to `calculate_manner_nh3()` individually (MANNER's dispatch is inherently per-row categorical, not vectorizable across the coefficient joins); this row-iteration is isolated to a small private helper.

`method = "manner_default"` requires the same `manner_fertiliser` column (restricted to the organic-manure keys, since the gross default only covers that path) plus `rainfall_mm`, `irrigated`, `windspeed_ms`, `system`, `temp_c` and `species` (unless `manner_fertiliser == "urban"`). It does NOT require `technique` or `incorporation_delay_h`: those are filled in from manner_default_technique_mix (see `calculate_manner_nh3_default()`'s Details for the gross-assumption reasoning), never invented per-row.

Where the MANNER drivers come from: nothing in the package produces them on any build path, so a caller supplies them. The one exception with a reader is `windspeed_ms`, which `read_lpjml_wind()` returns as monthly 0.5-degree GSWP3-W5E5 wind speed (m/s, the forcing LPJmL is driven with, from the `lpjml-wind-isimip-1901-2019` input via `WHEP_WIND_DIR`). It is not joined onto the N-loss rows by any builder: the grain of that join, the averaging window (growing season, application month or year) and the rule for years outside the wind record are open modelling choices (whep#1078). `build_nitrogen_balance()` therefore refuses a MANNER `nh3` method before assembling anything unless its `data$n_balance_drivers` carries the columns listed above. `"ipcc"` needs none of them.

Value

`x` with `nh3_n_t` and `method_nh3` appended.

Examples

```
calculate_nh3(example = TRUE)
```

---

calculate_nonsymbiotic_bnf

Estimate non-symbiotic biological nitrogen fixation.

Description

Estimates free-living and associative BNF in agricultural soils from a base rate (crop-specific from `bnf`, or a default) scaled by environmental modifiers for nitrogen, temperature, water, soil organic matter, pH and clay. Each modifier activates only when its driver column is present.

Usage

```
calculate_nonsymbiotic_bnf(
  x,
  nonsymbiotic_params = list(),
  soil_params = list()
)
```

Arguments

`x`	A tibble with `area_ha`. Optional `nonsymbiotic_base_kg_ha` (or `item_prod_code` to join the crop-specific base rate), plus the nitrogen, temperature, water and soil (`som_pct`, `soil_ph`, `clay_pct`) drivers.
`nonsymbiotic_params`	Named list overriding `nsbnf_default_kg_ha`, `k_n_synth`, `k_n_org`, `t_opt`, `t_sigma`, `ai_threshold`.
`soil_params`	Named list overriding `k_som`, `som_ref`, `ph_opt`, `ph_sigma`, `k_clay`, `clay_ref`.

Value

The input tibble with `nonsymbiotic_base_kg_ha`, the six `f_*_nonsymbiotic` modifiers, `f_env_nonsymbiotic` and `nonsymbiotic_bnf_t`.

Examples

```
calculate_nonsymbiotic_bnf(tibble::tibble(area_ha = 40))
```

`calculate_npp_carbon_nitrogen` *Partition crop and weed NPP into dry matter, carbon and nitrogen.*

Description

Converts crop NPP components (product, residue, root) and weed biomass to nitrogen and carbon using the `bio_coefs` per-component coefficients and the `weed_coefs` scalars. Root and weed below-ground nitrogen include rhizodeposits. Root carbon uses `root_c_kgdm`, which includes root biomass carbon plus rhizodeposit carbon per tonne of root dry matter; use `root_mass_c_kgdm` in `bio_coefs` for root tissue carbon alone. The residue-to-soil split is computed only when a `residue_soil_dm_t` column (from `calculate_residue_destinies()`) is present.

Usage

```
calculate_npp_carbon_nitrogen(x)
```

Arguments

`x`	A tibble with `item_prod_code`, `product_dm_t`, `residue_dm_t` and `root_dm_t`. Optional `weed_ag_dm_t` (above-ground weed dry matter; treated as 0 when absent), `crop_npp_dm_t` (kept when present) and `residue_soil_dm_t` (enables the soil-residue nitrogen and carbon split).

Value

The input tibble with weed dry matter, and nitrogen (`*_n_t`) and carbon (`*_c_t`) for product, residue, root, weeds, crop NPP and total NPP, plus `method_weed_npp`: `"supplied"` where `weed_ag_dm_t` arrived and `"absent_zero"` where it did not and the weed terms are a fill. Filter on it before reading `weed_npp_c_t` or `weed_npp_n_t` as a measurement – an absent weed stream and a weed-free field are the same zero otherwise (whep#1034).

Examples

```
tibble::tibble(item_prod_code = "15", production_t = 100, area_ha = 40) |>
  calculate_crop_npp() |>
  calculate_npp_carbon_nitrogen()
```

`calculate_nue_crops` *N soil inputs and Nitrogen Use Efficiency (NUE) for crop*

Description

N inputs (deposition, fixation, synthetic fertilizers, urban sources, manure) and N production in Spain from 1860 to the present for the GRAFS model at the provincial level. The crop NUE is defined as the percentage of produced nitrogen relative to the total nitrogen inputs to the soil. Total soil inputs are calculated as: inputs = deposition + fixation + synthetic + manure + urban

Usage

```
calculate_nue_crops(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing nitrogen input, production, and NUE data. It includes the following columns:

- `year`: Year.
- `province_name`: The Spanish province.
- `item`: The item which was produced, defined in `names_biomass_cb`.
- `box`: One of the two systems of the GRAFS model: cropland or semi-natural agroecosystems.
- `deposition`: Atmospheric nitrogen deposition in megagrams (Mg).
- `fixation`: Nitrogen fixation in megagrams (Mg).
- `synthetic`: Synthetic nitrogen fertilizer applied to the land in megagrams (Mg).

- `manure`: Nitrogen in manure applied to the land in megagrams (Mg).
- `urban`: Nitrogen in wastewater from human sources in megagrams (Mg).
- `prod`: Produced nitrogen in megagrams (Mg).
- `inputs`: Total nitrogen inputs in megagrams (Mg).

Examples

```
calculate_nue_crops(example = TRUE)
```

---

calculate_nue_livestock

NUE for Livestock

Description

Calculates Nitrogen Use Efficiency (NUE) for livestock categories (excluding pets).

The livestock NUE is defined as the percentage of nitrogen in livestock products relative to the nitrogen in feed intake: nue = prod_n / feed_n * 100

Additionally, a mass balance is calculated to check the recovery of N in products and excretion relative to feed intake: mass_balance = (prod_n + excretion_n) / feed_n

Usage

```
calculate_nue_livestock(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing:

- `year`: Year
- `province_name`: Spanish province
- `livestock_cat`: Livestock category
- `item`: Produced item
- `prod_n`: Nitrogen in livestock products (Mg)
- `feed_n`: Nitrogen in feed intake (Mg)
- `excretion_n`: Nitrogen excreted (Mg)
- `nue`: Nitrogen Use Efficiency (%)
- `mass_balance`: Mass balance ratio (%)

Examples

```
calculate_nue_livestock(example = TRUE)
```

`calculate_potential_npp` *Estimate potential net primary production.*

Description

Estimates potential net primary production per unit area, used downstream to scale weed biomass on cropland. Several methods are available; the default `lpjml` reads gross managed-grassland NPP from a finished LPJmL run, while the climate methods compute potential NPP from temperature, water input and actual evapotranspiration.

Usage

```
calculate_potential_npp(
  x,
  method = c("lpjml", "miami", "nceas", "rosenzweig"),
  lpjml = list()
)
```

Arguments

`x`	A tibble. The climate methods require `temp_c` (mean annual temperature, degrees C), `water_input_mm` (precipitation plus irrigation) and `aet_mm` (actual evapotranspiration). The `lpjml` method instead needs a spatial key to join the gridded grass NPP (see `lpjml`).
`method`	Potential-NPP method, one of `"lpjml"` (default), `"miami"`, `"nceas"` or `"rosenzweig"`.
`lpjml`	A named list of options for the `lpjml` method (passed to the LPJmL grass reader). Ignored by the climate methods.

Value

The input tibble with `npp_potential_dm_t_ha` (potential NPP, t DM per hectare) and `method_npp_potential` (the method used).

Examples

```
calculate_potential_npp(
  tibble::tibble(temp_c = 15, water_input_mm = 800, aet_mm = 700),
  method = "miami"
)
```

`calculate_residue_destinies` *Estimate the destinies of crop residues.*

Description

Splits crop residue dry matter into four destinies that sum to the total residue: fed to livestock, used as livestock bedding, burned / removed for fuel, and left on the field for soil incorporation.

Usage

```
calculate_residue_destinies(
  x,
  method = c("recovery_regional", "shares"),
  bedding_fraction = 0,
  unmatched_recovery = c("report", "abort"),
  recovery = c("wirsenius", "legacy"),
  feed_share = c("wirsenius", "legacy"),
  bedding = c("wirsenius", "fraction")
)
```

Arguments

`x`	A tibble with `item_prod_code` and `residue_dm_t`. The `recovery_regional` method also needs `region_krausmann` (for the recovery rate) and `region_un_sub` (for the feed-use fraction, the UN M49 sub-regions of `regions_full$region_UN_sub`). `region_krausmann` can use the recovery-table labels or the matching `regions_full` labels. The `shares` method needs `year`.
`method`	Destiny method: `"recovery_regional"` (default, the regional recovery rate times the UN-sub-regional feed-use fraction) or `"shares"` (the Spain-specific per-crop-year use/burn shares, flagged `to_be_revised`).
`bedding_fraction`	Fraction of the recovered **non-feed** residue used as livestock bedding under `bedding = "fraction"`, one number in `[0, 1]`. Default `0`. Must stay `0` under `bedding = "wirsenius"`, which sets its own share.
`unmatched_recovery`	What the `recovery_regional` method does with a row that reaches no recovery rate at all, because its crop carries no Krausmann category or its region label reaches no recovery region: `"report"` (default) keeps the historical all-to-soil treatment and warns with the row count and tonnage, `"abort"` refuses to continue. Ignored by the `"shares"` method.
`recovery`	Which recovery-rate variant the `recovery_regional` method reads: `"wirsenius"` (default since whep#1330, every rate Wirsenius 2000 states, at the value it states) or `"legacy"` (the table as shipped before whep#1163). See the Two recovery variants section. Ignored by the `"shares"` method.

`feed_share` Which feed-use fraction the `recovery_regional` method reads: `"wirsenius"` (default since whep#1398, Wirsenius 2000 Table 3.20, per crop by-product and Wirsenius region, the feed half of the recovery rates) or `"legacy"` (`residue_feed_fraction.csv`, one fraction per UN M49 sub-region for every crop). See the Two feed-share variants section. Ignored by the `"shares"` method.

`bedding` Bedding rule: `"wirsenius"` (default, 12.4% of the recovered cereal straw, Wirsenius 2000 Table 3.21) or `"fraction"` (`bedding_fraction` of the recovered non-feed residue of every crop). See the Where the bedding share comes from section.

Value

The input tibble with `residue_feed_dm_t`, `residue_bedding_dm_t`, `residue_burn_dm_t`, `residue_soil_dm_t`, `residue_bedding_fraction` (the share applied: of the recovered residue under `"wirsenius"`, zero for a non-cereal; of the recovered non-feed residue under `"fraction"`), `method_residue_bedding`, `method_residue_destiny`, and `method_residue_recovery` (the `recovery` variant, `NA` for the `"shares"` method) and `method_residue_feed` (per row, the table that priced its feed share: `"wirsenius"` or `"legacy"`, `NA` for the `"shares"` method). The `"recovery_regional"` method also returns `residue_recovery_matched`, `FALSE` where no recovery rate was found, which is what separates a rate the table gives as zero from a zero standing in for a failed lookup.

Bedding

Bedding straw leaves the field with the rest of the recovered residue and comes back to the soil later, through the yard, as part of the managed manure. It is therefore carved out of the recovered **non-feed** residue – the mass the commodity balance books as `other_uses` – and **never** out of `residue_soil_dm_t`, which is the residue that stays on the field. That is the split IPCC 2019 Refinement Vol. 4 Ch. 10 p. 10.95 asks for when it tells inventory compilers to cross-check bedding nitrogen "relative to the amount of agricultural residues that is removed for other purposes (i.e. bedding) other than the amount of agricultural residues returned to soils or burnt", so as "to eliminate the possibility of double counting".

Where the bedding share comes from

`bedding = "wirsenius"` (default) reads the one bedding figure that comes from the same accounting as the recovery rates: **Wirsenius (2000)**, *Human Use of Land and Organic Materials*, PhD thesis, Chalmers University of Technology, Table 3.21 (p. 126). Of 2700 Tg DM/yr of cereals straw & stover generated, 530 Tg is "not recovered", so 2170 Tg is recovered, and 270 Tg is "used as litter for animal bedding" – 14% of the 1940 Tg *distributed*. This function's denominator is the *recovered* residue, so the share is taken against the table's own recovered total, `270 / 2170 = 0.124`. That is the same distributed -> recovered conversion whep#1431 makes for the Table 3.20 feed shares; here the world table carries both totals, so no loss factor has to be assumed (the 10% loss route, `0.14 * 0.90`, gives 0.126).

Wirsenius models cereals straw and stover as the *only* bedding material (p. 85, after Section 2.3.1), so the share applies to the five cereal categories (`Wheat, other cereals`, `Rice, Paddy`, `Maize`, `Sorghum`, `Millet`) and every other crop beds nothing. It is one **world**

share applied in every region: the thesis prints no regional litter total, and its regional litter-use rates (Table 3.13, p. 86, kg straw per kg housed manure) are "very rough figures" (p. 85), with South & Central Asia cattle set to zero because "no reasonable values on litter use could be identified" – not an estimate to inherit. The bedding is carved out of the recovered non-feed residue and capped at it; where the cap binds a warning says how much.

`bedding = "fraction"` keeps the caller-set rule: `bedding_fraction` of the recovered non-feed residue of every crop, default `0` (no bedding). Two other anchors are on that denominator family, neither of them global:

- Statistics Denmark HALM/HALM1/HALM2 – the only official statistic with a bedding-only column: 16-21% of straw *production*, about 30% of *removed* straw.
- Bentsen, Felby & Thorsen (2014), Prog. Energy Combust. Sci. 40:59-73, Table 5 – Denmark, barley 16% and wheat 11% of *production*.

FAO GLEAM's `FracRemove` and IPCC 2019 Eq. 11.6's `FracRemove` both merge bedding with feed and construction into one term, so neither is usable. The rule used is recorded in `method_residue_bedding`.

Where the recovery rates come from

The `recovery_regional` recovery rates live in `inst/extdata/coefs/residue_recovery.csv`. Its numeric columns are sourced separately and carry a provenance column each (`source_ratio`, `source_recovery` and `source_recovery_wirsenius`), because they agree with the source to different degrees. All are **Wirsenius (2000)**, *Human Use of Land and Organic Materials*, PhD thesis, Chalmers University of Technology – Table 3.17 (recovery rates, p. 94) and Table 3.16 (harvest index, p. 92, from which `residue_dm_product_dm` is the residue:product ratio rounded to one decimal). The file was named after Krausmann and its two key columns still are, but no coefficient in it is Krausmann's. Only the crop category (`items_prod_full$Cat_Krausmann`) really is his: `region_krausmann` here holds `regions_full$region_HANPP` labels, whose eight values are Wirsenius's eight region names (whep#1132), though not his country membership (see the Two feed-share variants section).

Table 3.17 states its rates per crop **category**, not per crop: one "Cereals straw & stover" row governs every cereal, and one "Sugar crops tops & leaves" row governs both crops Table 3.16 files under sugar crops, cane and beet. Read that way it governs twelve of the twenty categories here, and `source_recovery` splits them (whep#1150):

- nine match it cell for cell;
- three sit **below** it – `Groundnuts in Shell`, `Sugar Beets` and `Sugar Crops nes`, each against 0.90 in every region;
- three – roots and tubers, cassava and oil palm – are residues the thesis does model (Table 2.6, pp. 48-49: cassava leaves and tops, potato tops, oil palm leaves and trunks) but Table 3.17 does not list, and for those Wirsenius states that recovery rates "were assumed to be close to 100 percent" (p. 94);
- five – dry beans, pulses, castor beans, permanent crops and fodder crops – have **no residue flow in the thesis at all**: Table 2.6 gives pulses, fruits, tree nuts, vegetables and stimulants "no representation of by-products" (p. 47), models forage crops whole, and has no castor. The p. 94 default does not reach them, so the source is silent on them (whep#1163).

Two recovery variants

`recovery =` selects the rate column, and `method_residue_recovery` records which one was used:

- `"wirsenius"` (default) reads `recovery_rates_wirsenius`: every rate the thesis states, at the value it states. The three below-source categories take 0.90, the three p. 94 categories take 1.00 – "close to 100 percent" read as 1.00, which is a reading of the text and not a number it prints – and the five categories the source is silent on keep the legacy assumed rate. `source_recovery_wirsenius` labels each.
- `"legacy"` reads `recovery_rates`, the table as shipped before whep#1163, whose departures from the source are all downward.

`"wirsenius"` is the only variant in which every rate is traceable to the cited source, and it is the default since whep#1330. Its switch waited on the gross residue base these rates multiply, once thought to be about 36% too high (step 2 of whep#1132). whep#1330 found that excess gone in dry matter, but it measured the `crop_residues` pin as it was then read, and that pin holds residue already multiplied by the `"legacy"` rate (whep#1195). Read as the residue produced, world cereal residue in dry matter was 16.1% above the three-method mean of Smerald, Rahimi & Scheer (2023), doi:10.1038/s4159702302587-0, over 1997–2021. Since whep#1448 `get_primary_residues()` estimates cereal residue from yield, and the 1997–2021 mean is 0.8% above theirs, inside their band in every year (`validation/residue_base_dm.R`). The rates the switch moves are all non-cereal, and no published global total for the non-cereal residue base was found to check it against.

Both rate columns are keyed on Wirsenius's eight regions, so a row takes his own region membership (Table 3.1, p. 58), not the HANPP label's: from its HANPP region and UN M49 sub-region, through `residue_feed_regions.csv`. The HANPP label files Southeast Asia, Russia, Belarus and the Caucasus under South & Central Asia, whose rates are the highest (whep#1430). On the commodity balance that lowers residue production by 2.4% in 2010 and 2.7% in 2020.

Measured on `get_primary_residues()`'s fresh `value` (the commodity balance's basis), `"wirsenius"` against `"legacy"` in 2010 raises recovered residue from 6939 to 7148 Mt fresh matter (+3.0%), the feed destiny from 1964 to 2025 Mt (+3.1%) and the burned/other-use destiny from 4976 to 5123 Mt (+3.0%), and lowers the soil destiny from 1789 to 1580 Mt (-11.7%); over 1961–1965 the same moves are +6.3%, +5.4%, +6.6% and -20.4%. Roots and tubers, cassava, sugar beet and groundnut are the only categories that move.

Three categories move no mass at all today, which is why the fodder rate of 0 is not the live problem it looks like: `Sugar Crops nes` and `Oil Palm Fruit` are in the table and in no production item, and no `Fodder crops` item reaches the residue pin.

Two feed-share variants

`feed_share =` selects the fraction of the **recovered** residue booked as feed (the rest is the burned / other-use destiny), and `method_residue_feed` records, per row, which table priced it:

- `"wirsenius"` (default) reads `residue_feed_assignment.csv`: Wirsenius (2000) Table 3.20 (p. 102), the "Assignm." rows for crop by-products, which the thesis coordinates directly with the Table 3.17 recovery rates – "assumptions on recovery rates were directly coordinated with those on assignment for use as feed" (p. 95). One row

covers rice straw, one the other cereals (read for millet too, as Table 3.17 is), and one each cassava leaves, potato tops (read for all roots and tubers), sugar cane tops, sugar beet tops, groundnut stalks and the other oil crops. The seven categories it has no row for keep the legacy fraction, and their rows say `"legacy"`.

- `"legacy"` reads `residue_feed_fraction.csv` (Smil 1999, Lal 2005, Krausmann 2008, Erenstein 2014, McIntire 1992): one fraction per UN M49 sub-region, 0.05 to 0.45 around a 0.20 global default, the same for every crop. It is not paired with the recovery rates, and was the only feed share before whep#1398.

Table 3.20 is a share of the amount **distributed**, which is not the amount recovered: Wirsenius sets the distribution and storage losses of all crop by-products to 10% (p. 98), and in North America & Oceania leaves 5% of the recovered cereal and soybean straw in the field (p. 95, note 117). So the share of recovered residue fed is the printed value times 0.90 (0.855 for those North American straws): wheat straw in South & Central Asia is 0.80 of the distributed straw and 0.72 of the recovered. Read without the conversion it would book more feed than Wirsenius does.

Like the recovery rates, the table is keyed on Wirsenius's own region membership (Table 3.1, p. 58), through `residue_feed_regions.csv`. It matters more here: the HANPP label files Southeast Asia, Russia and Belarus, and the Caucasus under South and Central Asia, whose cereal feed share is 0.80, where Wirsenius has them at 0.30, 0.10 and 0.60.

Measured on `get_primary_residues()` in dry matter, the cereals' feed share of their generated residue is 0.224 (2010) and 0.225 (2020) under `"legacy"`, 0.305 and 0.309 under `"wirsenius"`. Wirsenius reports "some 33 percent of the amount generated" for the early 1990s (p. 177), and Smerald, Rahimi & Scheer (2023), *Scientific Data* **10**:685, doi:10.1038/s41597023025870, a share of 0.331 of the residue produced going to livestock. That figure is read from the authors' Zenodo notebook (whep#1132), not printed in the article, and their livestock category is "animal feed or bedding", so it is an upper bound for feed alone. The residue the commodity balance books as feed (Straw and Other crop residues) rises from 1089 to 1550 Tg of dry matter in 2010 (+42%) and from 1293 to 1856 Tg in 2020 (+44%); recovered and soil-returned residue do not move. In the 2010 national feed allocation (`get_feed_intake()`), residue intake rises from 1578 to 1985 Tg of dry matter and grass intake falls from 2703 to 2367 Tg (-12.4%), the residue displacing grazing. These are measured on the yield-based cereal residue base of whep#1448.

The default is `"wirsenius"` because it is the half of a matched pair whose other half, the recovery rates, is already the default, every value it reads is traceable to the source, and it lands within 8% of both published anchors above, where the legacy table falls about a third short of them.

Examples

```
calculate_residue_destinies(
  tibble::tibble(
    item_prod_code = "15", residue_dm_t = 100,
    region_krausmann = "Western Europe", region_un_sub = "Western Europe"
  )
)
```

`calculate_soc_amg` *Simulate soil organic carbon with the AMG model.*

Description

Analytical AMG trajectory: an active decomposing pool and an inert stable pool. The humification coefficient is selected from the carbon input type via `whep::amg_h_by_input_type` (default 0.15). The active pool relaxes to its steady state `h * input / k`; the stable pool is constant.

Usage

```
calculate_soc_amg(
  initial_soc_mgc_ha,
  c_input_mgc_ha_yr,
  years,
  clay_pct = NA,
  climate_modifier = 1,
  c_input_type = NA,
  init_mode = c("fixed_iom", "steady_state")
)
```

Arguments

`initial_soc_mgc_ha`	Initial soil organic carbon stock (Mg C per ha).
`c_input_mgc_ha_yr`	Annual carbon input (Mg C per ha per year).
`years`	Number of years to simulate.
`clay_pct`	Soil clay content (percent); unused, kept for contract.
`climate_modifier`	Annual climate rate modifier (dimensionless), scaling the active-pool decomposition rate.
`c_input_type`	Carbon input type label used to look up the humification coefficient.
`init_mode`	Initial pool split. `"fixed_iom"` splits the supplied initial stock by the published stable fraction. `"steady_state"` ignores the supplied stock and starts from the analytical equilibrium `ca_ss / (1 - f_iom)` (active pool at its steady state `ca_ss = h * input / k`, stable pool the matching inert share).

Value

A tibble with one row per year: `year`, `ca`, `cs` and `soc_total`.

Source

Saffih-Hdadi, K. & Mary, B. (2008). doi:10.1016/j.soilbio.2007.08.022.

Examples

```
calculate_soc_amg(
  initial_soc_mgc_ha = 50,
  c_input_mgc_ha_yr = 2,
  years = 5
)
```

---

calculate_soc_century

Simulate soil organic carbon with the Century five-pool model.

Description

Annual Century trajectory integrated with `deSolve::lsoda`. Carbon flows through structural and metabolic litter and active, slow and passive soil organic matter pools, with texture-dependent rates and inter-pool transfer fractions.

Usage

```
calculate_soc_century(
  initial_soc_mgc_ha,
  c_input_mgc_ha_yr,
  years,
  clay_pct = NA,
  climate_modifier = 1,
  silt_pct = NA
)
```

Arguments

`initial_soc_mgc_ha`	Initial soil organic carbon stock (Mg C per ha).
`c_input_mgc_ha_yr`	Annual carbon input (Mg C per ha per year).
`years`	Number of years to simulate.
`clay_pct`	Soil clay content (percent).
`climate_modifier`	Annual climate rate modifier (dimensionless), scaling every pool decomposition rate.
`silt_pct`	Soil silt content (percent). `NA`, the default, falls back to soc_turnover_params' `century,defaults,silt_pct`, which is SoilR's function-signature default of 45 rather than a measurement. Both of Century's texture terms are functions of clay PLUS silt – the active-pool multiplier `fTxtr = 1 - 0.75 * (clay + silt)` and its respired fraction `Es = 0.85 - 0.68 * (clay + silt)` – so silt is as load-bearing as clay. HWSD's share-weighted global mean is 28.2%, not 45%.

Value

A tibble with one row per year: `year`, `str`, `met`, `act`, `slw`, `pas` and `soc_total`.

Source

Parton, W. J. et al. (1987). doi:10.2136/sssaj1987.03615995005100050015x; SoilR implementation: Sierra, C. A. et al. (2012). doi:10.5194/gmd510452012.

Examples

```
calculate_soc_century(
  initial_soc_mgc_ha = 50,
  c_input_mgc_ha_yr = 2,
  years = 5,
  clay_pct = 20
)
```

---

calculate_soc_dynamics

Simulate soil organic carbon dynamics with a selectable model.

Description

Run one of the five soil-organic-carbon turnover models (HSOC, RothC, ICBM, AMG or Century) through a single interface. When climate drivers are present in `data` (for example `temp_c`), the selected model's native annual climate rate modifier is computed from the matching climate function and passed in; otherwise the model runs with a neutral modifier of 1. The chosen model is stamped into a `method_soc` column on the output.

Usage

```
calculate_soc_dynamics(
  model = c("hsoc", "rothc", "icbm", "amg", "century", "lpjml"),
  data = list(),
  example = FALSE
)
```

Arguments

`model`	SOC model to run: one of `"hsoc"`, `"rothc"`, `"icbm"`, `"amg"`, `"century"` or `"lpjml"`. Defaults to the most detailed pool structure available, `"hsoc"`.
`data`	Named list of model arguments. Always carries `initial_soc_mgc_ha`, `c_input_mgc_ha_yr` and `years`; may also carry `clay_pct` and model-specific arguments (for example `c_input_type` and `init_mode` for AMG). Climate drivers, when present, are read from the same list to build the climate modifier (see `soc_rate_modifier_rothc` and siblings for the per-model driver names). If the drivers are absent but `data$climate_modifier` already holds a value, that value is used instead of recomputing it; only when neither is available does the model run with a neutral modifier of 1.

`example` If `TRUE`, return a small hardcoded example tibble instead of running the model.

Value

A tibble in the same long schema for every model, one row per year and pool: `year`, `pool` (the running model's native pool name), `stock_mgc_ha` (that pool's stock), `soc_total` (the year's total over all pools, repeated on each pool row) and `method_soc` (the model that ran). Long rather than wide because the five models have different pool sets, so only a long shape can carry the pool detail under identical column names; total-only callers read `dplyr::distinct(out, year, soc_total)` without branching on the model. The per-model functions such as `calculate_soc_hsoc` still return their native wide trajectory.

Source

Model bodies and climate functions as cited in `calculate_soc_hsoc` and `soc_rate_modifier_rothc`.

Examples

```
calculate_soc_dynamics(
  model = "icbm",
  data = list(initial_soc_mgc_ha = 50, c_input_mgc_ha_yr = 2, years = 5)
)
```

---

calculate_soc_hsoc *Simulate soil organic carbon with the HSOC two-pool model.*

Description

Annual HSOC trajectory (as in an earlier regional historical reconstruction): a fresh and a humus decomposing pool plus an inert organic matter pool. The inert pool is the Falloon (1998) function of initial carbon and, as in that paper, is a component of the measured stock rather than an addition to it, so the two decomposing pools open on the remainder `initial_soc_mgc_ha - iom`, split between them in the proportion of their steady states `input_pool / k_pool`. Each year a pool stock loses first-order decomposition and gains its carbon input, so the trajectory relaxes from the supplied stock toward that steady state. Land-use-change carbon transfer is deferred to a later phase (single land use here).

Usage

```
calculate_soc_hsoc(
  initial_soc_mgc_ha,
  c_input_mgc_ha_yr,
  years,
  clay_pct = NA,
  climate_modifier = 1,
  humification_fraction = 0.3
)
```

Arguments

`initial_soc_mgc_ha`
: Initial soil organic carbon stock (Mg C per ha). The trajectory starts here: year 0 of the returned tibble reports this stock, as it does for the four sibling models.

`c_input_mgc_ha_yr`
: Annual carbon input (Mg C per ha per year).

`years`
: Number of years to simulate.

`clay_pct`
: Soil clay content (percent). Scales `humification_fraction` by the Aguilera et al. (2018) Eq. 5-6 texture modifier, which runs 0.72 at 5% clay to 1.13 at 60% and is 1 at RothC's Rothamsted reference of 23.4%: coarse soils stabilise less of the same carbon input. `NA`, the default, means no texture information was supplied and applies no texture adjustment. The gridded balance always supplies clay, so this function and `build_carbon_balance(model = "hsoc")` now return the same stock for the same inputs.

`climate_modifier`
: Annual climate rate modifier (dimensionless).

`humification_fraction`
: Fraction of carbon input humified into the humus pool (the remainder feeds the fresh pool).

Value

A tibble with one row per year: `year`, `fresh`, `humus`, `iom` and `soc_total`.

Source

Coleman, K. & Jenkinson, D. S. (1996). doi:10.1007/9783642610943_17; inert organic matter: Falloon, P. et al. (1998). doi:10.1016/S00380717(97)002563.

Examples

```
calculate_soc_hsoc(
  initial_soc_mgc_ha = 50,
  c_input_mgc_ha_yr = 2,
  years = 5
)
```

`calculate_soc_icbm` *Simulate soil organic carbon with the ICBM two-pool model.*

Description

Analytical ICBM trajectory: a young and an old pool with a closed-form solution evaluated at integer years. A degenerate branch handles the case where the two effective rate constants coincide.

Usage

```
calculate_soc_icbm(
  initial_soc_mgc_ha,
  c_input_mgc_ha_yr,
  years,
  clay_pct = NA,
  climate_modifier = 1
)
```

Arguments

`initial_soc_mgc_ha`	Initial soil organic carbon stock (Mg C per ha).
`c_input_mgc_ha_yr`	Annual carbon input (Mg C per ha per year).
`years`	Number of years to simulate.
`clay_pct`	Soil clay content (percent); unused, kept for contract.
`climate_modifier`	Annual climate rate modifier (dimensionless), scaling both pool decomposition rates.

Value

A tibble with one row per year: `year`, `y`, `o` and `soc_total`.

Source

Andren, O. & Katterer, T. (1997). doi:10.1890/10510761(1997)007[1226:ITICBM]2.0.CO;2.

Examples

```
calculate_soc_icbm(
  initial_soc_mgc_ha = 50,
  c_input_mgc_ha_yr = 2,
  years = 5
)
```

`calculate_soc_lpjml` *Simulate soil organic carbon with the LPJmL two-pool soil model.*

Description

Annual trajectory of LPJmL's soil carbon submodel: a fast and a slow mineral-soil pool, each losing first-order decomposition and gaining a share of the carbon that survives litter respiration. Of the litter carbon that DECOMPOSES, `atmosphere_fraction` is respired straight to the atmosphere and the remainder is split `fast_fraction` to the fast pool and the rest to the slow pool. Neither soil pool transfers to the other and there is no inert pool, so soil respiration leaves the system entirely.

Usage

```
calculate_soc_lpjml(
  initial_soc_mgc_ha,
  c_input_mgc_ha_yr,
  years,
  clay_pct = NA,
  climate_modifier = 1
)
```

Arguments

`initial_soc_mgc_ha`	Initial soil organic carbon stock (Mg C per ha), split between the two pools by their steady-state proportions.
`c_input_mgc_ha_yr`	Annual carbon input to the litter layer (Mg C per ha per year), before the atmospheric respiration share.
`years`	Number of years to simulate.
`clay_pct`	Soil clay content (percent); unused, kept for contract.
`climate_modifier`	Annual decomposition response (dimensionless), scaling both pool rates. See `soc_rate_modifier_lpjml`.

Value

A tibble with one row per year: `year`, `fast`, `slow` and `soc_total`.

What the atmospheric fraction applies to

In LPJmL the fraction multiplies the flux LEAVING the litter pool by decomposition, not the litterfall entering it. The two are equal only when the litter pool is at steady state, and litterfall is strictly the larger wherever litter burns or methanogenises. WHEP has no litter pool, so this function applies the fraction to the carbon input it is given and is therefore a steady-state reading of that chain. Over a transient the approximation is good for leaf and fine-root litter, which transit in one to three years, and poor for woody litter, whose 10-26 year transit is comparable to the 22.25-year soil-bound residence itself.

Earlier wording here said the fraction applied to carbon "entering the litter layer", which contradicted `soc_turnover_params`'s own description of it ("Share of decomposed litter respired straight to the atmosphere"). The parameter table was right.

This applies LPJmL's kinetics to the carbon input WHEP supplies, exactly as `calculate_soc_rothc` applies RothC's. It is **not** a reproduction of LPJmL's own soil carbon stock, which is reported over 0-300 cm and distributed down the profile; the stock returned here is over the same layer as WHEP's other models. The litter pool is excluded, as it is from LPJmL's own `soilc` output.

That vertical distribution is not a rooting-depth function, as this page previously said: LPJmL sums leaf, woody and belowground decomposition into one flux before applying the layer weights, so surface litter is carried down too, and Schaphoff et al. (2018) attribute the weights to cryoturbation and bioturbation.

Source

Schaphoff, S., von Bloh, W., Rammig, A., Thonicke, K., Biemans, H., Forkel, M., ... Waha, K. (2018). LPJmL4 - a dynamic global vegetation model with managed land - Part 1: Model description. *Geoscientific Model Development*, 11, 1343-1375. doi:10.5194/gmd1113432018, Sect. 2.5 and Eqs. 90-100. Rate constants and fractions are taken from the WHEP LPJmL 6.1.1 run configuration, which differs from the published values for `atmosphere_fraction`, `fast_fraction` and the fast-pool rate; see `soc_turnover_params`.

Examples

```
calculate_soc_lpjml(
  initial_soc_mgc_ha = 50,
  c_input_mgc_ha_yr = 2,
  years = 5
)
```

`calculate_soc_rothc` *Simulate soil organic carbon with the RothC five-pool model.*

Description

Annual RothC trajectory integrated by monthly Euler steps. Active carbon is partitioned into decomposable plant material, resistant plant material, microbial biomass and humified organic matter, plus a Falloon (1998) inert pool. Decomposed carbon is split between biomass and humus by a clay-texture function.

Usage

```
calculate_soc_rothc(
  initial_soc_mgc_ha,
  c_input_mgc_ha_yr,
  years,
  clay_pct = NA,
  climate_modifier = 1
)
```

Arguments

`initial_soc_mgc_ha`
: Initial soil organic carbon stock (Mg C per ha).

`c_input_mgc_ha_yr`
: Annual carbon input (Mg C per ha per year).

`years`
: Number of years to simulate.

`clay_pct`
: Soil clay content (percent).

`climate_modifier`
: Annual climate rate modifier (dimensionless), applied to every monthly decomposition step.

Value

A tibble with one row per year: `year`, `dpm`, `rpm`, `bio`, `hum`, `iom` and `soc_total`.

Source

Coleman, K. & Jenkinson, D. S. (1996). doi:10.1007/9783642610943_17; inert organic matter: Falloon, P. et al. (1998). doi:10.1016/S00380717(97)002563.

Examples

```
calculate_soc_rothc(
  initial_soc_mgc_ha = 50,
  c_input_mgc_ha_yr = 2,
  years = 5,
  clay_pct = 20
)
```

`calculate_soil_n2o` *Estimate direct soil N2O emissions from applied nitrogen.*

Description

Four emission-factor regimes for direct nitrous oxide from nitrogen applied to soil, `n2o_direct_n_t = n_input_t * ef`.

`"ipcc2019"` (the default) is the IPCC 2019 Tier 1 wet/dry disaggregation, keyed on each row's `moisture_regime` (`"wet"` or `"dry"`), with every factor read from ipcc_2019_soil_n_ef:

- `EF1` (IPCC 2019 Refinement Vol 4 Ch 11 Table 11.1 (Updated) p. 11.13): 0.016 for synthetic fertiliser and 0.006 for every other N input in wet climates, 0.005 for all N inputs in dry climates. Stamped `"ipcc2019_wet_dry"`.
- `EF1FR` (same table): 0.004, the aggregated flooded-rice factor, on every rice row in both moisture and both water regimes. No drainage layer exists for its continuous-flooding (0.003) versus drained (0.005) split, and upland rice, which the table prices with `EF1`, has no layer of its own either. Rice rows are those whose `item_cbs_code` is the CBS item of a production item classed `Crop_irrig == "Rice"` in items_prod_full (2807, rice and products). Stamped `"ipcc2019_ef1fr"`.
- `EF3PRP` (same table) on excreta deposited by grazing animals, as the N-weighted blend `excreta_cpp_share * EF3PRP,CPP + (1 - excreta_cpp_share) * EF3PRP,SO`, with `EF3PRP,CPP` 0.006 wet and 0.002 dry for cattle, poultry and pigs and `EF3PRP,SO` 0.003 for sheep and other animals. Excreta keep this factor on a rice row too: Equation 11.1 prices urine and dung deposited by grazing animals apart from the N applied to the crop. A row labelled by species reads its share from the label: 1 on `"Excreta_cattle_monog"`, 0 on `"Excreta_other"`. Stamped `"ipcc2019_ef3prp_blend"`.
- Excreta labelled so only as the fallback of a manure-type split (`method_excreta_species == "fallback_label_no_species"`) is collected manure applied to the land, not grazing deposition: it takes the factor of applied manure on its row (`EF1` for other N inputs, `EF1FR` on a rice item), whatever its missing species mix. Stamped `"ipcc2019_fallback_applied_manure"`.

"ipcc2019_atl_med" is the earlier IPCC 2019 Tier 1 option keyed on the Atlantic/Mediterranean climate instead: the n2o_efs_disaggregated table's two climate-level rows (irrig_type == "Tier_1" for ATL, irrig_type == "Med_average" for MED, 0.010 and 0.005), whatever the input. "aguilera" is a finer Mediterranean-calibrated disaggregation (Cayuela et al. 2017), selectable where its irrig_type/fert_type granularity is available and its regional emission factors apply: ef * mf, ef from n2o_efs_disaggregated on (irrig_type, climate), mf from fertiliser_n2o_modifiers on (fert_type, climate). An "Excreta" row, whose species are carried as a share rather than in its label, takes excreta_cpp_share * mf(Excreta_cattle_monog) + (1 - excreta_cpp_share) * mf(Excreta_other), stamped "aguilera_excreta_blend"; a fallback-labelled "Excreta" row, which is collected manure applied to the land and not grazing deposition, takes mf(Solid) (0.38 on MED, 0.60 on ATL), stamped "aguilera_fallback_applied_manure "ipcc2006" uses the n2o_efs_ipcc2006 table (IPCC 2006 Tier 1 defaults, flat 0.010 except flooded rice 0.003), keyed like "aguilera" on (irrig_type, climate) with no mf.

Usage

```
calculate_soil_n2o(
  x,
  method = c("ipcc2019", "ipcc2019_atl_med", "aguilera", "ipcc2006"),
  example = FALSE
)
```

Arguments

x	A tibble with n_input_t. method = "ipcc2019" requires fert_type ("Synthetic", "Solid", "Liquid", "Human", "Recycling", "SOM" or an excreta label: "Excreta", "Excreta_cattle_monog", "Excreta_other"), item_cbs_code and moisture_regime, plus excreta_cpp_share on "Excreta" rows. "ipcc2019_atl_med" requires climate. "aguilera" and "ipcc2006" require climate and irrig_type, and aguilera fert_type (and excreta_cpp_share on "Excreta" rows).
method	"ipcc2019" (default, IPCC 2019 Tier 1 wet/dry), "ipcc2019_atl_med" (IPCC 2019 Tier 1 on ATL/MED), "aguilera" (Mediterranean-calibrated) or "ipcc2006" (IPCC 2006 Tier 1).
example	If TRUE, return a small fixture instead of computing from x. Defaults to FALSE.

Details

method = "ipcc2019" aborts when moisture_regime is absent (whep_missing_moisture_regime), missing on any row (whep_moisture_regime_na, naming the count) or neither "wet" nor "dry" (whep_moisture_regime_value): which class a row belongs to is decided where the drivers are built, not here. An excreta row carrying N without an excreta_cpp_share aborts (whep_excreta_share_missing), as does a share outside 0 to 1 (whep_excreta_share_invalid) and a share on a species-labelled row that contradicts its label (whep_excreta_share_label_conflict); a row with no excreta N emits nothing whatever its mix. A fert_type with no direct factor in Equation 11.1 ("BNF", "Deposition") aborts (whep_soil_n2o_fert_type).

Value

x with n2o_direct_n_t and method_soil_n2o appended. method_soil_n2o names the factor each row took: the method name, or for "ipcc2019" one of "ipcc2019_wet_dry", "ipcc2019_ef1fr", "ipcc2019_ef3prp_blend", "ipcc2019_fallback_applied_manure", and for "aguilera" "aguilera_excreta_blend" on blended excreta rows and "aguilera_fallback_applied_man on fallback-labelled ones.

Source

IPCC (2019) 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol 4, Ch 11, Table 11.1 (Updated), p. 11.13.

Examples

```
calculate_soil_n2o(example = TRUE)
```

---

calculate_system_nue *System NUE*

Description

Calculates the NUE for Spain at the provincial level. The system NUE is defined as the percentage of total nitrogen production (total_prod) relative to the sum of all nitrogen inputs (inputs) into the soil system.

Usage

```
calculate_system_nue(n_soil_inputs = create_n_soil_inputs(), example = FALSE)
```

Arguments

n_soil_inputs	A tibble of nitrogen soil input (deposition, fixation, synthetic, manure, urban). If not provided and example = FALSE, it will be computed from create_n_soil_inputs().
example	If TRUE, return a small example output without downloading remote data. Default is FALSE.

Value

A tibble with the following columns:

- year: Year
- province_name: Spanish province
- total_prod: Total nitrogen production (Mg)
- inputs: Total nitrogen inputs (Mg)
- nue_system: System-level Nitrogen Use Efficiency (%)

Examples

```
calculate_system_nue(example = TRUE)
```

---

calculate_uncertainty_bounds

Calculate uncertainty bounds for livestock emissions.

Description

Applies IPCC uncertainty ranges to emission estimates. Multipliers sourced from `uncertainty_ranges` table (no hardcoded values).

Usage

```
calculate_uncertainty_bounds(data)
```

Arguments

`data`	Dataframe with emission columns from `calculate_livestock_emissions()`.

Value

Dataframe with added `_lower` and `_upper` columns for each emission estimate.

Examples

```
tibble::tibble(
  species = "Dairy Cattle",
  cohort = "Adult Female",
  heads = 1000,
  weight = 600,
  diet_quality = "High",
  milk_yield_kg_day = 20
) |>
  calculate_livestock_emissions() |>
  calculate_uncertainty_bounds() |>
  dplyr::select(species, cohort, heads,
    enteric_ch4_tier2, enteric_ch4_tier2_lower,
    enteric_ch4_tier2_upper, manure_ch4_tier2,
    manure_ch4_tier2_lower, manure_ch4_tier2_upper)
```

calculate_weed_bnf *Estimate symbiotic biological nitrogen fixation by weeds and cover crops.*

Description

Estimates symbiotic BNF from leguminous weeds and seeded cover crops. The legume fraction is a weighted average of spontaneous weeds and seeded cover crops. Environmental modifiers activate only when their driver columns are present.

Usage

```
calculate_weed_bnf(x, symbiotic_params = list())
```

Arguments

x	A tibble with `weed_npp_n_t`, `land_use`, `legumes_seeded` and `seeded_cover_crop_share`. Optional `legumes_spontaneous` overrides the land-use default; the same environmental driver columns as `calculate_crop_bnf()` are supported.
symbiotic_params	Named list overriding the symbiotic-BNF parameters (see `calculate_crop_bnf()`).

Value

The input tibble with `weed_ndfa_ref`, `weed_ndfa`, `weed_leg_share`, `f_env_weed` and `weed_bnf_t`.

Examples

```
calculate_weed_bnf(
  tibble::tibble(
    weed_npp_n_t = 10, land_use = "Cropland",
    legumes_seeded = 0, seeded_cover_crop_share = 0
  )
)
```

cb_processing *Commodity balance sheet processing fractions*

Description

Specifies the product fractions obtained when CBS items are processed, linking processed items to their output CBS categories.

Usage

```
cb_processing
```

Format

A tibble where each row corresponds to one processed-item / output-category combination. It contains the following columns:

- `ProcessedItem`: Name of the CBS item being processed (e.g., `"Apples and products"`, `"Barley and products"`).
- `item_cbs`: Name of the output CBS category produced by processing (e.g., `"Alcohol, Non-Food"`).
- `Product_fraction`: Conversion factor from processed input quantity to output product quantity. This can exceed 1 when the output includes added mass, such as water in beverages.
- `Value_fraction`: Economic value fraction associated with the output product (numeric; largely `NA` in current data).
- `Required`: Marks required co-product links in selected processing chains.

A fraction is a prior conversion rate per candidate output, not a mass-conserving recipe: `.correct_processed()` rescales it per area-year to reproduce the observed production of the output item, and several inputs declare a single lossy output (olives to olive oil at 0.20).

The one dairy row, `"Milk - Excluding Butter"` to `"Butter, Ghee"`, carries the milk churned into butter and ghee, which FAOSTAT's new FBS reports as milk's `processing` destiny from 2010 (the old FBS does not report it at all). Without it that mass was split onto food and feed, inflating 2010 world milk food by 30.5% (#757).

Source

Derived from FAOSTAT commodity balance sheet processing assumptions. The milk-to-butter fraction of 0.045 is the median `"Butter of Cow Milk"` extraction rate over the 69 countries reporting one in FAO (1997), *Technical Conversion Factors for Agricultural Commodities*, Rome: FAO, MILK section (range 3.3-7.3%). It is consistent with the ratio implied by the FBS itself: global butter production over milk processing is 0.047 in 2010 and 0.044-0.047 across 2010-2019.

Examples

```
head(cb_processing)
```

`cbs_trade_codes` *CBS to trade item code mapping*

Description

Maps detailed FAOSTAT trade item codes to their corresponding CBS item categories, enabling aggregation of bilateral trade data into the CBS framework.

Usage

```
cbs_trade_codes
```

Format

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

- `item_code_trade`: Numeric FAOSTAT trade item code (e.g., `15` for wheat).
- `item_trade`: Name of the trade item (e.g., `"Wheat"`, `"Flour, wheat"`, `"Bran, wheat"`).
- `item_cbs`: Name of the CBS category this trade item belongs to (e.g., `"Wheat and products"`).
- `item_check`: Cross-validation column repeating the mapped CBS name; used to flag mapping inconsistencies during data processing.

Details

Only single commodities are mapped. The FAOSTAT trade totals also carry group rows (codes from 1719 up, such as 1895 ”Beverages” or 1896 ”Tobacco”) that sum the items listed here, so mapping a group would count its members twice (whep#960).

Source

Derived from FAOSTAT Detailed Trade Matrix and commodity balance sheet correspondence tables.

Examples

```
head(cbs_trade_codes)
```

`cft_mapping` *FAOSTAT crop to LPJmL crop functional type (CFT) mapping*

Description

Maps FAOSTAT primary-production item codes to WHEP’s granular 33-class crop functional type taxonomy and the coarser LPJmL-compatible parent class. Used by `build_gridded_landuse()` and `run_spatialize()` to aggregate spatialized crop-level output into named crop functional types.

Usage

```
cft_mapping
```

Format

A tibble with one row per mapped FAOSTAT item. Columns:

- `item_prod_code`: Integer FAOSTAT item code.
- `item_prod_name`: Human-readable FAOSTAT item name.
- `cft_name`: Granular WHEP CFT name (33 classes, e.g. `"temperate_cereals"`, `"coffee"`, `"oil_crops_oilpalm"`).

- `cft_lpjml`: LPJmL-compatible parent class; one of the 12 LPJmL v6 named crop CFTs or `"others"`.
- `luh2_type`: LUH2 crop functional type (`c3ann`, `c4ann`, `c3per`, or `c3nfx`).

Details

Only the items listed here reach the grid, so each crop is listed on the code that carries its harvested area. Where `build_primary_production()` books the area of several co-products on one item (primary_double), that item is the one listed: Coconuts (248), not coconuts in shell (249); Linum (772), not linseed (333) or flax (773); Hemp (776), not hempseed (336) or true hemp (777); Kapok fruit (310), not kapok fibre (778). Linum takes linseed's `cft_name` and Hemp takes true hemp's, after the product with the larger FAOSTAT harvested area (2010: linseed 2.35 Mha against flax 0.22 Mha; true hemp 0.051 Mha against hempseed 0.005 Mha).

Source

Adapted from LandInG's `crop_types_FAOSTAT_LPJmL_default.csv` (Ostberg et al. 2023) with WHEP granular extensions.

Examples

```
head(cft_mapping)
```

---

check_arable_composition

Check where the arable-land extension changes composition mid-panel.

Description

Report, per term and `area_code`, whether a term of the fallow-inclusive arable land extension is present for part of a country's panel and absent for the rest. Two terms switch off inside the published panel and neither switch is a land-use change:

- `fodder` — the FAOSTAT fodder items (`Cat_1 == "Fodder_green"` and not grass, i.e. CBS 2000-2003). FAOSTAT's fodder tonnage (`faostat-production-old`, production only) ends in 2013 and EU AgriDB, the only other source, ends in 2019, so a build reaching 2020 has no fodder at all from that year on and every ordinary arable crop silently absorbs fodder's share of the country's arable land (whep#938).
- `temp_grassland_netting` — the temporary grassland netted out of the arable target. Modelled CBS 3002 comes from EU AgriDB alone, so it exists for 26 EU polities and stops at 2019; from 2020 the netting term is identically zero while FAO's arable land still contains temporary meadows, and the reconciliation changes method at the boundary (whep#937).

This is a diagnostic, not a correction: it flags the discontinuity so a series is not read across it. The treatments live behind `build_fao_arable_fallow_extension()`'s `fodder_gap` and `temp_grassland_basis` arguments, whose defaults reproduce the published behaviour.

Usage

```
check_arable_composition(extension, items_prod_full = whep::items_prod_full)
```

Arguments

`extension`	Tibble of the arable/permanent land extension as returned by `build_fao_arable_fallow_exten` `year`, `area_code`, `item_cbs_code`, `impact_u`, and optionally `temp_grassland_source` or `temp_grassland_netted_ha`. The `temp_grassland_netting` term is reported only when one of them is present. With `temp_grassland_source` a country-year counts as covered whenever a source supplied the term, zero included; with `temp_grassland_netted_ha` alone, only when it is positive.
`items_prod_full`	Crosswalk used to classify `item_cbs_code` as perennial via `Herb_Woody`. Defaults to items_prod_full.

Details

`check_series_jumps()` does not find either break by default. A term does not fall to a small value at the boundary, it stops having rows, and its `min_value` guard skips any pair involving a zero. With `dropouts = TRUE` it completes each series with zero and flags the stop, so `check_series_jumps(extension, impact_u, .by = c("area_code", "item_cbs_code"), dropouts = TRUE)` reports each fodder series at its break year. This function answers the coverage question per term and area instead, and also covers the netting term, which is not an item series.

Value

A tibble with one row per `(term, area_code)`:

- `term`: `"fodder"` or `"temp_grassland_netting"`.
- `area_code`: the country.
- `panel_first_year`, `panel_last_year`: the years that country has arable rows for.
- `term_first_year`, `term_last_year`: the years the term is present, as defined under `extension` (`NA` when it never is).
- `n_years_absent`: panel years in which the term is absent.
- `break_year`: the first panel year after `term_last_year` with no term (`NA` when the term runs to the end of the panel, or never appears).
- `broken`: `TRUE` when the term is present in some panel year and absent in a later one.
- `never_present`: `TRUE` when the term is absent for the whole panel.

Examples

```
extension <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~impact_u, ~temp_grassland_netted_ha,
  2019L, 10L, 2003L, 100, 50, # fodder mix, netting fires
  2019L, 10L, 2511L, 900, 50,
  2020L, 10L, 2511L, 1000, 0 # fodder gone, netting off
```

```
)
check_arable_composition(extension)
```

`check_extent_jumps` *Flag implausible year-on-year jumps in a per-unit extent*

Description

Run `check_series_jumps()` on `extent_ha` per unit series, the check the plan's seam section asks for before an extent modulates a share. WHEP repairs LUH2's isolated single-year cropland collapses nationally (`.fix_luh2_crop_collapse()`, `R/build_production.R:627-695`); the same collapse reaches a unit series unrepaired, where it would propagate into every earlier back-cast year through the growth chain.

The scan covers whatever years `extent` carries. `backcast_admin_shares()` calls it on the back-cast window alone – `t0` and everything before it, the years the modulation actually reads – so a jump after the seam is not a reason to refuse a back-cast.

Usage

```
check_extent_jumps(
  extent,
  ratio_bounds = c(0.55, 1.6),
  min_value = 0,
  verbose = FALSE
)
```

Arguments

`extent`	A per-unit extent table as returned by `aggregate_unit_extent()`. Every column that is not `year`, `extent_ha` or `extent_basis` identifies the series.
`ratio_bounds`	Length-2 numeric `c(low, high)` plausible band for the ratio of consecutive years, passed to `check_series_jumps()`, whose own default it inherits.
`min_value`	Minimum extent both members of a pair must exceed to be flagged, passed to `check_series_jumps()`.
`verbose`	Logical, passed to `check_series_jumps()`. Default `FALSE`, so a guard inside a back-cast stays quiet.

Value

The `check_series_jumps()` flags tibble: the series key columns, `year`, `prev_value`, `value`, `ratio` and `allowlisted`.

Examples

```
extent <- tibble::tibble(
  area_code = 900L,
  level_polity_code = "A1",
  level = 1L,
  year = 1900:1903,
  extent_ha = c(1000, 1010, 5, 1020),
  extent_basis = "cropland_ha"
)
check_extent_jumps(extent)
```

---

check_fodder_land_share

Check how much arable land the fallow split attributes to fodder crops.

Description

Report, per `(year, area_code)`, the share of the reconciled arable land extension that lands on the FAOSTAT fodder items (commodity-balance items of `Cat_1 == "Fodder_green"` that are not grass, i.e. 2000-2003).

With `fallow_weights = NULL` — the default, and the path the land-balance footprint (`build_land_balance_footprint()`) takes — `build_fao_arable_fallow_extension()` rescales every arable crop's cropped physical area proportionally up to FAO Arable land. Each crop's share of the output is therefore exactly its share of the input harvested area, so an inflated fodder harvested area is passed straight through into the published per-crop arable land footprint, taking land away from the ordinary crops (whep#356). Fodder harvested area is reconstructed rather than surveyed (dry-matter yield imputation, EU AgriDB splicing, linear filling), so it is the term most likely to be wrong, and in some country-years it alone exceeds the country's whole FAO arable land.

This is a diagnostic, not a correction: nothing is rescaled or dropped. It flags where the attribution is implausible so the fodder reconstruction can be inspected, or agro-climatic `fallow_weights` (see `gridded_fallow_weights()`) supplied instead.

Usage

```
check_fodder_land_share(
  extension,
  threshold = 0.5,
  items_prod_full = whep::items_prod_full
)
```

Arguments

`extension`	Tibble of the arable/permanent land extension with columns `year`, `area_code`, `item_cbs_code` and `impact_u`, as returned by `build_fao_arable_fallow_extension()`.

`threshold` Share of arable land above which a `(year, area_code)` is flagged (default `0.5`). Fodder is real land use, so a moderate share is expected; half a country's arable land is not.

`items_prod_full`
Crosswalk used to classify `item_cbs_code` as perennial via `Herb_Woody`. Defaults to items_prod_full.

Value

A tibble with one row per `(year, area_code)` that has at least one arable row in `extension`, ordered by descending `fodder_share`:

- `year`, `area_code`: the country-year.
- `fodder_ha`: arable land attributed to fodder items.
- `arable_ha`: total arable (non-perennial) land attributed.
- `fodder_share`: `fodder_ha / arable_ha` (`NA` when `arable_ha` is zero).
- `flagged`: `TRUE` when `fodder_share > threshold`.

Examples

```
extension <- tibble::tribble(
  ~year, ~area_code, ~item_cbs_code, ~impact_u,
  2000L, 10L, 2003L, 700, # fodder mix
  2000L, 10L, 2511L, 300 # wheat
)
check_fodder_land_share(extension)
```

`check_footprint_conservation` *Check footprint conservation against direct extensions.*

Description

Verify the master input-output accounting identity for a footprint: the environmental pressure embodied across all final demand and traced back to an origin sector should equal the direct extension (the source pressure) of that sector.

The footprint engine zeroes negative coefficients, caps column sums, and drops near-zero-output sectors (the FABIO conventions), so the identity holds only approximately. This check quantifies the discrepancy per origin sector instead of asserting exact equality. Crucially it detects *under-tracing* (pressure that silently disappears and never reaches final demand), which `compute_footprint()`'s `conserve_extensions` bounding never reports because it only rescales results downward.

Only the positive side of the extensions is traced, matching the engine, which traces `pmax(extensions, 0)` and ignores sectors with output `<= output_tol`.

Usage

```
check_footprint_conservation(
  footprint,
  extensions,
  labels,
  x_vec,
  output_tol = 1e-08,
  tol = 0.01
)
```

Arguments

`footprint`	Footprint tibble from `compute_footprint()`, with `origin_area`, `origin_item` and `value` columns.
`extensions`	Numeric vector of environmental extensions per sector, as passed to `compute_footprint()`.
`labels`	Tibble with `area_code` and `item_cbs_code` mapping each sector to its meaning, as passed to `compute_footprint()`.
`x_vec`	Numeric vector of total output per sector.
`output_tol`	Minimum output for a sector to be traceable. Sectors with `x_vec <= output_tol` contribute zero direct pressure, matching `compute_footprint()`.
`tol`	Relative tolerance for the conservation status. Discrepancies within `tol` of the direct pressure are `"ok"`.

Value

A tibble with one row per origin sector:

- `origin_area`: Country where the pressure occurs.
- `origin_item`: Item causing the pressure.
- `direct`: Direct (source) extension for the sector.
- `embodied`: Footprint traced back to the sector.
- `discrepancy`: `embodied - direct`.
- `rel_discrepancy`: `discrepancy / direct` (`NA` when `direct` is zero).
- `status`: One of `"ok"`, `"under_traced"`, `"dropped"` (embodied is zero while direct is positive) or `"over_traced"`. Rows are ordered by descending absolute relative discrepancy.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
y_mat <- matrix(c(85, 195), ncol = 1)
extensions <- c(50, 30)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(1L, 2L)
```

```
)
fp <- compute_footprint(
  x_vec = x_vec, y_mat = y_mat, extensions = extensions,
  labels = labels, z_mat = z_mat
)
check_footprint_conservation(fp, extensions, labels, x_vec)
```

`check_inputs_supplied` *Refuse an input that was never supplied*

Description

Assert that each named input **arrived and is not vacuous**, rather than that the totals it feeds reconcile. An input is *supplied* when its column exists and holds at least one value that is neither missing nor zero; it is *absent* when the column is missing, or present and identically zero or missing in every row.

Usage

```
check_inputs_supplied(
  data,
  required,
  stamp = NULL,
  action = c("abort", "warn"),
  details = NULL
)
```

Arguments

`data`	A tibble carrying the inputs to check.
`required`	Column names to check, optionally named. A name is the key used in messages and matched against `stamp`; an unnamed entry uses the column name as its own key.
`stamp`	Optional name of a provenance column written by `stamp_inputs_supplied()`. Used when `data` carries it, ignored otherwise.
`action`	`"abort"` (default) to raise, `"warn"` to carry on. Abort where a wrong answer is worse than no answer – a density whose denominator is inflated, a stressor that would ship as zero. Warn where a caller reading only the columns that did arrive is unaffected.
`details`	Extra `cli` bullets appended to the message, for the caller to name its source and its remedy.

Value

`data`, invisibly. Raises or warns with class `whep_absent_input` when an input is absent; the condition carries the absent keys in its `absent` field.

Why reconciliation cannot do this

Zero satisfies a sum. A pin published without its inland water and ice layers booked 1,643 Mha of lakes and 618 Mha of glaciers as land, and `polity_area_ha == land_area_ha + inland_water_ha + ice_area_ha` still held to `max |residual| = 0 ha`, because both absent layers were zero (whep#885, whep#1010). Three FAOSTAT emission labels lost to a pin-vocabulary change summed to a literal zero through `sum(na.rm = TRUE)`, and a test named *"enteric_ch4_kt conservation is exact"* went on passing, because zero distributes to zero (whep#1016). A consistency identity, a conservation check and a row count are all satisfied by an absence. Only an assertion that the input was supplied is not.

Supplied is a label first, a value test second

When `stamp` names a column the producer wrote, that column decides: it is a comma-separated list of the keys that were actually consumed, written by `stamp_inputs_supplied()`, and no arithmetic downstream can forge it. The value test is the fallback for a table published before its producer carried a stamp. The fallback is deliberately **scale-free** – identically zero, not a magnitude floor – so that it holds on a single-country development build as well as on a global one. A floor belongs at publication time, where the expected scale is known.

What it cannot see

Three absences pass this check, and a caller must not read a pass as more than it is.

- A **partial** absence. One non-zero value anywhere in the column is enough; a layer supplied for three countries out of two hundred is supplied as far as this is concerned.
- A **zero-row** input. Zero rows is a filter that matched nothing, not a zero-filled column, and the caller that wrote the filter is the one to answer for it. Columns are judged only when `data` has rows.
- A value the **code failed to reach**. If the lookup was never called, the quantity is absent from the code path and not from the data, and that is a defect rather than a fill (whep#1034). Run the lookup and look at what it returns.

Examples

```
supplied <- tibble::tibble(land_ha = c(1, 2), water_ha = c(0.5, 0))
check_inputs_supplied(supplied, c("land_ha", "water_ha"))

absent <- tibble::tibble(land_ha = c(1, 2), water_ha = c(0, 0))
try(check_inputs_supplied(absent, c(land = "land_ha", water = "water_ha")))
```

`check_keys_complete` *Refuse an aggregate built over an incomplete key lattice*

Description

Assert that every key an expected lattice requires is supplied by some row, and abort naming the keys that are **missing** rather than the rows that are present. This is the absent-row sibling of the absent-column and absent-label guards: an aggregate over eleven of twelve months is a number with nothing wrong with it except that it is wrong, and no `NA` policy, identity or row count can tell it from a twelve-month one.

The gaps themselves come from `key_lattice_gaps()`; see its documentation for the forms `expected` may take and for the staged cost that makes the check affordable on a lattice of tens of millions of keys.

Usage

```
check_keys_complete(
  data,
  expected,
  .by = NULL,
  action = c("abort", "warn"),
  details = NULL
)
```

Arguments

`data`	A tibble to check.
`expected`	The expected key set; see `key_lattice_gaps()`.
`.by`	Character vector of grouping columns, or `NULL` for one group.
`action`	`"abort"` (default) to raise, `"warn"` to carry on.
`details`	Extra `cli` bullets appended to the message, for the caller to name its source and its remedy. Interpolated in the caller's own environment, so a bullet may name the caller's variables.

Value

`data`, invisibly. Raises or warns with class `whep_incomplete_lattice` (and `whep_absent_input`) when a key is missing; the condition carries the gap tibble in its `missing` field.

Abort versus warn is the caller's decision, not this function's

Both are wrong as a blanket policy, so neither is imposed.

- **Abort** is right where the aggregate is the deliverable and a short-summed one would ship: an eleven-month annual water balance drives nitrogen leaching downstream, and no consumer of the annual total can recover the twelfth month. It is wrong where a legitimately partial period exists – a model run still writing its current year – because it blocks a build over the complete years for the sake of an incomplete one the caller never asked for.
- **Warn** keeps that build alive, and is what a caller who has decided the gap is acceptable should select explicitly. It must not be the default: two warnings were already being

printed when whep#1010 shipped 533 Mha of lakes and glaciers as land, so a warning inside a build that runs for hours gates nothing.

The default is therefore `"abort"`, the most rigorous option, and a caller who wants the build to continue says so and is recorded as having said so. A third policy – drop the incomplete groups, so the aggregate is absent rather than wrong – is deliberately not an `action` here, because this function returns `data` unchanged and a silent drop is the very thing being fixed. Do it with `key_lattice_gaps()` and an anti-join at the site, where the reader can see which groups went and why.

Examples

```
complete <- tibble::tibble(year = 2000L, month = 1:12, value = 1)
check_keys_complete(complete, list(month = 1:12), .by = "year")

eleven <- tibble::tibble(year = 2000L, month = c(1:6, 8:12), value = 1)
try(check_keys_complete(eleven, list(month = 1:12), .by = "year"))
```

---

check_labels_supplied

Refuse a value vocabulary that no longer contains the labels a filter needs

---

Description

Assert that each label a downstream `filter()` selects on actually occurs in the column it selects from. A label that has been renamed upstream does not raise: the filter matches no rows, the sum of no rows is zero, and the zero is indistinguishable from a measurement.

This is the sibling of `check_inputs_supplied()` for the case where the column is present and populated but its **vocabulary** has moved. It is the shape of whep#1016, where three FAOSTAT emission `Element` labels were lost to a pin revision and roughly 108 Tg CH4/yr shipped as literal zero, and it is invisible to any check written around `coalesce()`, `replace_na()` or `na.rm` – none of those appear anywhere near it.

The message names the labels that are **present**, up to six of them, as well as the ones that are missing. A rename is only obvious once the new spelling is in front of the reader.

A table with **no rows** carries no vocabulary and is passed, for the same reason `check_inputs_supplied()` does not judge one: an empty table is an absent table rather than a moved label, and the caller that produced it is the one to answer for it. The case this is for – whep#1016 – is a pin of 2.5 million rows whose labels moved, not an empty one.

Usage

```
check_labels_supplied(
  data,
  column,
  labels,
  action = c("abort", "warn"),
  details = NULL
)
```

Arguments

`data`	A tibble carrying the vocabulary to check.
`column`	Name of the column the labels are drawn from. A column that is not there at all is reported as the whole vocabulary being absent, even on a table with no rows.
`labels`	Labels that must each occur at least once. Compared as character, so a numeric code may be given as a number.
`action`	`"abort"` (default) to raise, `"warn"` to carry on.
`details`	Extra `cli` bullets appended to the message.

Value

`data`, invisibly. Raises or warns with class `whep_absent_label` (and `whep_absent_input`) when a label is missing; the condition carries the missing labels in its `absent` field and what was seen instead in its `observed` field.

Examples

```
landuse <- tibble::tibble(Element = c("Area", "Area"), Value = c(1, 2))
check_labels_supplied(landuse, "Element", "Area")

renamed <- tibble::tibble(Element = c("Area under cultivation"), Value = 1)
try(check_labels_supplied(renamed, "Element", "Area"))
```

---

check_parquet_integrity

Check the structural integrity of a Parquet file.

Description

Detect Parquet files whose footer describes a layout the file itself does not have. The motivating failure is `nanoparquet` before 0.5.0, which stored column-chunk file offsets and sizes as 32-bit integers: past 4 GiB (2^32 bytes) they wrapped around, so a multi-gigabyte cube received a footer that still declared every row group and row but pointed at the wrong bytes for most of them. A reader then returns the first ~4 GiB and throws thrift `"Deserializing page header failed"` on everything after it, which a consumer that does not read row group by row group never sees: it silently gets truncated data (whep#531).

The default check is metadata-only and therefore costs milliseconds even on a 15 GB file. Column chunks are written contiguously in `(row_group, column)` order by every mainstream writer, so their byte ranges must not overlap and must stay inside the file. An offset that jumps backwards is the signature of the 32-bit wraparound. The upper bound is the start of the footer rather than the end of the file, so a chunk that runs into the metadata is caught too.

`deep = TRUE` additionally decodes every row group with `arrow`, which is exact rather than structural but reads the whole file.

Usage

```
check_parquet_integrity(path, deep = FALSE)
```

Arguments

`path`	Path to a Parquet file.
`deep`	Whether to also read every row group and report the ones that fail to decode. Exact, but reads the entire file.

Value

A tibble with one row per anomaly, empty when the file is sound:

- `row_group`: Zero-based row group index.
- `column`: Zero-based column index, `NA` for whole-row-group problems found by `deep = TRUE`.
- `issue`: One of `"offset_overlap"` (a chunk starts before the previous chunk ends, i.e. offsets are not monotonic), `"offset_past_data"` (a chunk ends beyond the last byte the data section can hold) or `"row_group_unreadable"` (`deep = TRUE` only).
- `chunk_start`, `chunk_end`: Byte range the footer claims for the chunk.
- `detail`: Human-readable context, e.g. the reader's error.

Examples

```
path <- tempfile(fileext = ".parquet")
nanoparquet::write_parquet(data.frame(x = 1:10), path)
check_parquet_integrity(path)
unlink(path)
```

`check_series_jumps` *Flag implausible year-on-year jumps in a time series.*

Description

Scan each series for consecutive observations whose ratio falls outside a plausible band, the level-2 (within-series) detector of the AFE data and code validation framework. It catches unjustified single-year steps, spikes and the level shifts that appear where two sources are spliced, while a break-year allowlist keeps documented regime changes from firing as false positives.

Usage

```
check_series_jumps(
  data,
  value_col,
  time_col = year,
  .by = NULL,
  ratio_bounds = c(0.55, 1.6),
  bands = NULL,
  min_value = 0,
  consecutive_only = TRUE,
  allowlist = NULL,
  verbose = TRUE,
  dropouts = FALSE
)
```

Arguments

`data`	A data frame with one observation per row.
`value_col`	The column holding the series values to scan.
`time_col`	The column holding time values. Default: `year`.
`.by`	A character vector of grouping columns identifying each series (optional). When `NULL`, the whole table is one series.
`ratio_bounds`	Length-2 numeric `c(low, high)`: the default plausible band for the ratio of consecutive values. A ratio below `low` or above `high` is flagged.
`bands`	Optional data frame of per-group band overrides: the grouping columns (a subset of `.by`) plus `lo` and `hi`. Where a group matches, its `lo`/`hi` replace `ratio_bounds` for that group.
`min_value`	Minimum value both members of a pair must exceed to be flagged. Keeps genuine near-zero technology onsets from firing. Default: `0`.
`consecutive_only`	Logical. If `TRUE` (default), only pairs one time step apart are scanned; larger gaps are skipped.
`allowlist`	Optional data frame of documented break years, matched on the grouping columns plus `time_col`. Matching flags are returned with `allowlisted = TRUE` rather than dropped.
`verbose`	Logical. If `TRUE` (default), report flag counts with `cli`.
`dropouts`	Whether to also flag a series that stops: one that falls from above `min_value` to exactly zero, or has no row at a time value its panel has after the series began. Each series is completed with zero over those time values, so the flag carries `value = 0` and `ratio = 0`, and the step back up from zero is an onset the `min_value` gate keeps quiet. `FALSE` (default) scans only the rows supplied and skips any pair involving a zero. `TRUE` takes the panel to be every time value in `data`. A character vector, a subset of `.by`, takes it to be the time values present within that coarser group instead: with `.by = c("area_code", "item_cbs_code")`, `dropouts = "area_code"` flags an item that stops while its country goes on, and not a country that leaves the panel.

Details

This is the first landed detector of the reusable `check_*` library described in the AFE data-validation framework decision (`afse-wiki/wiki/decisions/afse-data-validation-framework.md`, level 2, within-series). It generalises the energy-hist consecutive-year jump scan (a world-total ratio scan against a break-year allowlist) into a grouped, band-parameterised check usable both as an inline pipeline guard and as a test backend.

Two pieces of the framework's metadata model are exposed as arguments. `bands` supplies per-variable plausible-jump bands (land area is tight, yield is wide, so a single global band is wrong for a mixed panel), and `allowlist` supplies the historically justified break years that are reported but not treated as defects. Undocumented jumps stay flagged.

A robust variant (flagging via the median absolute deviation of log ratios, per the Hampel/MAD anchors in the framework) is a documented future extension, not implemented here.

A term that switches off inside a panel is not a jump by default: its series stops having rows, or falls to zero, and the `min_value` gate skips any pair involving a zero. That is how the FAOSTAT fodder items leave the arable land extension at 2020 without a flag (whep#938). `dropouts` makes such a stop a flagged step, and the `allowlist` can then mark the documented ones.

Value

A tibble with one row per flagged jump: the grouping columns, the `time_col` of the later observation, `prev_value`, `value`, `ratio` (`value / prev_value`) and `allowlisted`. When nothing is flagged, a zero-row tibble of the same shape and column types.

Examples

```
series <- tibble::tibble(
  category = rep(c("area", "yield"), each = 5),
  year = rep(2000:2004, times = 2),
  value = c(100, 101, 102, 180, 181, 3.0, 3.1, 5.2, 3.0, 3.1)
)
# area steps 102 -> 180 (1.76x); yield steps 3.1 -> 5.2 (1.68x)
check_series_jumps(series, value, .by = "category")

# Widen the band for yield only, leaving area on the default:
bands <- tibble::tibble(category = "yield", lo = 0.4, hi = 2.5)
check_series_jumps(series, value, .by = "category", bands = bands)
```

---

check_supply_use_balance

Check the commodity balance sheet supply-use identity.

Description

Verify that total supply equals total use for every row of a wide commodity balance sheet, the fundamental accounting identity behind the input-output model. Supply is `production + import + stock_withdrawal`; use is `export + food + feed + seed + processing + processing_primary + other_uses + stock_addition`. `processing_primary` is treated as `0` when the column is absent, since it is only nonzero for the handful of primary items whose entire domestic supply is destined for processing (see `get_wide_cbs()`).

Usage

```
check_supply_use_balance(cbs, tol = 1e-06)
```

Arguments

`cbs`	Wide commodity balance sheet, e.g. from `get_wide_cbs()`.
`tol`	Absolute tolerance (in the data's mass units) for a row to count as balanced.

Value

A tibble with the key columns present in `cbs` plus:

- `supply`: Total supply.
- `use`: Total use.
- `abs_diff`: `abs(supply - use)`.
- `rel_diff`: `abs_diff / supply` (`NA` when supply is zero).
- `balanced`: `TRUE` when `abs_diff <= tol`. Rows are ordered by descending absolute difference.

Examples

```
get_wide_cbs(example = TRUE) |>
  check_supply_use_balance()
```

`check_table_schema` *Check a table against a declarative schema.*

Description

Validate a tibble against a serializable schema and return one row of diagnostics per violation. The input is never modified, so this is the read-only counterpart of `ensure_columns()`, which *coerces* a table to a typed prototype. Use `ensure_columns()` to reach a schema and `check_table_schema()` to prove a table is already there.

The schema is plain data — nested lists of scalars and atomic vectors — so it round-trips through `yaml` or `jsonlite` unchanged and can live next to the artifact it describes. Project vocabularies and scientific bounds stay in the caller's schema; this function hard-codes none of them.

Usage

```
check_table_schema(data, schema)
```

Arguments

`data`	Table to validate. Not modified.
`schema`	Declarative schema, as described above.

Value

A tibble of diagnostics, empty when `data` conforms.

Schema representation

A list with these fields, all optional except `columns`:

- `columns`: An *ordered* list of column specifications (see below). A list, not a named list, so the declared order survives any serialization. May be empty.
- `key`: Character vector of column names that must jointly be unique. Every name must be declared in `columns`.
- `extra_columns`: `"allow"` (default) or `"forbid"` for columns the schema does not declare.
- `column_order`: `"ignore"` (default) or `"strict"`. Under `"strict"` the declared columns that are present must appear in the declared relative order.
- `allow_empty`: Whether a zero-row table is acceptable. `TRUE` by default; set `FALSE` to flag an empty result.

Each column specification is a list with:

- `name`: Column name. Required.
- `type`: One of `"logical"`, `"integer"`, `"double"`, `"character"`, `"Date"`, `"list"` or `"any"`. Required. Types are compared exactly (`vctrs::vec_is()`): an `integer` column does not satisfy `"double"`. `"any"` skips the type check.
- `required`: Whether the column must be present. `TRUE` by default.
- `allow_missing`: Whether `NA` is acceptable. `TRUE` by default.
- `min`, `max`: Inclusive bounds, for `"integer"`, `"double"` and `"Date"` only.
- `allowed`: Permitted values, i.e. a caller-owned vocabulary.
- `unique`: Whether values must be unique within the column. `FALSE` by default.
- `severity`: `"error"` (default) or `"warning"` for every diagnostic attributed to this column.

`min`, `max`, `allowed`, `unique` and `allow_missing = FALSE` do not apply to `"list"` or `"any"` columns. Unknown fields, at either level, abort rather than being ignored, so a mistyped `minimum` cannot silently disable a bound.

Diagnostics

One row per violation, ordered deterministically: table-scope rules first, then each declared column in schema order (by row, then rule), then undeclared columns in input order, then key duplicates. Columns:

- `row`: Row index in `data`, `NA` for table- and column-scope rules.
- `column`: Column name, `NA` for table-scope rules.
- `rule`: One of `"empty_table"`, `"column_order"`, `"missing_column"`, `"unexpected_column"`, `"type_mismatch"`, `"missing_value"`, `"below_min"`, `"above_max"`, `"not_allowed"`, `"duplicate_value"`, `"duplicate_key"`.
- `value`: The offending value, formatted, `NA` where no single value is at fault.
- `severity`: `"error"` or `"warning"`, from the column specification. Table-scope rules are always `"error"`.
- `detail`: Human-readable context.

A column whose type does not match reports `type_mismatch` and its value rules are skipped, so a wrongly typed column yields one diagnostic rather than one per row.

Examples

```
# A keyed long table with a vocabulary and a scientific bound.
schema <- list(
  columns = list(
    list(name = "year", type = "integer", min = 1961, max = 2023),
    list(name = "area_code", type = "integer", allow_missing = FALSE),
    list(
      name = "source",
      type = "character",
      allowed = c("FAOSTAT_prod", "LUH2")
    ),
    list(name = "value", type = "double", min = 0)
  ),
  key = c("year", "area_code")
)
data <- tibble::tibble(
  year = c(2000L, 2000L, 1900L),
  area_code = c(4L, 4L, 8L),
  source = c("FAOSTAT_prod", "guess", "LUH2"),
  value = c(1, -2, 3)
)
check_table_schema(data, schema)

# The schema is data: it survives a YAML round trip unchanged.
identical(
  check_table_schema(data, yaml::yaml.load(yaml::as.yaml(schema))),
  check_table_schema(data, schema)
)

# A structurally different schema: closed column set, strict order,
# no key, a non-empty requirement and a list column.
```

```
manifest_schema <- list(
  columns = list(
    list(name = "built_at", type = "Date"),
    list(name = "inputs", type = "list")
  ),
  extra_columns = "forbid",
  column_order = "strict",
  allow_empty = FALSE
)
check_table_schema(tibble::tibble(inputs = list()), manifest_schema)
```

`classify_sjos_n` *Classify crops into the 2-way SJOS-N safe-and-just space.*

Description

Crosses the ecological boundary side with the nourishment side into the six ordered sjos_levels. Per crop (`item_cbs_code`), the boundary side is `"Exceedance"` when the crop-country-year's `exceedance_n_t` is positive and `"Within_boundary"` otherwise (the all-zero and missing cases fall to `"Within_boundary"`). The country's nourishment class (`nourish`, from `normalize_nourishment()`) is joined by `year` and `area_code` and broadcast to each of its crops. The classification `paste(boundary_side, nourish)` is one of `"Within_boundary Under"` ... `"Exceedance Over"`, returned as a factor with all six `sjos_levels$level` levels. This reproduces Global's 2-way remap (`Global/R/sjos_n.r:363`) at the per-`item_cbs` granularity Module 4's footprint needs.

The boundary side reads `exceedance_n_t`, the crop's signed share of its source cell's overshoot `pmax(actual - critical, 0)`, summed to the country (see `build_n_boundary_exceedance()`). The cell comparison is made before any crop attribution and the overshoot is not capped at the actual pressure. With `negative_critical = "keep"`, a cell whose critical surplus is negative exceeds even at zero actual surplus, its overshoot is larger than its actual surplus, and its `within_boundary_n_t` is negative; with `"clamp"` that cell's allowance is zero instead. A crop whose own surplus is negative inside an exceeding cell receives a negative exceedance share, so it falls on the `"Within_boundary"` side unless positive shares from other cells of the same country outweigh it.

Usage

```
classify_sjos_n(exceedance, nourishment, level_col = sjos_class)
```

Arguments

`exceedance`	A `build_n_boundary_exceedance()` output at `resolution = "country"`, keyed by `year`, `area_code`, `item_cbs_code` with the mass terms `exceedance_n_t`, `within_boundary_n_t`, `actual_n_t`.
`nourishment`	A `normalize_nourishment()` output carrying `year`, `area_code` and the `nourish` class (`"Under"` / `"Adequate"` / `"Over"`), one row per country-year.
`level_col`	The unquoted name for the classification column. Defaults to `sjos_class`.

Value

A tibble keyed by `year`, `area_code`, `item_cbs_code` with the mass terms `exceedance_n_t`, `within_boundary_n_t`, `actual_n_t`, the joined `nourish` class, the `boundary_side` and the classification column (a factor over `sjos_levels$level`, named by `level_col`).

Examples

```
classify_sjos_n(
  exceedance = tibble::tribble(
    ~year,
    ~area_code,
    ~item_cbs_code,
    ~exceedance_n_t,
    ~within_boundary_n_t,
    ~actual_n_t,
    2010L, 10L, 2511L, 5, 3, 8,
    2010L, 10L, 2513L, 0, 4, 4
  ),
  nourishment = tibble::tribble(
    ~year, ~area_code, ~nourish,
    2010L, 10L, "Over"
  )
)
```

`climate_mcf` *Climate-zone MCF values.*

Description

Methane Conversion Factors by MMS type and climate zone (Cool/Temperate/Warm).

Usage

```
climate_mcf
```

Format

A tibble with `mms_type`, `climate_zone`, `mcf_percent`.

Source

Predominantly the 2006 Guidelines, Vol 4, Ch 10, Table 10.17, **not** the 2019 Refinement, and with the same provenance profile as ipcc_2019_mcf_manure, which holds the same values under the Tier 1 system labels. Since whep#1022 this object is **no longer the default** the Tier 2 path reads: `.calc_manure_ch4_tier2()` weights the `edition == "ipcc_2019"` rows of climate_mcf_ipcc by the manure-system mix, and this table is reached only with `mcf_source = "as_shipped"`. Verified against both editions:

- Matching both: daily spread 0.1/0.5/1.0, solid storage 2.0/4.0/5.0, poultry manure 1.5 and burned for fuel 10.

- The 2006 edition only: pasture/range/paddock 1.0/1.5/2.0, against a single 0.47 percent in the 2019 Refinement.
- Derived from the 2006 per-degree rows, but not at the class bound: liquid/slurry 17/35/80 takes 35 from the 18 degree column rather than the 42 of 20 degrees, and anaerobic lagoon 66/73/80 takes 73 from the 14 degree column rather than the 78 of 20 degrees.
- Matching neither edition: dry lot 1.5/2.5/4.0 (both give 1.0/1.5/2.0) and the single all-climate values for intensive-windrow composting 0.5 and passive-windrow composting 1.0, which both editions resolve by climate (2006 gives 0.5/1.0/1.5 for each; the 2019 Refinement 0.5/1.0/1.5 and 1.0/2.0/2.5). `"Anaerobic Digester"` 0 is published by neither: 2006 gives the range 0-100 percent and requires a calculation, the 2019 Refinement six leakage-and-storage classes spanning 1.00 to 13.17 percent. **Assumed, unverified.**

Only six of the eleven systems here are reachable on the live path: regional_mms_distribution routes manure to daily spread, solid storage, liquid/slurry, the anaerobic lagoon, pasture/range/paddock and poultry manure, and `.calc_weighted_mcf()` aborts on any label outside that set. So the dry-lot, composting and anaerobic-digester cells enter no published number today, and the four rows keyed `climate_zone "All"` are unreachable for a second reason: the MCF join is on `(mms_type, climate_zone)` and never asks for `"All"`. The cells that both are live and depend on the edition are pasture/range/paddock, liquid/slurry and the anaerobic lagoon. climate_mcf_ipcc carries the as-published alternative for each and now supplies the default.

Tracked in whep#601 and whep#1022.

Examples

```
climate_mcf
```

`climate_mcf_ipcc` *As-published MCF alternatives to climate_mcf.*

Description

Methane conversion factors transcribed from Table 10.17 of each IPCC edition, over the same `mms_type` / `climate_zone` key space as climate_mcf so that either can be substituted for it. Selected with the `mcf_source` manure-engine option (see manure_engine_options). Since whep#1022 the `edition == "ipcc_2019"` rows are the **default**: the 2019 Refinement is the current IPCC guidance, and the six cells of climate_mcf whose provenance could not be established should not be what ships. climate_mcf stays reachable as `mcf_source = "as_shipped"` so an older run can be reproduced.

Six cells of climate_mcf match no published IPCC value, and three of the `mms_type` labels that carry them – dry lot, the two composting rows and the anaerobic digester – are unreachable on the live Tier 2 path, because regional_mms_distribution routes manure to only six systems and none of them is one of those. The cells that *are* live and edition-dependent are pasture/range/paddock, liquid/slurry and the anaerobic lagoon, and it is those three that the default change moves.

Usage

```
climate_mcf_ipcc
```

Format

A tibble with columns:

edition `"ipcc_2006"` or `"ipcc_2019"`, the two values the `mcf_source` option takes.

mms_type Manure-management system, in the climate_mcf vocabulary, plus the six 2019 anaerobic-digester classes.

climate_zone `"Cool"`, `"Temperate"` or `"Warm"`.

mcf_percent Methane conversion factor (percent of `Bo` achieved). `NA` where the edition publishes no default – the 2006 anaerobic digester, which is "0-100 percent, calculate".

paired_bo_m3_kg_vs The methane potential (`Bo`, m3 CH4 per kg volatile solids) the edition publishes this MCF together with, or `NA` where it publishes none and the animal-category `Bo` of ipcc_tier2_bo_values applies. Only the three 2019 pasture/range/paddock rows carry one, 0.19 (whep#1137).

Collapse rules

Neither edition publishes exactly three numbers for every system, so two collapse rules are applied. Both are WHEP choices, stated here because they are not IPCC statements – and because the 2019 edition is now the default, the second of them is live on every Tier 2 run:

- **2006 per-degree rows** (liquid/slurry, uncovered anaerobic lagoon) are read at the middle column of each temperature class. Table 10.15 of the same chapter defines the classes as Cool below 15 C, Temperate 15 to 25 C and Warm above 25 C, and Table 10.17 groups its 19 per-degree columns the same way, so the middle columns are 12 C, 20 C and 27 C. Liquid/slurry takes the "without natural crust cover" series and the optional 40 percent crust reduction is not applied.
- **2019 sub-zone rows** are the unweighted mean over the sub-zones the Refinement itself groups into each of Cool (Cool Temperate Moist, Cool Temperate Dry, Boreal Moist, Boreal Dry), Temperate (Warm Temperate Moist, Warm Temperate Dry) and Warm (Tropical Montane, Tropical Wet, Tropical Moist, Tropical Dry). Liquid/slurry is read at the 6-month retention time, which is the Refinement's own default where retention time is unknown (Table 10.17 footnote 1).

The 2019 pasture value is half of a pair

The Refinement's single 0.47 percent for pasture, range and paddock is **not** a drop-in replacement for the 2006 triple. Section 10.4.2 of the same chapter (p. 10.66) states that it "must be used in conjunction with a single B0 value of 0.19 m3 CH4 kg-1 of VS excreted", and that this pair "was judged by the expert panel to be more accurate than emission factors estimated from regionally based MCFs and animal category based B0"; footnote 2 of Table 10.17 (Updated) (p. 10.70) repeats that pasture MCFs "must always be used in conjunction with a B0 value of 0.19", and Annex 10B.6 (p. 10.164) derives the pair from grazing measurements on cattle and sheep, finding no significant difference by climate zone or animal category. The pair is carried in `paired_bo_m3_kg_vs`.

Since whep#1137 the Tier 2 kernel honours it by default: the `pasture_bo = "paired"` manure-engine option (see manure_engine_options) prices the pasture stream at 0.19 and every other stream at the animal-category `Bo`, and `pasture_bo = "species"` restores the single per-species `Bo` – the hybrid the Refinement rejects – for reproducibility and sensitivity. `method_manure_ch4` records which.

Measured on FAOSTAT 2020 heads, national grain, the Temperate default and the Global GLEAM 2.0 manure-management split, the pairing moves global Tier 2 manure CH4 by **+0.033 percent** (14.3353 to 14.3401 Tg): goats +1.1 percent (`Bo` 0.18, pasture a fifth of their weighted MCF), buffalo +0.84 percent (`Bo` 0.10, pasture under 1 percent of theirs), cattle -0.07 percent and sheep unchanged (`Bo` 0.19 already). Horses, mules and asses and camels, whose `Bo` is furthest from 0.19, reach no Tier 2 CH4 in that run. The +30.5 percent buffalo figure first quoted in whep#1137 was measured under the placeholder split that whep#958 replaced, which sent 60 percent of buffalo manure to pasture.

What the table cannot hold

The 2019 Refinement resolves the anaerobic digester into **six** leakage-and-storage classes spanning 1.00 to 13.17 percent, so no single `"Anaerobic Digester"` MCF exists to ship. They are carried as six distinct `mms_type` values rather than flattened to one number, and no shipped MMS vocabulary selects one: which class a country's digesters belong to is an inventory input WHEP does not have (whep#1022). The 2006 digester is `NA` for the same reason – that edition publishes a 0 to 100 percent range and requires the compiler to evaluate its Formula 1.

Source

IPCC. 2006. *2006 IPCC Guidelines for National Greenhouse Gas Inventories*, Vol 4, Ch 10, Table 10.17 "MCF values by temperature for manure management systems", pp. 10.44-10.47 (PDF md5 `97b97e8d0e4ca101c77fac2d63a0cb86`).

IPCC. 2019. *2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories*, Vol 4, Ch 10, Table 10.17 (Updated) "Methane conversion factors for manure management systems", pp. 10.68-10.70 (PDF md5 `c1784e747af9bb307e93f4170c12679a`).

Both PDFs re-downloaded from `ipcc-nggip.iges.or.jp` and checksummed on 2026-09-08. Every cell here was read off them; the collapse rules above are the only step that is not a transcription.

Examples

```
climate_mcf_ipcc
```

`coello_synthetic_n` *Coello (2025) crop-specific synthetic nitrogen application rates*

Description

Corrected average synthetic-nitrogen application rates (kg N ha^{-1}) by calendar year, FAOSTAT area and CBS crop item, derived from Coello et al. (2025). Coello's 13 crop groups are crosswalked to FAOSTAT CBS items via `inst/extdata/coello_mapping.csv` and items_prod_full; native years 1961-2019 are carried forward to 2023 with fill_linear. Used by the package synthetic-fertiliser path (`method = "coello"`) to differentiate the FAOSTAT national synthetic-N total across crops while conserving that national total. Rates are the source values clamped to non-negative; implausible outliers above 1000 kg N/ha (Coello model-extrapolation artifacts in a few small areas) are treated as missing, so those crop-area-years follow the missing-rate fallback (temporal fill where available, else the area-share weight) rather than skewing the split. The downstream rate-weighted share normalises within each country-year, so the national total is conserved regardless.

Usage

```
coello_synthetic_n
```

Format

A tibble with one row per year-area-crop:

- `year`: Integer calendar year (1961-2023).
- `area_code`: Integer FAOSTAT area code.
- `item_cbs_code`: Integer CBS crop item code.
- `kg_n_ha`: Synthetic-N application rate (kg N ha^{-1}).

Source

Coello, D. et al. (2025) A global gridded crop-specific fertilization dataset from 1961 to 2019. *Scientific Data* 12:40. doi:10.1038/s4159702404215x

Examples

```
head(coello_synthetic_n)
```

`combine_cov` *Combine independent coefficient-of-variation components.*

Description

Aggregate several independent sources of relative uncertainty into a single coefficient of variation by adding them in quadrature. This is how per-indicator data-quality scores (for example the reliability, completeness and representativeness columns of a pedigree matrix) are turned into one CoV for `propagate_fp_uncertainty()`. Supply the per-indicator CoV values yourself from a sourced, citable pedigree table; no uncertainty factors are hard-coded here.

Usage

```
combine_cov(...)
```

Arguments

...	Numeric vectors of equal length, one per uncertainty source, or a single matrix/data frame with one column per source.

Value

A numeric vector of combined coefficients of variation.

Examples

```
combine_cov(c(0.3, 0.1), c(0.4, 0.2))
```

---

combine_row_evidence *Combine row evidence from several producers.*

Description

Merge evidence tables deterministically. Nothing is overwritten: the result is the union of the claims, with exact duplicates collapsed and the surviving rows in the format's canonical order, so the same inputs always give the same output whatever order they arrive in.

Two producers claiming different values for the same row and field is kept as what it is — two claims — instead of one silently winning by argument position. Find those with `evidence_conflicts()` and decide explicitly.

All inputs must agree on `key_columns`: evidence keyed on `(area_code, year)` and evidence keyed on `(area_code)` address different things, and merging them would change what a `row_key` means. That aborts rather than being reconciled.

Usage

```
combine_row_evidence(...)
```

Arguments

...	Evidence tables from `row_evidence()`, or lists of them. Each is validated against `row_evidence_schema()`. Zero-row tables are allowed and contribute no rows or key constraint.

Value

A tibble of row evidence, in canonical order.

Examples

```
production <- tibble::tibble(area_code = c(724L, 76L), year = 2020L)
key <- c("area_code", "year")
recorded_at <- as.POSIXct("2026-01-01", tz = "UTC")

faostat <- row_evidence(
  production,
  source_id = "FAOSTAT_prod",
  key = key,
  fields = list(method_land = "reported"),
  recorded_at = recorded_at
)
luh2 <- row_evidence(
  production,
  source_id = "LUH2",
  key = key,
  fields = list(method_land = "back-cast"),
  recorded_at = recorded_at
)

combined <- combine_row_evidence(faostat, luh2)
combined

# Both claims survive, and the disagreement is visible.
evidence_conflicts(combined)
```

---

compare_footprint_methods

Compare two footprint estimates.

Description

Align two consumption-based footprint estimates (for example the Leontief and land-balance methods) by consuming country and item and report their difference. Disagreement is diagnostic: it localises where a method's assumptions matter most.

Usage

```
compare_footprint_methods(method_a, method_b)
```

Arguments

`method_a`, `method_b`
: Tibbles with `area_code`, `item_cbs_code` and `value` (consumption footprint).

Value

A tibble with `area_code`, `item_cbs_code`, `value_a`, `value_b`, `abs_diff` and `rel_diff` (relative to the larger of the two), ordered by descending `abs_diff`. A cell missing from one method counts as zero there; a cell present with an `NA` value stays `NA`, so its differences are `NA` too.

Examples

```
a <- tibble::tibble(area_code = 1L, item_cbs_code = 10L, value = 30)
b <- tibble::tibble(area_code = 1L, item_cbs_code = 10L, value = 25)
compare_footprint_methods(a, b)
```

`compute_footprint` *Compute environmental footprints.*

Description

Trace environmental extensions through the supply chain using the Leontief inverse, following the FABIO methodology (Bruckner et al., 2019). The footprint shows how much of an environmental pressure (e.g. land use, water, emissions) is embodied in the final consumption of each product in each country.

The multiplier matrix is computed as $MP_{ij} = (e_i / X_i) \cdot L_{ij}$, where e_i is the extension for sector i. For each demand category, the footprint is decomposed per target item using the FABIO diagonal approach: $FP = MP \cdot \mathrm{diag}(y)$, aggregated by item.

For large systems, pass `z_mat` and `x_vec` instead of `l_inv`. This solves $(I - A)x = Y$ directly using a sparse LU factorisation, avoiding the dense Leontief inverse entirely and reducing memory from $O(n^2)$ to $O(nnz)$.

The `z_mat` and `l_inv` paths give identical results while the largest column sum of A stays below 1 (the Leontief inverse is then non-negative). At or above 1 the paths can diverge: the dense inverse zeroes negative entries whereas the sparse solve drops the resulting negative embodied flows, and a precomputed `l_inv` may have been capped differently (`compute_leontief_inverse()` defaults to a conservative cap below 1). The `z_mat` path emits a warning when this happens; `max_column_sum` is ignored when `l_inv` is supplied directly.

Usage

```
compute_footprint(
  l_inv = NULL,
  x_vec,
  y_mat,
  extensions,
  labels,
  z_mat = NULL,
  fd_labels = NULL,
  output_tol = 1e-08,
```

```
  value_added_floor = 0.001,
  max_column_sum = 100,
  a_denominator = c("traceable", "nonzero"),
  conserve_extensions = TRUE,
  report_conservation = FALSE
)
```

Arguments

`l_inv`	Leontief inverse matrix from `compute_leontief_inverse()`. Ignored when `z_mat` is provided.
`x_vec`	Numeric vector of total output per sector.
`y_mat`	Final demand matrix from `build_io_model()`.
`extensions`	Numeric vector of environmental extensions (e.g. hectares of land use) per sector. Must have the same length as `x_vec`.
`labels`	Tibble with `area_code` and `item_cbs_code` mapping row/column indices to their meaning. From `build_io_model()`.
`z_mat`	Optional inter-industry flow matrix from `build_io_model()`. When provided, the system is solved directly (sparse LU), and `l_inv` is not needed.
`fd_labels`	Optional tibble labelling Y columns. Pass `fd_labels[[i]]` from `build_io_model()` output. When provided, footprints are decomposed per target item using the FABIO diagonal approach, and the result includes a `target_fd` column. When omitted, columns of Y are treated as sectors (appropriate only when Y is square).
`output_tol`	Minimum output considered valid when computing extension intensities. Sectors with `x_vec <= output_tol` get zero intensity to avoid infinite or numerically explosive footprints from zero-output residuals.
`value_added_floor`	Minimum share of each sector's output that is treated as non-intermediate leakage when constructing A from `z_mat` if `max_column_sum` is left at its low-level default. Ignored when a precomputed `l_inv` is supplied without `z_mat`.
`max_column_sum`	Maximum allowed column sum in A when using `z_mat`. Physical biomass systems can require more than one unit of intermediate input per unit of output, so the footprint path defaults to `100` and only clips extreme columns caused by residual inconsistencies or tiny outputs.
`a_denominator`	Which outputs A divides by when using `z_mat`. `"traceable"` (the default) treats a sector with `x_vec <= output_tol` as having no output, exactly as the extension intensities and `check_footprint_conservation()` do, so its A column is zero. `"nonzero"` divides by any non-zero output, which is the behaviour before whep#1110: a residue output such as `1e-12` then inflates its column by up to `1e12`, clipped only by `max_column_sum`. The two differ only for a residue-output sector that still receives intermediate inputs, and a warning of class `whep_residue_output_inputs`

names how many there are under either rule. Ignored when a precomputed `l_inv` is supplied without `z_mat`. The choice is recorded in the `method_a_denominator` column.

`conserve_extensions`
: If `TRUE`, rescale positive footprint flows within each origin area/item so their sum does not exceed the corresponding positive extension total. This keeps footprint outputs conservative when capped coefficients or negative final demand columns would otherwise make positive-only paths larger than the source extension.

`report_conservation`
: If `TRUE`, emit a message after computing the footprint reporting the conservation gap: the signed global discrepancy between direct extension and embodied footprint, via `check_footprint_conservation()` – the share not embodied in final demand when negative, or the share over-traced when positive. Off by default so the gap is opt-in but never silent when requested.

Value

A tibble with footprint results containing:

- `origin_area`: Country where the pressure occurs.
- `origin_polity_code`: WHEP polity for `origin_area`.
- `origin_item`: Item causing the pressure.
- `target_area`: Country consuming the product.
- `target_polity_code`: WHEP polity for `target_area`.
- `target_item`: Item consumed.
- `target_fd`: Demand category (e.g. `"food"`). Only present when `fd_labels` is provided.
- `value`: Footprint value in extension units.
- `method_a_denominator`: The `a_denominator` rule A was built with, or `"precomputed_l_inv"` when a precomputed `l_inv` was used.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
l_inv <- compute_leontief_inverse(z_mat, x_vec)
y_mat <- matrix(c(85, 195), ncol = 1)
extensions <- c(50, 30)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(1L, 2L)
)

# Small system: pass pre-computed L
compute_footprint(l_inv, x_vec, y_mat, extensions, labels)
```

```
# Using Z directly (computes L internally)
compute_footprint(
  x_vec = x_vec, y_mat = y_mat,
  extensions = extensions, labels = labels,
  z_mat = z_mat
)
```

`compute_footprint_balance` *Compute land footprints by physical trade balance.*

Description

Estimate consumption-based land footprints by propagating direct land use through the bilateral physical trade network, an approach independent of the Leontief inverse used by `compute_footprint()`. Each country's supply pool (production plus imports) carries an embodied-land intensity `s`; solving the balance $(D - M)s = L$ per item gives `s`, where `D` is diagonal supply throughput, `M` routes import intensities, and `L` is direct land. The footprint of consumption (supply minus exports) is then `consumption * s`.

Unlike the multi-regional input-output method, this tracer handles only the trade dimension, not inter-product processing transformations. Comparing the two with `compare_footprint_methods()` isolates the effect of that assumption – an apples-to-apples stress test on shared FAOSTAT data.

Usage

```
compute_footprint_balance(production, trade, extension)
```

Arguments

`production`	Tibble with `area_code`, `item_cbs_code` and `value` (quantity produced).
`trade`	Tibble with `from_code`, `to_code`, `item_cbs_code` and `value` (quantity exported from `from_code` to `to_code`).
`extension`	Tibble with `area_code`, `item_cbs_code` and `value` (direct land use of domestic production).

Value

A tibble with `area_code` (consuming country), `item_cbs_code`, `value` (embodied land in consumption) and `method` (`"land_balance"`).

Examples

```
production <- tibble::tibble(
  area_code = c(1L, 2L),
  item_cbs_code = c(10L, 10L),
  value = c(100, 0)
)
```

```
trade <- tibble::tibble(
  from_code = 1L, to_code = 2L, item_cbs_code = 10L, value = 40
)
extension <- tibble::tibble(
  area_code = c(1L, 2L),
  item_cbs_code = c(10L, 10L),
  value = c(50, 0)
)
compute_footprint_balance(production, trade, extension)
```

`compute_footprint_paths` *Compute first-use footprint paths.*

Description

Decompose an origin footprint into the first sector that directly uses the origin product before the footprint reaches final demand. This is useful for Sankey views that show paths such as origin product -> first-use area -> first-use product -> final-demand area.

The decomposition uses the IO identity $x = d + Ax$. For each selected origin sector i and final-demand target, the origin requirement x_i is split into direct final demand d_i and direct intermediate use $A_{ij}x_j$. Values are multiplied by the origin extension intensity e_i/X_i.

Usage

```
compute_footprint_paths(
  z_mat,
  x_vec,
  y_mat,
  extensions,
  labels,
  fd_labels,
  origin_area = NULL,
  origin_item = NULL,
  output_tol = 1e-08,
  value_added_floor = 0.001,
  max_column_sum = 100,
  a_denominator = c("traceable", "nonzero"),
  conserve_extensions = TRUE,
  min_value = 0
)
```

Arguments

`z_mat`	Inter-industry flow matrix from `build_io_model()`.
`x_vec`	Numeric vector of total output per sector.
`y_mat`	Final demand matrix from `build_io_model()`.

`extensions` Numeric vector of environmental extensions per sector.

`labels` Tibble with `area_code` and `item_cbs_code` mapping sectors.

`fd_labels` Tibble labelling Y columns, from `build_io_model()`.

`origin_area` Optional vector limiting origin sectors. Each value is matched against `labels$area_code` – the LEGACY numeric area code – and a value that matches no `area_code` is then resolved through the polity vocabulary of polity_area_crosswalk, so a `polity_area_code` (`206`, the Sudan aggregation bucket) or a `polity_code` (`"SDN-2011-2025"`) selects the area codes it covers. Legacy codes keep their legacy meaning. Values that resolve to no sector are dropped with a warning, and a call in which nothing resolves aborts instead of returning an empty table.

`origin_item` Optional item code vector limiting origin sectors.

`output_tol` Minimum output considered valid when computing extension intensities.

`value_added_floor` Minimum non-intermediate leakage share used when constructing technical coefficients from `z_mat`.

`max_column_sum` Maximum allowed column sum in A. Must match the value used by `compute_footprint()` (default `100`) so the path decomposition and the footprint it decomposes share an identical A cap.

`a_denominator` Which outputs A divides by. Must match the value used by `compute_footprint()` (default `"traceable"`, which treats `x_vec <= output_tol` as no output) so the paths decompose the same A. See `compute_footprint()` for the alternatives.

`conserve_extensions` If `TRUE`, rescale positive paths within each origin area/item so their sum does not exceed the corresponding positive extension total.

`min_value` Drop paths with values less than or equal to this value before returning.

Value

A tibble with `origin_area`, `origin_item`, `use_area`, `use_item`, `target_area`, `target_item`, role-specific polity metadata, `target_fd`, `path_type`, and `value`.

Examples

```
# A two-sector economy: sector 1 (item 10) carries the whole extension and
# supplies 50 of intermediate input to sector 2 (item 20). Its footprint
# therefore splits into a direct final-demand path and a first-use path
# through item 20.
z_mat <- matrix(c(0, 50, 0, 0), nrow = 2, byrow = TRUE)
x_vec <- c(100, 200)
y_mat <- matrix(c(10, 100), nrow = 2)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(10L, 20L)
)
fd_labels <- tibble::tibble(area_code = 2L, fd_col = "food")
```

```
compute_footprint_paths(
  z_mat = z_mat,
  x_vec = x_vec,
  y_mat = y_mat,
  extensions = c(200, 0),
  labels = labels,
  fd_labels = fd_labels,
  conserve_extensions = FALSE
) |>
  dplyr::select(origin_item, use_item, target_area, path_type, value)
```

`compute_fp_product_paths` *Compute final-product footprint paths.*

Description

Decompose an origin footprint by the area and item of the product supplied to final demand. This adds the missing FABIO-viewer style phase between origin product and final-demand area: origin product -> supplied product area -> supplied product -> final-demand area.

Unlike `compute_footprint_paths()`, this does not show the first direct intermediate input. It shows the downstream product row in `Y` whose final demand carries the origin footprint.

Usage

```
compute_fp_product_paths(
  z_mat,
  x_vec,
  y_mat,
  extensions,
  labels,
  fd_labels,
  origin_area = NULL,
  origin_item = NULL,
  output_tol = 1e-08,
  value_added_floor = 0.001,
  max_column_sum = 100,
  a_denominator = c("traceable", "nonzero"),
  conserve_extensions = TRUE,
  min_value = 0
)
```

Arguments

`z_mat`	Inter-industry flow matrix from `build_io_model()`.
`x_vec`	Numeric vector of total output per sector.

`y_mat`	Final demand matrix from `build_io_model()`.
`extensions`	Numeric vector of environmental extensions per sector.
`labels`	Tibble with `area_code` and `item_cbs_code` mapping sectors.
`fd_labels`	Tibble labelling Y columns, from `build_io_model()`.
`origin_area`	Optional vector limiting origin sectors. Each value is matched against `labels$area_code` – the LEGACY numeric area code – and a value that matches no `area_code` is then resolved through the polity vocabulary of polity_area_crosswalk, so a `polity_area_code` (`206`, the Sudan aggregation bucket) or a `polity_code` (`"SDN-2011-2025"`) selects the area codes it covers. Legacy codes keep their legacy meaning. Values that resolve to no sector are dropped with a warning, and a call in which nothing resolves aborts instead of returning an empty table.
`origin_item`	Optional item code vector limiting origin sectors.
`output_tol`	Minimum output considered valid when computing extension intensities.
`value_added_floor`	Minimum non-intermediate leakage share used when constructing technical coefficients from `z_mat`.
`max_column_sum`	Maximum allowed column sum in A. Must match the value used by `compute_footprint()` (default `100`) so the path decomposition and the footprint it decomposes share an identical A cap.
`a_denominator`	Which outputs A divides by. Must match the value used by `compute_footprint()` (default `"traceable"`, which treats `x_vec <= output_tol` as no output) so the paths decompose the same A. See `compute_footprint()` for the alternatives.
`conserve_extensions`	If `TRUE`, rescale positive paths within each origin area/item so their sum does not exceed the corresponding positive extension total.
`min_value`	Drop paths with values less than or equal to this value before returning.

Value

A tibble with `origin_area`, `origin_item`, `product_area`, `product_item`, `target_area`, role-specific polity metadata, `target_fd`, and `value`.

Examples

```
# Same two-sector economy as [compute_footprint_paths()], decomposed by the
# product that reaches final demand instead of by the first intermediate
# use. The path totals of the two functions agree with each other and with
# [compute_footprint()] under a shared `max_column_sum`.
z_mat <- matrix(c(0, 50, 0, 0), nrow = 2, byrow = TRUE)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(10L, 20L)
)
```

```
compute_fp_product_paths(
  z_mat = z_mat,
  x_vec = c(100, 200),
  y_mat = matrix(c(10, 100), nrow = 2),
  extensions = c(200, 0),
  labels = labels,
  fd_labels = tibble::tibble(area_code = 2L, fd_col = "food"),
  conserve_extensions = FALSE
) |>
  dplyr::select(origin_item, product_area, product_item, value)
```

`compute_leontief_inverse`
Compute Leontief inverse.

Description

Compute the Leontief inverse matrix from intermediate flows and total output. The Leontief inverse captures both direct and indirect requirements across the entire supply chain, enabling footprint tracing.

The technical coefficients matrix is computed as $A_{ij} = Z_{ij}/X_j$, representing the input of sector i needed per unit of output from sector j. Column sums of A are capped using `max_column_sum` to avoid singular systems from inconsistent supply-use data. By default this uses `1 - value_added_floor`, preserving the previous conservative behavior for explicit Leontief inverses. The Leontief inverse is then $L = (I - A)^{-1}$.

For large systems (thousands of sectors) this function is not usable: the dense L matrix requires $n^2 \times 8$ bytes of memory (e.g. ~4.8 GiB for n = 25 000). Use `compute_footprint()` directly with `z_mat` and `x_vec` instead, which solves $(I-A)x = Y$ without ever materialising L.

Accepts both dense and sparse (Matrix package) inputs.

Usage

```
compute_leontief_inverse(
  z_mat,
  x_vec,
  max_n = 5000,
  value_added_floor = 0.001,
  max_column_sum = 1 - value_added_floor
)
```

Arguments

`z_mat`	Square numeric matrix of inter-industry flows. Entry Z_{ij} is the flow from sector i to sector j. Can be dense or sparse.
`x_vec`	Numeric vector of total output per sector. Must have the same length as `nrow(z_mat)`.

`max_n` Maximum system size before aborting. Defaults to 5000. Set higher at your own risk of memory exhaustion.

`value_added_floor` Minimum share of each sector's output that is treated as non-intermediate leakage when constructing A if `max_column_sum` is left at its default.

`max_column_sum` Maximum allowed column sum in A. Columns above this value are rescaled. Defaults to `1 - value_added_floor`.

Value

The Leontief inverse matrix L. Negative values are set to zero. Returns a dense matrix.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
compute_leontief_inverse(z_mat, x_vec)
```

---

compute_trade_flows_raw

Compute national trade flows: model vs. raw historical FAO series

---

Description

Computes gross export and import flows (in MgN) by item and year, both from the national GRAFS model (production and consumption aggregated to Spain before splitting into export/import, avoiding the double-counting of inter-provincial trade that a province-level sum would introduce) and from the raw historical Export/Import series for Spain in `Europe_FAO_completed.xlsx` (1849-1960), restricted to the item/year combinations reported in the raw sheets. Items are classified as `"Crop"` or `"Livestock"` for downstream aggregation, e.g. in `plot_national_trade_flows_raw()`.

Usage

```
compute_trade_flows_raw(n_nat_destiny = NULL, example = FALSE)
```

Arguments

`n_nat_destiny` Optional pre-computed output from `create_n_nat_destiny()`. If `NULL`, calls that function internally.

`example` If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble with columns `year`, `item`, `category` (`"Crop"` or `"Livestock"`), `source` (`"WHEP model"` or `"FAO (raw)"`), `flow` (`"Export"` or `"Import"`), and `value_n` (MgN).

Examples

```
compute_trade_flows_raw(example = TRUE)
```

`consolidate_sources` *Consolidate a multi-source panel to one winning row per cell.*

Description

Reduce a long panel in which several sources report the same (`.by`, `time_col`) cell to a single winning row per cell, chosen by an explicit source-priority ranking with measure-aware demotion, a coverage tie-break, an optional quality tie-break, and a continuity override. It is the general form of the priority-based deduplication used to build the long-term historical energy panel.

Usage

```
consolidate_sources(
  data,
  value_col,
  source_col,
  priority,
  .by = NULL,
  time_col = year,
  drop_at = 100L,
  measure = NULL,
  tie_break = NULL,
  continuity_override = TRUE,
  verbose = TRUE,
  priority_scope = c("specific", "source")
)
```

Arguments

`data`	A tibble with one row per source per (`.by`, `time_col`) cell.
`value_col`	Unquoted name of the value column. Coverage counts the cells where this column is non-missing, or only those where it is strictly positive when `tie_break$coverage` is `"positive"`.
`source_col`	Unquoted name of the source-label column.
`priority`	Source-to-rank map, as a named integer vector (`c(OWID = 1L, Malanima = 4L)`), a two-column data frame (source, rank), or a scoped table: a data frame with the source column named as `source_col`, a `rank` column, and any number of further columns present in `data` that scope the entry (`NA` matches any value). Lower rank wins. A row takes the rank of the most specific entry it matches; sources absent here take the fallback rank `drop_at - 1L`. A data frame with more than two columns must name them: which column carries the rank cannot be guessed, and guessing wrong would publish another source's number.

`.by` Character vector of grouping columns that, with `time_col`, key a cell (for example `c("region", "category")`). `NULL` (default) keys cells by `time_col` alone.

`time_col` Unquoted name of the time column. Default: `year`. Must be numeric; the continuity override treats a difference of one as adjacent.

`drop_at` Integer rank at or above which a source is dropped before consolidation. Default: `100L`.

`measure` Optional named list of measure-demotion options:

- `basis`: data frame flagging measure-mismatched rows. It must contain the source column and may add further key columns present in `data` (for example a category column) to scope the flag; a data row is flagged when it matches any `basis` row on all its columns. Default: `NULL` (no demotion).
- `penalty`: integer added to the effective rank of a flagged, non-exempt row. Default: `1000L` (larger than any sensible base rank, so a flagged source falls below every unflagged one while flagged sources keep their relative order).
- `exempt`: one-sided formula selecting rows the penalty never applies to, such as `~ region == "WLD"`, evaluated on the rows that survive the hard drop. Default: `NULL`.

`tie_break` Optional named list of options breaking equal-rank ties:

- `coverage`: break ties by broader within-series coverage. `TRUE` (default), or equivalently `"nonmissing"`, counts the cells where `value_col` is non-missing; `"positive"` counts only the cells where it is strictly positive (`value_col` must then be numeric); `FALSE` disables the coverage tie-break.
- `coverage_by`: character vector of `.by` columns at which coverage is counted, overriding the default of the full `.by` group. It must be a subset of `.by` – coverage may be counted at a coarser grain than the cell, never at a different one – and `character(0)` counts a source's coverage across the whole panel. Counting coverage one level coarser than the cell lets a source with broad category-level coverage win a tie in a subcategory where it is thin. Default: `NULL` (the `.by` grain).
- `quality_col`: string naming a quality column used as a tie-break after coverage. Default: `NULL`.
- `quality_levels`: character vector ordering `quality_col` values best first (unlisted values rank last). Required when `quality_col` is set.
- `quality_variants`: logical. When `TRUE`, a source contributing several `quality_col` variants of one cell keeps its best-ranked variant instead of aborting; rows sharing source, cell and quality level still abort, as do variants whose best rank is not unique. Requires `quality_col`. Default: `FALSE`.

`continuity_override` Revert isolated single-period winner flips. `TRUE` (default) or `FALSE`, or a named list of options, which also turns the override on:

- `adjacency`: one of `"step"` (default) or `"within"`. `"step"` flags a flip only when both flanking periods sit exactly one time step away; `"within"` accepts at most one step, flagging flips in a series whose spacing is finer or irregular.
- `exempt`: one-sided formula selecting winning rows the isolation flag never applies to, such as `~ source == "Smil_2017"`, evaluated on the winners. Use it for a source whose observations are deliberately sparse, which would otherwise lose every anchor to the override. Default: `NULL`.

`verbose` Logical. Report the drop count, how many rows took a scope-specific priority rank, any resolved quality variants, name-order ties, and continuity reversions. Default: `TRUE`.

`priority_scope`
One of `"specific"` (default) or `"source"`, selecting how a scoped `priority` table is read. `"specific"` honours the scope keys, so a source can outrank its usual tier in one category only. `"source"` drops every scoped entry and ranks by source alone, reproducing what the table expressed before scoping existed; it is the sensitivity run that says what the scope is worth. No effect on an unscoped `priority`.

Details

Selection proceeds in four stages.

1. **Hard drop.** Every row whose source ranks at or above `drop_at` is removed before any cell is contested, so a pinned source can never win even an uncontested cell. Sources absent from `priority` receive the documented fallback rank `drop_at - 1L`: kept in play but ranked below every source listed with a smaller rank. To exclude an unreliable source, list it at `drop_at` or above.

`priority` may also be **scoped**. A table keyed on the source plus further columns of `data` (a category, say) pins one source's rank inside one category while leaving its ordinary rank everywhere else, which listing that source at a different global rank cannot do: that would move the outcome in every other category too. A row takes the rank of the most specific entry it matches, an `NA` scope key meaning "any value", and two equally specific entries that disagree abort rather than let table order decide. `priority_scope = "source"` reads the same table with its scoped entries ignored, which is exactly what it bought before scoping existed and is therefore the comparison that quantifies what the scope changes.

1. **Measure-aware demotion.** A source can report a different measure than the panel's target concept (production where the panel means consumption, generation shares where it means primary energy, a sector fragment where it means a category total). Rows flagged by `measure$basis` receive `measure$penalty` added to their effective rank, so a measure-mismatched source loses any cell a measure-consistent source also reports, yet still wins a cell it alone reports (a lone reporter is never demoted away). Rows matching `measure$exempt` keep their base rank (for example world-level cells, where production equals consumption).
2. **Winner selection.** Within each (`.by`, `time_col`) cell any row with a real (non-missing) value outranks every `value_col`-missing row, so a higher-priority source's `NA` never

discards a lower-priority source's real observation; a cell wins `NA` only when no source reports a real value. Among rows with a real value the winner is the row of lowest effective rank; ties are broken by broader within-series coverage (the count of cells the source reports across the `.by` group, or across the coarser group `tie_break$coverage_by` names) when `tie_break$coverage`, then by `tie_break$quality_col` ordered per `tie_break$quality_levels`, then by ascending source name (reported when `verbose`). Coverage counts the cells where `value_col` is non-missing, or only the strictly positive ones under `tie_break$coverage = "positive"`, for panels where an exact zero reads as "not reported" as often as "measured zero" and would otherwise inflate the coverage of a mostly-zero series.

3. **Continuity override.** When enabled, an isolated single-period winner flip is reverted: if the immediately preceding and following periods share a different winner that also reports the middle period, that continuous source reclaims the middle cell, removing single-period teeth from otherwise smooth series. The reversion is skipped when the flanking source's middle-period value is itself missing (continuity never reinstates an `NA`) and when it would hand a cell won by a measure-consistent source back to a measure-demoted one: continuity never undoes the measure penalty, because a single-period source switch is cosmetic while a measure switch corrupts the series.

A source whose observations are deliberately sparse – a milestone grid meant to be interpolated between – looks like a run of isolated flips inside another source's annual run, and the override strips every one of its anchors, collapsing the backbone to a lower-priority partial series. `continuity_override$exempt` selects the rows the isolation flag never applies to. `continuity_override$adjacency` states what counts as a flanking period: `"step"` (the default) requires both neighbours to sit exactly one time step away, so only a true single-period tooth is reverted; `"within"` accepts at most one step, which also reverts a flip flanked at a finer-than-unit spacing in an irregular series.

This operationalises the AFE decision *Consolidate multi-source panels measure-consistently* (`wiki/decisions/measure-consistent-panel-consolidation`): measure identity is part of the dedup key's semantics, and priority alone cannot arbitrate cells whose sources report different measures.

The input must hold at most one row per source per cell; pre-aggregate any sub-detail rows first (by default the function aborts on duplicates rather than sum silently). Set `tie_break$quality_variants` when a source legitimately contributes several `tie_break$quality_col` variants of one cell (an observed and an interpolated estimate, say): the variants then collapse to the best-ranked one before any other stage, and only rows sharing a source, a cell *and* a quality level still abort.

Value

A tibble with the winning row per (`.by`, `time_col`) cell, the original columns of `data`, and five added provenance columns: `n_sources` (distinct sources contesting the cell after the hard drop), `source_rank` (the winner's base priority rank), `effective_rank` (base rank plus any measure penalty applied), `measure_demoted` (whether the winner carried the measure penalty; a flagged source only wins a cell that no measure-consistent source reports), and `method_source`, naming the stage that decided the cell: `"sole_source"` (no rival contested it), `"nonmissing"` (the rival reported no value), `"priority"` or `"priority_scoped"` (a lower effective rank, from a source-keyed or a scope-keyed `priority` entry), `"coverage"`,

`"quality"`, `"name_order"` (ascending source name settled a full tie), or `"continuity"` (the continuity override handed the cell back). Rows are ordered by `.by` then `time_col`.

Examples

```
panel <- tibble::tribble(
  ~year, ~region, ~category, ~source, ~value,
  1900, "WLD", "Coal", "OWID", 10,
  1900, "WLD", "Coal", "Malanima", 20,
  1901, "WLD", "Coal", "Malanima", 21,
  1902, "WLD", "Coal", "Malanima", 22
)

consolidate_sources(
  panel,
  value_col = value,
  source_col = source,
  priority = c(OWID = 1L, Malanima = 4L),
  .by = c("region", "category"),
  verbose = FALSE
)

# A scoped table pins Malanima above OWID for Coal alone: its rank in every
# other category stays 4, which inflating its global rank could not do.
scoped <- tibble::tribble(
  ~source, ~category, ~rank,
  "OWID", NA_character_, 1L,
  "Malanima", NA_character_, 4L,
  "Malanima", "Coal", 0L
)

consolidate_sources(
  panel,
  value_col = value,
  source_col = source,
  priority = scoped,
  .by = c("region", "category"),
  verbose = FALSE
)
```

---

conv_bouwman *Bouwman feed conversion ratios.*

Description

Feed conversion (kilograms dry matter feed per kilogram product) and per feed type composition for the main production species, from Bouwman et al. (2005). Migrated from the afsetools `Codes_coefs.xlsx` workbook.

Usage

```
conv_bouwman
```

Format

A tibble with one row per species, feed type, region and anchor year:

item_bouwman Bouwman species label (Beef cattle, Dairy cattle, Pigs, Poultry, Sheep and goats).

feed_type Feed type: animals, crops, grass, residues or scavenging.

year Anchor year (1970, 1995 or 2030).

region_bouwman Bouwman seventeen region label.

conversion Per feed type conversion factor (kg DM feed per kg product). Sum across feed types is the total feed conversion ratio; normalised across feed types it gives the default feed composition shares.

Source

Bouwman et al. (2005), via afsetools `Codes_coefs.xlsx`.

`conv_krausmann` *Krausmann per head feed intake.*

Description

Annual feed intake (tonnes dry matter per head per year) for draft and non productive species that lack a product based feed conversion ratio, from Krausmann et al. (2013). Migrated from the afsetools `Codes_coefs.xlsx` workbook.

Usage

```
conv_krausmann
```

Format

A tibble with one row per species:

item_cbs_code FAOSTAT commodity balance item code.

species Species name (Horses, Asses, Mules, Camels and so on).

conversion Feed intake (t DM per head per year).

Details

The unit is tonnes, not kilograms: the feed demand builder multiplies FAOSTAT head counts directly by `conversion` and books the product in the same column as the product based path (tonnes of product times a dimensionless feed conversion ratio), which downstream is named `demand_dm_t`. A horse at 3.65 t DM per year eats 10 kg DM a day; read as kilograms it would be 10 g. The workbook's `original_value` column (kept in `inst/extdata/feed/conv_krausmann.csv`) reads the same way: its values are round daily intakes of 10, 6, 5 and 0.1 kg DM times 365 / 1000. Which table of Krausmann et al. (2013) the values come from, and why seven of the eight `conversion` values (all but horses) differ from `original_value`, is unverified.

Source

Krausmann et al. (2013), via afsetools `Codes_coefs.xlsx`.

`correct_n_deposition` *Correct HaNi's European deposition trend toward EMEP.*

Description

Rescales one HaNi species, country by country and year by year, so that its trajectory inside the EMEP core countries follows EMEP's while its level over a reference period stays HaNi's own. This is the year-resolved correction whep#1121 asks for and is **opt-in**: nothing in the package calls it, and `build_n_deposition()` keeps reading plain HaNi unless a corrected field is injected through its `data` argument.

Usage

```
correct_n_deposition(
  hani,
  emep,
  cell_polity,
  method = c("emep_trend", "none"),
  reference_years = 2010:2019,
  area_codes = NULL
)
```

Arguments

`hani`	One HaNi species as returned by `read_n_deposition()`: `lon`, `lat`, `year`, `value_g` and `method_deposition`, which must be `"hani"` on every row so that a field cannot be corrected twice. It must cover `reference_years`.
`emep`	The same species from `read_emep_deposition()`: `lon`, `lat`, `year` and `deposition_kgn_ha`. It must cover `reference_years`.
`cell_polity`	The cell-polity table, as `build_cell_polity()` returns it: `lon`, `lat`, `area_code`, `polity_frac` and `cell_area_ha`.

`method` `"emep_trend"` (default) applies the correction; `"none"` returns `hani` unchanged, so a pipeline can switch it off without a second code path.

`reference_years` Integer years over which the corrected mass keeps HaNi's mean level. Defaults to `2010:2019`.

`area_codes` Integer `area_code`s of the countries to correct. `NULL` (default) takes the 39 EMEP core countries, the European countries the EMEP model is built to represent. Its domain also reaches Central Asia and the Arabian peninsula, where the two products disagree by factors of 2-5 in both directions, which is a domain-edge artifact rather than a HaNi bias, so those countries are not corrected by default.

Value

`hani` with `value_g` rescaled on corrected rows, those rows stamped `method_deposition = "hani_emep_trend"`, and a `deposition_correction` column holding the factor applied (`1` on every untouched row).

What is wrong with HaNi over Europe

Over cells at least 98% land in the 39 EMEP core countries, HaNi falls from 10.6 to 8.6 kg N/ha/yr between 1990 and 2019 (-19%) while EMEP falls from 14.6 to 8.3 (-43%), so the HaNi/EMEP ratio climbs from 0.73 to 1.04 (`validation/n_deposition_emep.R`). HaNi is too flat over exactly the period in which European emission controls halved deposition.

The correction

For country c and year y the factor is

$$f_{c,y} = \frac{E_{c,y}/\bar{E}_c}{H_{c,y}/\bar{H}_c}$$

where H and E are the HaNi and EMEP masses summed over the country's cells and the bars are their means over `reference_years`. Every HaNi cell of that country-year is multiplied by $f_{c,y}$. Three properties follow, and the tests pin each:

- **Year-resolved.** Every year has its own factor, so the three years 2011-2013 in which HaNi sits above EMEP are lowered while 1990 is raised. A single scale factor anchored on one year cannot do both.
- **Trend, not level.** The mean of the corrected mass over `reference_years` equals HaNi's, and HaNi's spatial pattern inside a country is kept. EMEP's level is never imposed: HaNi books mass deposited to land while EMEP is a density over the whole cell, so their levels differ by the land fraction in every coastal cell, and that difference cancels in a ratio of ratios but not in a ratio.
- **Bounded in time and space.** Only country-years in the EMEP core list that have both products are touched; every other row is returned as read and keeps `method_deposition = "hani"`. In particular **nothing before 1990 is corrected** (EMEP rv5.6 starts in 1990), so a corrected series steps at 1989-1990 by that year's factor – about +40% over the core countries for the two species together. How to carry the correction back in time is the open part of whep#1121 and is not decided here.

`reference_years` defaults to 2010-2019, the last decade the two products overlap, in which their core-country ratio stays within 0.89-1.03. It is a choice, not a sourced value (assumed, unverified): it says which period's HaNi level is trusted. A decade rather than one year, because the ratio is not monotone and a single anchor year would carry its own noise into every other year.

Each border cell is assigned to the country holding its largest `polity_frac`, so it is counted once when the factors are formed and gets one factor.

Examples

```
correct_n_deposition(
  hani = read_n_deposition(example = TRUE),
  emep = read_emep_deposition(example = TRUE),
  cell_polity = tibble::tribble(
    ~lon, ~lat, ~area_code, ~polity_frac, ~cell_area_ha,
    -0.25, -0.25, 79L, 1, 300000
  ),
  reference_years = 2020L
)
```

---

create_alfredos_typologies

Alfredo's typology classification

Description

Calculates typologies for provinces based on grassland, fertilizer, imported feed, and woody/herbaceous shares.

Usage

```
create_alfredos_typologies(
  soil_inputs = NULL,
  prod_destiny = NULL,
  years = 1860:2020
)
```

Arguments

`soil_inputs`	A data frame containing soil nitrogen inputs.
`prod_destiny`	A data frame containing production and destiny data.
`years`	Years between 1860 and 2020.

Value

A data frame with the columns: Year, Province_name, grass_N, fertilizer_N, feed_import_N, woody, herbaceous, woody_share, and Category.

Examples

```
# Minimal stand-ins for `create_n_prov_destiny()` and
# `create_n_soil_inputs()`, carrying only the columns the classification
# reads. One province of each of three categories: Lugo is dominated by
# grassland N, Sevilla by synthetic N on herbaceous production, and Huelva
# by synthetic N where the local production is woody (acorns).
prod_destiny <- tibble::tribble(
  ~year, ~province_name, ~box, ~item, ~origin, ~destiny, ~mg_n,
  2000, "Lugo", "semi_natural_agroecosystems", "Grassland",
  "semi_natural_agroecosystems", "livestock_rum", 900,
  2000, "Sevilla", "semi_natural_agroecosystems", "Grassland",
  "semi_natural_agroecosystems", "livestock_rum", 100,
  2000, "Huelva", "semi_natural_agroecosystems", "Grassland",
  "semi_natural_agroecosystems", "livestock_rum", 30,
  2000, "Huelva", "semi_natural_agroecosystems", "Acorns",
  "semi_natural_agroecosystems", "export", 200,
  2000, "Lugo", "Cropland", "Maize and products",
  "Outside", "livestock_mono", 20,
  2000, "Sevilla", "Cropland", "Maize and products",
  "Outside", "livestock_mono", 50,
  2000, "Huelva", "Cropland", "Maize and products",
  "Outside", "livestock_mono", 10
)

soil_inputs <- tibble::tribble(
  ~year, ~province_name, ~synthetic,
  2000, "Lugo", 10,
  2000, "Sevilla", 400,
  2000, "Huelva", 150
)

create_alfredos_typologies(
  soil_inputs = soil_inputs,
  prod_destiny = prod_destiny,
  years = 2000
)
```

`create_finn_indicator` *Calculate Finn Cycling Index for each province and year*

Description

Calculates the Finn Cycling Index (FCI) for the GRAFS nitrogen flow network following Finn (1976) and Allesina & Ulanowicz (2004). The index measures the fraction of total system throughput that is cycled through internal compartments (Cropland, semi-natural agroecosystems, Livestock, and People).

Usage

```
create_finn_indicator(n_prov_destiny = NULL, example = FALSE)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble with columns `year`, `province_name`, and `finn_index`.

Examples

```
create_finn_indicator(example = TRUE)
```

`create_grafs_plot_df` *Create GRAFS plot dataset.*

Description

Combines land input data and N flows from crops, livestock, imports, and exports to generate a dataset of nitrogen (MgN) by province and year, ready to feed the GRAFS plot package offered by Alfredo Rodriguez.

Usage

```
create_grafs_plot_df(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble with columns `province`, `year`, `label`, `data`, `align`, and `arrowColor`. Each `label` is a GRAFS template token (e.g. `{ARAiN}`) and `data` holds its value as a character string.

Examples

```
create_grafs_plot_df(example = TRUE)
```

`create_n_nat_destiny` *GRAFS Nitrogen (N) flows – National Spain*

Description

Provides N flows of the Spanish agro-food system on a national level between 1860 and 2020. This dataset is the national equivalent of the provincial GRAFS model and represents Spain as a single system without internal trade between provinces. All production, consumption and soil inputs are aggregated nationally before calculating trade with the outside.

Usage

```
create_n_nat_destiny(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A final tibble containing national N flow data by origin and destiny. It includes the following columns:

- `year`: The year in which the recorded event occurred.
- `item`: The item which was produced, defined in `names_biomass_cb`.
- `irrig_cat`: Irrigation form (irrigated or rainfed)
- `box`: One of the GRAFS model systems: cropland, Semi-natural agroecosystems, Livestock, Fish, or Agro-industry.
- `origin`: The origin category of N: Cropland, Semi-natural agroecosystems, Livestock (manure), People (waste water), Deposition, Fixation, Synthetic, or Outside. Fish and Agro-industry are `box` values, not origins; their nitrogen enters as Outside.
- `destiny`: The destiny category of N: population_food, population_other_uses, livestock_mono, livestock_rum (feed), export, processing_losses (N not credited to a processed output), Cropland and semi_natural_agroecosystems (for N soil inputs).
- `mg_n`: Nitrogen amount in megagrams (Mg).
- `province_name`: Set to "Spain" for all national-level rows.

Examples

```
create_n_nat_destiny(example = TRUE)
```

`create_n_production` *N production for Spain*

Description

Calculates N production at the provincial level in Spain. Production is derived from consumption, export, import, and other uses.

Usage

```
create_n_production(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing:

- `year`: Year
- `province_name`: Spanish province
- `item`: Product item
- `box`: Ecosystem box
- `prod`: Produced N (Mg)

Examples

```
create_n_production(example = TRUE)
```

`create_n_prov_destiny`
GRAFS Nitrogen (N) flows

Description

Provides N flows of the Spanish agro-food system on a provincial level between 1860 and 2020. This dataset is the base of the GRAFS model and contains data in megagrams of N (MgN) for each year, province, item, origin and destiny. Thereby, the origin column represents where N comes from, which includes N soil inputs, imports and production. The destiny column shows where N goes to, which includes export, population food, population other uses and feed or cropland (in case of N soil inputs). Processed items, residues, woody crops, grazed weeds are taken into account.

Usage

```
create_n_prov_destiny(example = FALSE, wood_n_method = c("tissue", "single"))
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.
`wood_n_method`	How the nitrogen concentration of the two wood items is chosen. Both harvested `Wood` and the forest and shrubland residue booked as `Firewood` map to the single `Average wood` row of biomass_coefs, although one is stemwood and the other branches and bark. One of: • `"tissue"` (default): price each by its own tissue, using the pooled medians of Thurner et al. (2025), 0.0010 kg N per kg dry matter for stem sapwood (`Wood`) and 0.0035 for branches (`Firewood`). • `"single"`: use the shared `Average wood` cell for both, 0.0030, which is a branch concentration and carries `Wood` about 3 times too high. Thurner, M., Yu, K., Manzoni, S., Prokushkin, A., Thurner, M. A., Wang, Z., and Hickler, T. (2025). Nitrogen concentrations in boreal and temperate tree tissues vary with tree age/size, growth rate, and climate. *Biogeosciences* 22(5), 1475-1493. doi:10.5194/bg2214752025.

Value

A final tibble containing N flow data by origin and destiny. It includes the following columns:

- `year`: The year in which the recorded event occurred.
- `province_name`: The Spanish province where the data is from.
- `item`: The item which was produced, defined in `names_biomass_cb`.
- `irrig_cat`: Irrigation form (irrigated or rainfed)
- `box`: One of the GRAFS model systems: cropland, Semi-natural agroecosystems, Livestock, Fish, or Agro-industry.
- `origin`: The origin category of N: Cropland, Semi-natural agroecosystems, Livestock (manure), People (waste water), Deposition, Fixation, Synthetic, or Outside. Fish and Agro-industry are `box` values, not origins; their nitrogen enters as Outside.
- `destiny`: The destiny category of N: population_food, population_other_uses, livestock_mono, livestock_rum (feed), export, processing_losses (N not credited to a processed output), Cropland and semi_natural_agroecosystems (for N soil inputs).
- `mg_n`: Nitrogen amount in megagrams (Mg).

Examples

```
create_n_prov_destiny(example = TRUE)
```

`create_n_soil_inputs` *Nitrogen (N) soil inputs for Spain*

Description

Calculates total nitrogen inputs to soils in Spain at the provincial level. This includes contributions from:

- Atmospheric deposition (`deposition`)
- Biological nitrogen fixation (`fixation`)
- Synthetic fertilizers (`synthetic`)
- Manure (excreta, solid, liquid) (`manure`)
- Urban sources (`urban`)

Special land use categories and items are aggregated:

- Semi-natural agroecosystems (e.g., Dehesa, Pasture_Shrubland)
- Firewood biomass (e.g., Conifers, Holm oak)

Usage

```
create_n_soil_inputs(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

A tibble containing:

- `year`: Year
- `province_name`: Spanish province
- `item`: Crop, land use, or biomass item
- `irrig_cat`: Irrigation form (irrigated or rainfed)
- `box`: Land use or ecosystem box for aggregation
- `deposition`: N input from atmospheric deposition (Mg)
- `fixation`: N input from biological N fixation (Mg)
- `synthetic`: N input from synthetic fertilizers (Mg)
- `manure`: N input from livestock manure (Mg)
- `urban`: N input from urban sources (Mg)

Examples

```
create_n_soil_inputs(example = TRUE)
```

`create_typologies_grafs_spain` *Typologies of Julia*

Description

Typologies of provinces in Spain based on nitrogen (N) production data of crops and livestock, using various input datasets and generating classification maps and data frames.

Usage

```
create_typologies_grafs_spain(
  make_map = TRUE,
  shapefile_path = NULL,
  map_year = 1980,
  example = FALSE
)
```

Arguments

`make_map`	If TRUE a map of the typologies will be created.
`shapefile_path`	Optional path to a Natural Earth 10m admin-1 states/provinces shapefile. When `NULL` (default) the layer is downloaded from https://www.naturalearthdata.com on first use and cached locally; set `options(whep.provinces_shapefile = )` to point at an existing copy instead.
`map_year`	The year for which the typology map is created.
`example`	If `TRUE`, return a small example output without reading the remote inputs or the Natural Earth layer. Default is `FALSE`.

Value

A tibble with the classification of Spanish provinces into typologies for `map_year`. It contains the following columns:

- `Province_name`: The name of the Spanish province.
- `Typologie`: Assigned typology category for each province. This is based on thresholds in livestock density (livestock units per hectare of agricultural area), crop N productivity (kg N harvested per hectare of cropland), and the semi-natural and imported shares of feed. The typologies are:
  - `Specialized cropping system`
  - `Extensive cropping system`
  - `Extensive mixed crop-livestock system`
  - `Intensive mixed crop-livestock system`
  - `Specialized livestock-farming system`

Examples

```
create_typologies_grafs_spain(example = TRUE)
```

`create_typologies_of_josette` *Typologies of Josette*

Description

Typologies of provinces in Spain based on nitrogen (N) production data of crops and livestock, considering multiple data inputs and producing classification maps and data frames.

Usage

```
create_typologies_of_josette(
  make_map = TRUE,
  shapefile_path = NULL,
  map_year = 1980,
  example = FALSE
)
```

Arguments

`make_map`	If TRUE a map of the typologies will be created.
`shapefile_path`	Optional path to a Natural Earth 10m admin-1 states/provinces shapefile. When `NULL` (default) the layer is downloaded from https://www.naturalearthdata.com on first use and cached locally; set `options(whep.provinces_shapefile = )` to point at an existing copy instead.
`map_year`	The year for which the typology map is created.
`example`	If `TRUE`, return a small example output without reading the remote inputs or the Natural Earth layer. Default is `FALSE`. The example output carries the three data elements only, not the `df_inputs_plots` plot, so that it needs no plotting package.

Value

A named list with `typologies_df` (the typology classification per year and province), `n_input_df` (the N soil inputs the classification read), `imported_feed_share_df` (the imported-feed share per year and province) and `df_inputs_plots` (a `ggplot` of N inputs by typology).

Examples

```
create_typologies_of_josette(example = TRUE)$typologies_df
```

`create_typologies_whep` *Create WHEP typologies for Spain*

Description

Calculates all decision variables for the WHEP typology using only production and consumption data (no import/export).

Usage

```
create_typologies_whep(
  prod_destiny = create_n_prov_destiny(),
  prod_n = dplyr::rename(create_n_production(), production_n = prod),
  years = 2020
)
```

Arguments

`prod_destiny`	Tibble with N flows from `create_n_prov_destiny()`.
`prod_n`	Tibble with per-item N production from `create_n_production()` (column `prod`, renamed here to `production_n`).
`years`	Numeric vector of years to include (default = 2020).

Value

A data frame with Year, Province_name, decision variables, and Category.

Examples

```
# Minimal stand-ins for the two real inputs, carrying only the columns the
# decision variables read. Lugo feeds its livestock mostly on local grass;
# Barcelona feeds them mostly on imported N.
prod_destiny <- tibble::tribble(
  ~year, ~province_name, ~box, ~item, ~origin, ~destiny, ~mg_n,
  2020, "Lugo", "Cropland", "Wheat and products",
  "Cropland", "population_food", 100,
  2020, "Lugo", "semi_natural_agroecosystems", "Grassland",
  "semi_natural_agroecosystems", "livestock_rum", 800,
  2020, "Lugo", "Cropland", "Maize and products",
  "Cropland", "livestock_rum", 200,
  2020, "Lugo", "Cropland", "Soyabean cake",
  "Outside", "livestock_mono", 50,
  2020, "Barcelona", "Cropland", "Wheat and products",
  "Cropland", "population_food", 100,
  2020, "Barcelona", "Cropland", "Soyabean cake",
  "Outside", "livestock_mono", 900,
  2020, "Barcelona", "Cropland", "Maize and products",
  "Cropland", "livestock_mono", 100
```

```
)

prod_n <- tibble::tribble(
  ~year, ~province_name, ~box, ~production_n,
  2020, "Lugo", "Cropland", 300,
  2020, "Barcelona", "Cropland", 200
)

create_typologies_whep(
  prod_destiny = prod_destiny,
  prod_n = prod_n,
  years = 2020
) |>
  dplyr::select(year, province_name, human_share, import_share, Category)
```

`crops_eurostat` *Eurostat crop classification codes*

Description

Maps Eurostat crop codes to their full crop category names, used when integrating Eurostat agricultural statistics.

Usage

```
crops_eurostat
```

Format

A tibble where each row corresponds to one Eurostat crop category. It contains the following columns:

- `Crop`: Eurostat crop code (e.g., `"G0000"`, `"G1000"`).
- `Name_Eurostat`: Full name of the crop category as used in Eurostat (e.g., `"Plants harvested green from arable land"`, `"Temporary grasses and grazings"`).

Source

Eurostat Agricultural Statistics.

Examples

```
head(crops_eurostat)
```

crops_manure_n *Manure nitrogen application by crop and country*

Description

Country- and crop-level estimates of manure nitrogen applied to cropland, from West et al. (2014). Used as a reference for spatializing manure N inputs in the WHEP pipeline.

Usage

```
crops_manure_n
```

Format

A tibble with one row per crop-country combination containing:

- `Crop_name`: Crop name (character).
- `ISO`: ISO 3166-1 alpha-3 country code.
- `Continent`: Three-letter continent code (e.g. `"AFR"`, `"ASI"`).
- `Manure_N_Mg`: Manure nitrogen applied in megagrams (Mg).

Source

West, P. C. et al. (2014). Leverage points for improving global food security and the environment. *Science*, 345(6194), 325–328. doi:10.1126/science.1246067

Examples

```
head(crops_manure_n)
```

decompose_crop_livestock_conn *Compute the crop-livestock connectivity index per province*

Description

Computes, per province and year, two indicators of local crop-livestock integration described in the decomposition proposal (section 7c) as the "specialization of greatest interest" — regional crop-livestock disconnection:

- **Local feed self-sufficiency**: the share of feed consumed by livestock in a province that was itself grown in that same province (rather than sourced from anywhere else, whether another Spanish province or abroad — `n_prov_destiny` does not distinguish inter-provincial trade from international imports, so both count as "not self-sufficient" here).

- **Manure-recycling ratio**: the share of a province's total cropland and semi-natural N inputs that comes from its own livestock manure, rather than synthetic fertilizer, deposition, fixation, or urban waste.

A well-connected (mixed) province has high values on both; a disconnected (specialized crop-only or livestock-only) province has low values on both, since its livestock has nowhere local to send manure and/or its cropland has no local manure to draw on.

Usage

```
decompose_crop_livestock_conn(n_prov_destiny = NULL, example = FALSE)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A named list with tibbles `by_province` (columns `year`, `province_name`, `self_sufficiency`, `recycling_ratio`) and `national` (the unweighted across-province average of both indicators, by `year`).

Examples

```
decompose_crop_livestock_conn(example = TRUE)
```

`decompose_cropland_surplus` *Decompose cropland N surplus into size, intensity, and inefficiency drivers*

Description

Decomposes the year-on-year change in Spain's cropland nitrogen (N) surplus into three multiplicative drivers, following an additive LMDI (logarithmic mean Divisia index) shift-share decomposition computed at the national level:

- **Size**: national cropland area.
- **Intensity**: N input per hectare of cropland.
- **Inefficiency**: surplus fraction of inputs (1 - nitrogen use efficiency).

Contributions are additive and residual-free: they sum exactly to the observed change in cropland N surplus for every year-on-year transition.

This is a simplified, national-only view (no provincial or destiny breakdown); `decompose_specialization_cov()` still uses the full province x destiny detail for its `cropland_province` and `cropland_destiny` series.

Usage

```
decompose_cropland_surplus(
  n_prov_destiny = NULL,
  npp_ygpit = NULL,
  codes_coefs = NULL,
  by_period = FALSE,
  example = FALSE
)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`npp_ygpit`	Land use and area tibble from `whep_read_file("npp_ygpit")`. If `NULL`, loaded automatically.
`codes_coefs`	Item and biomass coefficients tibble from `whep_read_file("codes_coefs")`. If `NULL`, loaded automatically.
`by_period`	If `TRUE`, compares each reference period (each averaged across its ten years) against the immediately preceding one — 1860-1870 -> 1920-1930 -> 1960-1970 -> 2010-2020 — plus one extra transition spanning the full analysis window, 1860-1870 straight to 2010-2020 (the total change) — instead of chaining year on year.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble from `calculate_lmdi()` with columns `period`, `period_years`, `factor_label`, `component_type`, `additive`, `multiplicative`, and `multiplicative_log`.

Examples

```
decompose_cropland_surplus(example = TRUE)
```

`decompose_destiny_mix` *Compute the national cropland destiny mix over time*

Description

Computes, per year, the share of Spain's cropland output (by mass N) going to each destiny — domestic food, feed, exported food, non-food — from `create_n_nat_destiny()`'s national commodity balance sheets. Deliberately *not* `create_n_prov_destiny()`'s provincial data summed up: the provincial `"export"` destiny does not distinguish inter-provincial trade from true international export (per `decompose_crop_livestock_conn()`'s same caveat

on "Outside" imports), which would overstate the export share here. create_n_nat_destiny() instead recomputes export/import directly from the national production-vs-consumption balance per item, so trade between two Spanish provinces nets out rather than counting as export. This is the supplementary diagnostic recommended in the decomposition proposal (section 14, "attach to existing figures"), to show the local-food -> feed + export transition directly, since decompose_cropland_surplus() no longer carries a destiny factor.

Usage

```
decompose_destiny_mix(n_nat_destiny = NULL, example = FALSE)
```

Arguments

n_nat_destiny	National nitrogen flows tibble from create_n_nat_destiny(). If NULL, computed automatically (slow).
example	If TRUE, return a small hardcoded output without downloading remote data. Default is FALSE.

Value

A tibble with columns year, destiny_grp (one of "domestic_food", "feed", "exported", "non_food"), output_mg, and share (that destiny's share of the year's total cropland output).

Examples

```
decompose_destiny_mix(example = TRUE)
```

---

decompose_manure_losses

Decompose livestock manure management losses into herd, feed, excretion, and management-loss drivers

---

Description

Decomposes the year-on-year change in nitrogen (N) lost from livestock housing and manure storage (before any manure reaches land) into four multiplicative drivers, computed at the national level: herd size, feed N intake per livestock unit, excreted fraction of feed N (1 - feed nitrogen use efficiency), and the management-loss fraction of excreted N.

This is a simplified, national-only view with no species breakdown (no species-mix factor); decompose_specialization_cov() still uses the full per-species detail for its livestock_species series. Only livestock categories with a livestock-unit (LU) coefficient in livestock_units are included in the underlying herd/feed/excretion totals (currently Cattle_milk, Cattle_meat, Sheep, Goats, Horses, Donkeys_mules, Pigs, Poultry, Rabbits); categories present in intake_ygiac/n_excretion_ygs but absent from livestock_units (e.g. "Fur animals", "Other", "Other_birds") are dropped.

n_prov_destiny records manure already applied to land without retaining which species it came from, so the management-loss fraction has always been computed nationally, not per species.

Usage

```
decompose_manure_losses(
  n_prov_destiny = NULL,
  intake_ygiac = NULL,
  n_excretion_ygs = NULL,
  stock_prod_ygps = NULL,
  livestock_units = NULL,
  by_period = FALSE,
  example = FALSE
)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`intake_ygiac`	Feed intake tibble from `whep_read_file("intake_ygiac")`. If `NULL`, loaded automatically.
`n_excretion_ygs`	Livestock excretion tibble from `whep_read_file("n_excretion_ygs")`. If `NULL`, loaded automatically.
`stock_prod_ygps`	Livestock stock tibble from `whep_read_file("stock_prod_ygps")`. If `NULL`, loaded automatically.
`livestock_units`	Livestock unit coefficients tibble from `whep_read_file("livestock_units")`. If `NULL`, loaded automatically.
`by_period`	If `TRUE`, compares each reference period (each averaged across its ten years) against the immediately preceding one — 1860-1870 -> 1920-1930 -> 1960-1970 -> 2010-2020 — plus one extra transition spanning the full analysis window, 1860-1870 straight to 2010-2020 (the total change) — instead of chaining year on year.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble from `calculate_lmdi()` with columns `period`, `period_years`, `factor_label`, `component_type`, `additive`, `multiplicative`, and `multiplicative_log`.

Examples

```
decompose_manure_losses(example = TRUE)
```

`decompose_semi_natural_surplus`
Decompose semi-natural agroecosystem N surplus into size, intensity, and inefficiency drivers

Description

Decomposes the year-on-year change in Spain's semi-natural agroecosystem (grazing land, dehesa, and non-cropland vegetation) nitrogen (N) surplus into three multiplicative drivers, following an additive LMDI shift-share decomposition computed at the national level:

- **Size**: national semi-natural area.
- **Intensity**: N input per hectare of semi-natural land.
- **Inefficiency**: surplus fraction of inputs (1 - nitrogen use efficiency).

No destiny factor is used because grazed and cut vegetation is assumed to be overwhelmingly a single destiny (livestock feed).

The land-use categories included (`Dehesa`, `Forest_high`, `Forest_low`, `Other`, `Pasture_Shrubland`) are all of `npp_ygpit`'s non-cropland categories, matching the existing `semi_natural_agroecosystems` box used elsewhere in the package. Some of that land (e.g. `Forest_high`/ `Forest_low`) may not actually be grazed and can produce non-feed output (firewood), which would call for its own destiny factor (as in `decompose_cropland_surplus()`) rather than the single-destiny assumption above; that refinement is not implemented here.

This is a simplified, national-only view (no provincial breakdown). Semi-natural surplus can turn negative (soil N mining) in some years. LMDI relies on logarithms and cannot handle a series that changes sign between two compared years; this function warns when that occurs instead of silently returning `NA`, but does not implement the Shapley/Sun alternative required for those cases.

Usage

```
decompose_semi_natural_surplus(
  n_prov_destiny = NULL,
  npp_ygpit = NULL,
  by_period = FALSE,
  example = FALSE
)
```

Arguments

`n_prov_destiny`
Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.

`npp_ygpit`
Land use and area tibble from `whep_read_file("npp_ygpit")`. If `NULL`, loaded automatically.

`by_period` If `TRUE`, compares each reference period (each averaged across its ten years) against the immediately preceding one — 1860-1870 -> 1920-1930 -> 1960-1970 -> 2010-2020 — plus one extra transition spanning the full analysis window, 1860-1870 straight to 2010-2020 (the total change) — instead of chaining year on year.

`example` If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble from `calculate_lmdi()` with columns `period`, `period_years`, `factor_label`, `component_type`, `additive`, `multiplicative`, and `multiplicative_log`.

Examples

```
decompose_semi_natural_surplus(example = TRUE)
```

---

decompose_specialization_cov

Decompose specialization from diversification via the Olley-Pakes allocation covariance

---

Description

`decompose_cropland_surplus()`, `decompose_semi_natural_surplus()`, and `decompose_manure_losses()` are all simplified to national-only views with no spatial, destiny, or species-mix factor, so the LMDI "Specialization" mechanism (in `decompose_terr_losses()`) is currently empty. This function recovers the provincial and species allocation signal independently, straight from the underlying panels: it shows whether the allocation of area or herd across units (provinces, destinies, species) concentrated into high-surplus units (genuine specialization) or spread towards low-surplus ones (diversification) — a distinction the mix alone cannot make. This function adds that signal, following the Olley-Pakes allocation identity used in the decomposition proposal (`sum(w_i * s_i) = mean(s) + covariance(w_i, s_i)`): for a set of units with area/herd share `w_i` and per-unit surplus `s_i`, the covariance between the two is positive and growing when the allocation concentrates into high-surplus units (specialization raising surplus), and shrinks towards zero or turns negative under diversification.

Unlike the additive LMDI contributions (in Mg N), the covariance is expressed in per-unit-area or per-unit-herd surplus terms (Mg N per ha, or Mg N per livestock unit) — it is not directly comparable in magnitude to the "Specialization" mechanism total from `decompose_terr_losses()`, only in sign and trend.

Usage

```
decompose_specialization_cov(
  n_prov_destiny = NULL,
  raw = NULL,
  example = FALSE
)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`raw`	Named list overriding any of the raw inputs (`npp_ygpit`, `codes_coefs`, `intake_ygiac`, `n_excretion_ygs`, `stock_prod_ygps`, `livestock_units`). Missing elements are loaded automatically.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A named list with tibbles `cropland_province`, `cropland_destiny`, and `livestock_species`, each with columns `year` and `covariance`.

Examples

```
decompose_specialization_cov(example = TRUE)
```

`decompose_terr_losses` *Decompose total territorial N losses into compartments and mechanisms*

Description

Runs the cropland (`decompose_cropland_surplus()`), semi-natural (`decompose_semi_natural_surplus()`), manure (`decompose_manure_losses()`), and urban (`decompose_urban_losses()`) decompositions, then combines them into two cumulative, year-on-year contribution series:

- `by_compartment`: change in total territorial N losses attributed to each of the four compartments.
- `by_mechanism`: the same total change regrouped across compartments into scale, intensification, and efficiency (population and per-capita excretion are grouped under scale, since together they represent total human N throughput). Cropland, semi-natural, and manure no longer carry a spatial/destiny/species-mix factor (see their own simplified decompositions), so no factor currently maps to a "specialization" mechanism — that signal now lives only in `decompose_specialization_cov()` and `decompose_crop_livestock_conn()`.

Usage

```
decompose_terr_losses(n_prov_destiny = NULL, raw = NULL, example = FALSE)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`, shared across all four compartments. If `NULL`, loaded automatically.
`raw`	Named list overriding any of the raw inputs shared across compartments (`npp_ygpit`, `codes_coefs`, `intake_ygiac`, `n_excretion_ygs`, `stock_prod_ygps`, `livestock_units`, `population_yg`). Missing elements are loaded automatically.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A named list with tibbles `detail` (per-compartment LMDI output, with `compartment` and `mechanism` columns added), `by_compartment`, and `by_mechanism` (each with `t0`, `contribution_mgn`, and `cumulative_mgn`).

Examples

```
decompose_terr_losses(example = TRUE)
```

`decompose_terr_losses_periods` *Decompose territorial N losses by reference period (chained)*

Description

Runs the same four compartments as `decompose_terr_losses()`, but comparing each reference period (each averaged across its ten years) against the immediately preceding one — 1860-1870 -> 1920-1930 -> 1960-1970 -> 2010-2020 — plus one extra transition spanning the full analysis window, 1860-1870 straight to 2010-2020 (the total change), instead of chaining year on year. This is the periodised table recommended alongside the main chained figure in the decomposition proposal (section 12), summarizing the four historical phases rather than following the full 160-year trajectory.

Usage

```
decompose_terr_losses_periods(
  n_prov_destiny = NULL,
  raw = NULL,
  example = FALSE
)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`, shared across all four compartments. If `NULL`, loaded automatically.
`raw`	Named list overriding any of the raw inputs shared across compartments (`npp_ygpit`, `codes_coefs`, `intake_ygiac`, `n_excretion_ygs`, `stock_prod_ygps`, `livestock_units`, `population_yg`). Missing elements are loaded automatically.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A named list with tibbles `detail` (per-compartment LMDI output, with `compartment` and `mechanism` columns added), `by_compartment`, and `by_mechanism` (each with `period` — one of "1865-1925", "1925-1965", "1965-2015", "Total (1865-2015)", the mean year of each reference window (1865 = mean of 1860-1870, and so on) — plus `contribution_mgn`, `period_years`, and `contribution_per_yr_mgn`, the per-year-normalized value used for plotting).

Examples

```
decompose_terr_losses_periods(example = TRUE)
```

`decompose_urban_losses`

Decompose urban nitrogen losses into population, per-capita, and recycling drivers

---

Description

Decomposes the year-on-year change in non-recycled human excreta nitrogen (N) into three multiplicative drivers, computed at the national level: population, per-capita excreted N (approximated by per-capita food N consumption, since intake is approximately equal to excretion), and the non-recycled fraction of excreted N.

Only the `population_food` destiny is used as the excretion proxy. `population_other_uses` (non-food industrial use, e.g. cotton/tobacco) is deliberately excluded even though `.create_wastewater_surplus_` in `grafs_plot_df.R` includes it: that material is not ingested, so it cannot be assumed to leave the body as excreta, breaking the intake-approximates-excretion logic this compartment relies on.

Food waste is not included: the commodity-balance sheets underlying `n_prov_destiny` carry Food/Feed/Seed/Other-uses/Export/Import destinies but no separate consumer food-waste line, so this loss is excreta-only. Whether the `urban/People` recycling flows already folded in some food waste from the original source is unconfirmed.

Usage

```
decompose_urban_losses(
  n_prov_destiny = NULL,
  population_yg = NULL,
  by_period = FALSE,
  example = FALSE
)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`population_yg`	Population tibble from `whep_read_file("population_yg")`. If `NULL`, loaded automatically.
`by_period`	If `TRUE`, compares each reference period (each averaged across its ten years) against the immediately preceding one — 1860-1870 -> 1920-1930 -> 1960-1970 -> 2010-2020 — plus one extra transition spanning the full analysis window, 1860-1870 straight to 2010-2020 (the total change) — instead of chaining year on year.
`example`	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble from `calculate_lmdi()` with columns `period`, `period_years`, `factor_label`, `component_type`, `additive`, `multiplicative`, and `multiplicative_log`.

Examples

```
decompose_urban_losses(example = TRUE)
```

`decompose_weighted_ratio`

Decompose a weighted aggregate ratio.

Description

Separates a change in an aggregate numerator-to-denominator ratio into a between-group composition effect and a within-group ratio effect. The default symmetric Kitagawa allocation is equivalent to a two-factor Shapley value for the weight and within-group-ratio factor blocks.

Usage

```
decompose_weighted_ratio(
  data,
  time,
  .by,
  ratio,
  method = c("kitagawa", "lmdi", "weights_first", "ratios_first", "all")
)
```

Arguments

`data`	A tibble with exactly two periods and one row per group-period.
`time`	An unquoted numeric, date-time, or ordered-factor column identifying the two ordered periods.
`.by`	An unquoted column identifying persistent groups.
`ratio`	An unquoted expression of the exact form `numerator / denominator`, using two bare numeric column names.
`method`	Decomposition method. One of `"kitagawa"`, `"lmdi"`, `"weights_first"`, `"ratios_first"`, or `"all"`.

Details

For group `g` and endpoint `t`, the aggregate ratio is `R_t = sum_g(w_gt * r_gt)`, where `w_gt` is the group's share of the total denominator and `r_gt` is its numerator-to-denominator ratio.

Write `dw_g = w_g1 - w_g0` and `dr_g = r_g1 - r_g0`. The symmetric Kitagawa contributions are

$$B_g = dw_g(r_g0 + r_g1)/2,$$

$$W_g = (w_g0 + w_g1)dr_g/2.$$

The weights-first polar contributions are `B_g = dw_g r_g0` and `W_g = w_g1 dr_g`; the ratios-first polar contributions are `B_g = dw_g r_g1` and `W_g = w_g0 dr_g`. Kitagawa is their arithmetic mean. Its Shapley interpretation concerns the complete weight and ratio factor blocks; groups are not treated as Shapley players.

`"lmdi"` implements additive LMDI-I. With `y_gt = w_gt r_gt` and the logarithmic mean `L(a, b) = (a - b) / (log(a) - log(b))`, using `L(a, a) = a`, its contributions are

$$B_g = L(y_g1, y_g0)log(w_g1/w_g0),$$

$$W_g = L(y_g1, y_g0)log(r_g1/r_g0).$$

Every method satisfies `B_g + W_g = y_g1 - y_g0` and therefore `sum_g(B_g + W_g) = R_1 - R_0`, up to floating-point error. `"all"` returns every method on their common strictly positive domain.

The function requires exactly two periods and identical, unique group support. Denominators must be positive. Numerators may be zero for the arithmetic methods but must be positive for LMDI. Groups are never silently dropped and zeros are never replaced by an epsilon.

Reversing endpoints negates Kitagawa and LMDI-I contributions. Reversing a polar path negates the opposite forward polar path. Percentage contributions are signed and are not clipped to zero or 100. They are missing when `abs(R_1 - R_0)` is no larger than `sqrt(.Machine$double.eps) * max(abs(R_0), abs(R_1))`. High `cancellation_index` values indicate cancellation between the aggregate between and within effects. It is defined as `1 - abs(total_change) / (abs(between) + abs(within))`, with zero used when every effect is zero. Results are descriptive accounting identities, not causal attribution, and depend on the chosen group resolution.

Value

A tibble with one `component_type = "group"` row per group and one `component_type = "summary"` row for each requested method. Group rows contain endpoint stocks, weights, ratios, signed contributions, and group closure. Summary rows contain aggregate endpoint ratios, signed effects, percentage contributions, closure diagnostics, and cancellation metrics. Additive signed contributions have the same units as `ratio`.

References

Kitagawa, E. M. (1955). Components of a Difference Between Two Rates. *Journal of the American Statistical Association*, 50(272), 1168-1194. doi:10.1080/01621459.1955.10501299.

Ang, B. W. (2005). The LMDI approach to decomposition analysis: a practical guide. *Energy Policy*, 33(7), 867-871. doi:10.1016/j.enpol.2003.10.010.

Examples

```
ratio_data <- tibble::tribble(
  ~year, ~group, ~numerator, ~denominator,
  2000, "a", 400, 40,
  2000, "b", 1200, 60,
  2020, "a", 600, 50,
  2020, "b", 900, 50
)

decompose_weighted_ratio(
  ratio_data,
  year,
  group,
  numerator / denominator
)
```

`disaggregate_ussr` *Split the pre-1992 aggregate USSR supply into successor states.*

Description

For years before `cutoff_year`, replaces the single aggregate USSR row (the `area_code` listed as `ussr_area_code` in `ussr_shares`) with one row per successor state, inheriting the aggregate's per-capita supply values and scaling its population by the successor's 1992 population share, so the split conserves the aggregate population and each successor carries the aggregate per-capita supply. Rows at or after `cutoff_year`, and all non-USSR rows, pass through unchanged. This mirrors the Global SJOS-N pre-1992 USSR disaggregation, which distributes the Russian Federation Food Balance Sheet supply across the ex-USSR states by their 1992 population weights.

Usage

```
disaggregate_ussr(x, ussr_shares, cutoff_year = 1992L)
```

Arguments

`x`	A tibble with `year`, `area_code` and `population` columns (plus any per-capita supply columns to inherit), for example a `build_food_supply()` output.
`ussr_shares`	A lookup tibble with `ussr_area_code` (the aggregate USSR area code), `successor_area_code` (a successor state's area code) and `pop_share` (the successor's 1992 population share, summing to one per aggregate). Injected as a package-data / integration input.
`cutoff_year`	The first year for which successor states report separately; the split applies to years strictly before it. Defaults to `1992L`.

Value

`x` with the pre-cutoff aggregate USSR rows replaced by their successor-state rows.

Examples

```
disaggregate_ussr(
  tibble::tribble(
    ~year, ~area_code, ~protein_g_cap_day, ~population,
    1990L, 228L, 50, 100,
    1995L, 228L, 55, 120
  ),
  tibble::tribble(
    ~ussr_area_code, ~successor_area_code, ~pop_share,
    228L, 1L, 0.6,
    228L, 2L, 0.4
  )
)
```

`drainage_ranges` *Drainage-class bins keyed on annual soil drainage.*

Description

Half-open bins that map an annual soil drainage flux (mm) to a drainage class. A drainage value `s` is assigned to the class whose interval satisfies `s_min < s <= s_max`. The classes key the Meisinger denitrification matrix and the subsoil NO3 reduction table.

Usage

```
drainage_ranges
```

Format

A tibble with columns:

drainage_rate Drainage class: `"Very_high"`, `"High"`, `"Medium"`, `"Low"`, `"Very_low"` or `"None"`.

s_min Lower bound of the drainage flux interval (mm).

s_max Upper bound of the drainage flux interval (mm).

Source

Nitrogen coefficient workbook of an earlier regional historical reconstruction, companion to the Meisinger & Randall (1991) denitrification matrix. doi:10.2136/1991.managingnitrogen.c5.

Examples

```
drainage_ranges
```

`empty_table_from_schema` *Build a typed zero-row table from a declarative schema.*

Description

Turn the same serializable schema `check_table_schema()` validates into the zero-row tibble it describes: the declared column names, in the declared order, each with the declared type and no rows. The result passes `check_table_schema()` by construction, because both functions resolve the schema through one parser, so a prototype and the validator that judges it cannot drift apart.

This is the missing half of `ensure_columns()`, which *needs* a zero-row prototype tibble and cannot be handed a schema. Declare the schema once as data — in a YAML file beside the artifact, say — then `empty_table_from_schema()` for the prototype, `ensure_columns()` to coerce a table onto it, and `assert_table_schema()` to prove the result.

Usage

```
empty_table_from_schema(schema)
```

Arguments

`schema`	Declarative schema, as described above.

Value

A zero-row tibble with one column per declared column, in declared order. A schema declaring no columns yields a 0x0 tibble.

Types

`"logical"`, `"integer"`, `"double"`, `"character"`, `"Date"` and `"list"` produce a column of exactly that type. `"any"` declares that the schema does not constrain the type, so there is no type to build: the column is created as `logical()`, the type of a bare `NA` and the one any later cast widens from. Fields other than `name` and `type` (`min`, `allowed`, `key`, ...) constrain values, of which a zero-row table has none, so they only have to parse.

A schema with `allow_empty = FALSE` has no valid empty table, and is rejected rather than returning one that fails its own validation.

Examples

```
schema <- list(
  columns = list(
    list(name = "year", type = "integer", min = 1961, max = 2023),
    list(name = "area_code", type = "integer"),
    list(name = "source", type = "character"),
    list(name = "value", type = "double", min = 0)
  ),
  key = c("year", "area_code")
)
prototype <- empty_table_from_schema(schema)
prototype

# It is a prototype: `ensure_columns()` coerces a partial table onto it.
ensure_columns(tibble::tibble(year = 2020L, value = 1.5), prototype)

# And it conforms to the schema it was built from.
nrow(check_table_schema(prototype, schema))
```

`ensure_admin_shares` *Complete a table to the admin-shares contract*

Description

Complete `x` onto `admin_shares_prototype()` with `ensure_columns()` – adding absent columns as typed missing values and casting present ones to the contract's types – then prove the result against `admin_shares_schema()` with `assert_table_schema()`, which aborts naming the offending columns and values when it does not conform.

Two rules are proved here rather than in the schema, because a `check_table_schema()` specification speaks of one column at a time and skips a missing value on every bound:

- Every row must carry `value`, `share`, or both. A row with neither aborts with class `whep_error_admin_no_measure`. A row with `share` alone is accepted – see the *Shares-only rows* section of `admin_shares_schema()`, and never repair one by inventing a value.
- Neither measurement may be non-finite. `NaN` or `Inf` aborts with class `whep_error_admin_nonfinite`, because `is.na(NaN)` is `TRUE` and a `NaN` would otherwise travel as a shares-only row.

Usage

```
ensure_admin_shares(x)
```

Arguments

`x`	Tibble to complete. May already carry extra columns or be missing contract columns; see `ensure_columns()`.

Value

`x` completed to the admin-shares contract, invisibly.

Examples

```
# Omits columns the contract allows missing: `share`, `nuts_version`,
# `source_native_id`, `source_native_name`, `source_version` and
# `value_flag`. (`value` is allow-missing too, but a row must carry it
# or `share`, so this valued fixture keeps it.) `level_polity_code` is
# part of the key, so it must stay present and distinct per row even
# though the contract allows it to be `NA` for a genuinely unresolved
# row.
partial <- tibble::tibble(
  area_code = c(840L, 840L),
  level_polity_code = c("USA-IOWA", "USA-ILLINOIS"),
  level = c(1L, 1L),
  item_prod_code = c(44L, 44L),
  indicator_used = c("area_harvested", "area_harvested"),
  year = c(2020L, 2020L),
  value = c(1000000, 800000),
  source = c("USDA_NASS", "USDA_NASS"),
  tier = c(1L, 1L),
  grain = c("admin1", "admin1"),
  concept_break = c(FALSE, FALSE),
  source_id = c("USDA_NASS", "USDA_NASS"),
  recorded_at = "2026-01-01T00:00:00Z",
```

```
  treatment_year = c("observed", "observed")
)
completed <- ensure_admin_shares(partial)
names(completed)
nrow(check_table_schema(completed, admin_shares_schema()))
```

`ensure_columns` *Complete columns from a typed prototype.*

Description

Add absent columns to a tibble and safely cast present columns to the types declared by a zero-row prototype. Missing columns receive typed missing values unless a scalar default is supplied. Defaults never replace missing values inside a column that is already present.

Prototype columns are returned first in prototype order. With `extra = "keep"`, other input columns follow in their original relative order. With `extra = "drop"`, the output has exactly the prototype schema.

Usage

```
ensure_columns(data, prototype, defaults = NULL, extra = c("keep", "drop"))
```

Arguments

`data`	Input tibble.
`prototype`	Zero-row tibble defining required column names, types, and order.
`defaults`	Optional named list of size-one defaults for absent columns. Every name must occur in `prototype`, and each value must be safely convertible to the corresponding prototype type.
`extra`	Whether columns absent from `prototype` are `"keep"` or `"drop"`.

Value

An ungrouped tibble with the same rows as `data`, completed and ordered from `prototype`. Grouping metadata is not preserved.

Examples

```
prototype <- tibble::tibble(
  year = integer(),
  value = double(),
  source = character()
)
data <- tibble::tibble(value = 2, year = 2020L, note = "observed")

ensure_columns(
  data,
  prototype,
  defaults = list(source = "unknown")
)
```

`estimate_energy_demand` *Estimate energy demand (Gross Energy) - Tier 2*

Description

Calculate gross energy (GE) intake per IPCC 2019 Tier 2 equations (Vol 4, Ch 10). Estimates net energy components for maintenance, activity, lactation, work, pregnancy, growth, and wool, then derives total gross energy using the REM/REG ratio approach from IPCC Eq 10.16.

All coefficients come from internal package data.

Default weight gain and pregnancy fraction follow the cohort's life stage: mature cohorts (`"Adult Female"`, `"Adult Male"`, `"Adult"`, `"Sows"`, `"Boars"`, `"Layers"`) gain no weight, and only mature-female cohorts take the species pregnancy fraction (IPCC 2019 Vol 4 Ch 10, pp. 10.18 and 10.28). Beef breeding cows (`"Adult Female"` of non-dairy cattle) take the "Pregnant, %" of the IPCC 2019 Table 10A.2 "Mature Females" row for their IPCC region, resolved from `iso3` or `area_code`. A cohort of `NA` or `"All"` is a whole-herd row and keeps both species defaults. Any other cohort label aborts. Values the caller supplies are kept.

Usage

```
estimate_energy_demand(
  data,
  method = "ipcc2019",
  lactation_method = c("milk_composition", "ipcc2019")
)
```

Arguments

`data`	A dataframe with columns `species`, `cohort`, `heads`, and optionally `iso3`. Optional production columns: `weight`, `milk_yield_kg_day`, `fat_percent`, `weight_gain_kg_day`, `work_hours_day`, `work_coef`, `cfi`, `pregnant_fraction`, `temperature_c`, `diet_quality`, `grazing_distance_km`, `system`. `work_coef` overrides the joined IPCC work coefficient (`cw`, 0 by default for every species) for rows that need draught/work energy (IPCC Eq 10.11) without changing the global default. `cfi` overrides the joined maintenance coefficient (`cfi_mj_day_kg075`, IPCC Eq 10.3) for rows whose herd-average maintenance requirement is known from a national inventory (e.g. a housed dairy herd calibrated to a Zootecnicas/NIR Cfi), without changing the global default.
`method`	Method for calculation (default `"ipcc2019"`).
`lactation_method`	How net energy for lactation (NEl) is derived from milk yield. One of:

- `"milk_composition"` (default): the NRC (2001) milk-energy equation, `NEl = Milk * (0.389 * Fat + 0.229 * Protein + 0.165 * Lactose)` (MJ/kg: the published Mcal/kg coefficients 0.0929, 0.0547 and 0.0395 times 4.184), for rows with a positive protein and lactose content. Rows without that composition use the `"ipcc2019"` equations.
- `"ipcc2019"`: IPCC 2019 Refinement Vol 4 Ch 10. Eq 10.8, `NEl = Milk * (1.47 + 0.40 * Fat)`, for cattle, buffalo and other species; Eq 10.9, `NEl = Milk * EVmilk`, for sheep and goats with the default `EVmilk` of 4.6 MJ/kg for sheep (7% fat; AFRC 1993, 1995) and 3 MJ/kg for goats (3.8% fat; AFRC 1998). The defaults ignore `fat_percent`.

The equation used for each row is recorded in `method_lactation` (`"nrc2001_milk_composition"` `"ipcc2019_eq10_8"`, `"ipcc2019_eq10_9_default_ev"`, or `"none"` when there is no milk).

Value

Dataframe with added `gross_energy` (MJ/day), intermediate net energy components, `cohort_life_stage` (`"mature_female"`, `"mature"`, `"growing"`, or `NA` for a whole-herd row), and `method_energy`, `method_lactation` and `method_pregnancy` tracking columns. `method_pregnancy` is `"supplied"`, `"species_default"`, `"species_default_whole_herd"`, `"not_mature_female"`, `"ipcc2019_t10a2"` (a beef cow at its region's published rate), `"ipcc2019_t10a2_neighbour_assumed"` (Western Europe, which the table omits, at the 80% North America and Eastern Europe report) or `"ipcc2019_t10a2_mean_assumed"` (no resolvable region: the mean of the published regions).

Examples

```
tibble::tibble(
  species = "Dairy Cattle", cohort = "Adult Female",
  heads = 100, weight = 600, diet_quality = "High",
  milk_yield_kg_day = 20
) |>
  estimate_energy_demand() |>
  dplyr::select(species, cohort, heads, ne_maintenance,
    ne_activity, ne_lactation, ne_growth, gross_energy)
```

`estimate_n_excretion` *Estimate livestock nitrogen, carbon and volatile-solids excretion.*

Description

Converts realised feed intake (the output of `redistribute_feed()`) into excreted nitrogen, carbon and volatile solids per `year x territory x sub_territory x livestock_category`. All excretion methods share one canonical nitrogen intake, `n_intake = sum(intake_dm_t * feed_n_content)`, so the methods are directly comparable.

Carbon is the carbon of the excreted organic matter, `vs_excretion * c_vs_fraction`, so it inherits the intake-and-digestibility mass balance of the volatile solids. Urine carbon

is not part of the volatile solids under `method_vs = "intake_digestibility"` and is not counted: at a C:N near 0.9 it is under a tenth of a dairy cow's excreted carbon (Dijkstra et al. 2018, Table 5) and is respired within days of deposition, so it never reaches the soil carbon this feeds.

Usage

```
estimate_n_excretion(intake, options = list())
```

Arguments

`intake`	A tibble of realised feed intake with at least `year`, `territory`, `sub_territory`, `livestock_category`, `item_cbs_code`, `feed_quality` and `intake_dm_t` (the `redistribute_feed()` result). The output of `get_feed_intake()` does not fit: it is keyed by `area_code` and `live_anim_code` and carries no `intake_dm_t`, and it is refused with a pointer to `redistribute_feed()`. `territory` is a stringified `area_code` (`as.character(area_code)`, what `redistribute_feed()` emits and what the whole manure chain carries through to the nitrogen inputs); an `iso3c` literal is still resolved there but is deprecated, since it can only answer with an aggregation bucket and so loses the territory for 62 of the 257 codes it knows.
`options`	A named list of method options: • `method`: `"intake_minus_retention"` (default, `n_intake * (1 - n_retention_frac)`) or `"intake_minus_product_n"` (`n_intake - product_n`). • `method_vs`: `"intake_digestibility"` (default, `intake_dm_t * (1 - digestibility) * (1 - ash)`). • `method_digestibility`: where `digestibility` comes from. `"species_feed_quality"` (default) looks it up per species and feed quality: pigs and poultry fed concentrates digest far more of them than a ruminant does, so they take the IPCC (2019) Table 10.2 (Updated) values for confined monogastrics; see Details. `"feed_quality"` is the earlier species-blind lookup, one value per feed quality for every species, kept for comparison (whep#1007). • `method_c`: `"volatile_solids"` (the only method, `vs_excretion * c_vs_fraction`). The route it replaced, `n_excretion` times the `bio_coefs Excreta` C:N, is deliberately not offered as an alternative: it applied a fresh-dung C:N to whole-excreta nitrogen, some 60% of which is urinary and carries almost no carbon, and returned 0.73 kg C per kg of volatile solids – above pure protein, so not a composition organic matter can have (whep#1006). A route that cannot be right is not a sensitivity case, so this option names the algebra behind `c_excretion` rather than a choice, and `method_c_excretion` has one value until a second defensible method exists (whep#1100). • `c_vs_fraction`: carbon per unit of volatile solids, kg C / kg VS. Default 0.47; see Details. • `forage_n`: nitrogen content of the grazed forage that intake rows with no `item_cbs_code` take. `"assumed_midrange"` (default, 0.02 kg N/kg DM, an assumed unverified value), `"gleam_grass_fresh"`

(0.022), "gleam_grass_hay" (0.017) and "gleam_grass_mean" from GLEAM 3.0 Supplement S1 Tab. S.3.3, or "biomass_coefs_grass" (0.0174) from the bio_coefs Grass row.

- product_n: a tibble (year, territory, sub_territory, livestock_category, product_n) required by "intake_minus_product_n".

Details

The default c_vs_fraction of 0.47 kg C per kg of volatile solids is the carbon content of the components that make up faecal organic matter in Dijkstra et al. (2018, Front. Sustain. Food Syst. 2:63, doi:10.3389/fsufs.2018.00063, Table 1): fibre 0.44, microbial organic matter 0.47, protein 0.52 and lipids 0.75 g C per g dry matter, weighted towards the fibre and microbial debris that dominate faeces. Measured manures bracket it: 0.52 for fresh bedded dairy manure (Choi et al. 2022, PeerJ 10:e14134, doi:10.7717/peerj.14134, Table 1: 43.3% C and 83.3% VS of dry matter) and 0.39-0.46 for stored cattle and pig manure (Baek et al. 2020, Int. J. Environ. Res. Public Health 17:4737, doi:10.3390/ijerph17134737, Table 1).

Digestibility:

Digestibility here is the digestible share of the feed's gross energy (DE), which is what IPCC (2019) Eq. 10.24 uses for volatile solids. The per-feed-quality values are a scaffold anchored on the GLEAM 3.0 ruminant feed table (gleam_feed_digestibility: fresh grass 66 and hay 58, straws 44-47) and are applied to every species under "feed_quality". Under "species_feed_quality" three of them are replaced by the representative values of the IPCC 2019 Refinement, Vol. 4, Ch. 10, Table 10.2 (Updated), taking the mid-point of each range:

- pigs, high_quality: 0.85, growing swine in confinement (80-90). Market swine are the chain's representative pig category; mature confined swine are 70-80. The GLEAM 3.0 pig feed table (gleam_feed_conversion_ratios, Table S.3.4) agrees: maize 88, wheat 83, barley 76, soybean meal 80.
- pigs, scavenging: 0.60, free-range swine (50-70).
- poultry, high_quality: 0.89, broilers in confinement (85-93). Broilers are the representative poultry category of the whole manure chain; layers in confinement are 70-80.

Every other livestock category and feed quality keeps the scaffold value (rabbits too, although they borrow the swine coefficients elsewhere), and so does crop-residue feed for pigs and poultry: the GLEAM pig table gives straw 49 and crop tops 52, the same as the ruminant straws.

Value

A tibble with one row per year x territory x sub_territory x livestock_category and columns n_intake, n_excretion, c_excretion, vs_excretion, method_n_excretion, method_vs, method_digestibility, method_c_excretion and method_forage_n.

Examples

```
intake <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~livestock_category,
  ~item_cbs_code, ~feed_quality, ~intake_dm_t,
```

```
  2020L, "203", NA, "Cattle_milk", 2513L, "high_quality", 100,
  2020L, "203", NA, "Cattle_milk", NA, "grass", 500
)
estimate_n_excretion(intake)
```

`europe_fao_spain_trade`

Spain's historical export and import series, 1849-1960

Description

Spain's rows of the Export and Import sheets of the compilation `Europe_FAO_completed.xlsx`, reshaped to long format and converted from thousand tonnes to Mg of fresh matter. It is the raw historical trade series that `validate_national_trade_raw()` and `compute_trade_flows_raw()` compare the provincial model against, after crosswalking `Item` to CBS items through `cbs_trade_codes` and converting fresh matter to nitrogen.

The workbook itself is a 19-sheet, 35-country compilation and is not redistributed with the package; only this Spanish extract is. It is built by `data-raw/europe_fao_spain_trade_extract.R` (workbook to CSV) and `data-raw/europe_fao_spain_trade.R` (CSV to this dataset).

Usage

```
europe_fao_spain_trade
```

Format

A tibble with one row per flow, item and year:

- `Element`: Character, `"Export"` or `"Import"`.
- `Item`: Character, trade item name as labelled in the workbook.
- `Year`: Integer year, 1849 to 1960.
- `value_fm`: Double, traded quantity in Mg of fresh matter.

Source

Spain's rows of `Europe_FAO_completed.xlsx`; see `data-raw/europe_fao_spain_trade_extract.R` in the source repository.

Examples

```
head(europe_fao_spain_trade)
```

`evidence_conflicts` *Report where row evidence disagrees.*

Description

Find the `(row_key, field)` pairs carrying more than one distinct value, i.e. the places where two producers, or two runs of one producer, made different claims about the same row. Reports rather than aborts, so a caller can decide which disagreements matter; `nrow() == 0` is the clean result.

A repeated *identical* claim is not a conflict, so re-recording the same evidence is harmless.

Usage

```
evidence_conflicts(evidence)
```

Arguments

`evidence`	Evidence table, validated against `row_evidence_schema()`.

Value

A tibble with one row per conflicting `(row_key, field)`, ordered by both:

- `row_key`, `field`: The row and evidence field in dispute.
- `n_values`: Number of distinct values claimed. `NA` counts as a value of its own, since "unknown" and "0.5" are a disagreement.
- `values`: The claimed values, `" | "`-separated, with `NA` rendered as `"NA"`.
- `source_ids`: The producers involved, comma-separated.

Examples

```
rows <- tibble::tibble(area_code = 724L, year = 2020L)
key <- c("area_code", "year")
recorded_at <- as.POSIXct("2026-01-01", tz = "UTC")
agreeing <- combine_row_evidence(
  row_evidence(rows, "A", key, list(flag = "E"), recorded_at = recorded_at),
  row_evidence(rows, "B", key, list(flag = "E"), recorded_at = recorded_at)
)
evidence_conflicts(agreeing)

disagreeing <- combine_row_evidence(
  agreeing,
  row_evidence(rows, "C", key, list(flag = "A"), recorded_at = recorded_at)
)
evidence_conflicts(disagreeing)
```

`evidence_for` *Carry row evidence through a join or a filter.*

Description

Re-align an evidence table onto a table that has since been joined, filtered, reordered or reduced: the evidence rows whose `row_key` still occurs in `data` are returned, in canonical order, with their `source_id` and `source_version` untouched. Evidence is therefore narrowed by composition but never re-attributed by it.

The key is taken from the evidence's own `key_columns`, not from the caller, so a table that no longer carries those columns — an aggregation that dropped `year`, say — aborts instead of matching on whatever is left. Rows of `data` that carry no evidence are a warning, not a silent gap: after a left join they are exactly the rows whose provenance was lost.

Usage

```
evidence_for(data, evidence)
```

Arguments

`data`	Table the evidence is being carried onto. Must carry the evidence's key columns. Duplicated keys are allowed: a fan-out join does not change which claims apply.
`evidence`	Evidence table, validated against `row_evidence_schema()`.

Value

A tibble of row evidence, in canonical order, containing the rows of `evidence` that apply to `data`.

Examples

```
rows <- tibble::tibble(area_code = c(724L, 76L), year = 2020L)
evidence <- row_evidence(
  rows,
  source_id = "FAOSTAT_prod",
  key = c("area_code", "year"),
  fields = list(fao_flag = c("A", "E")),
  recorded_at = as.POSIXct("2026-01-01", tz = "UTC")
)

# A filter narrows the evidence to the rows that survive.
evidence_for(dplyr::filter(rows, area_code == 724L), evidence)

# A row with no evidence is reported rather than passed over.
extended <- dplyr::bind_rows(rows, tibble::tibble(
  area_code = 231L,
  year = 2020L
```

```
))
suppressWarnings(evidence_for(extended, evidence))
```

`expand_polycell_years` *Expand the interval-keyed polycell support to one row per year*

Description

Repeats every polycell interval over the calendar years it covers, adding a `year` column. `start_year` is inclusive; `end_year` is **exclusive at a succession**, so a boundary year resolves to the successor alone and is never counted twice, and **inclusive at the open end**, so the last year the table covers still resolves to the polity nothing succeeds instead of to nothing at all.

Usage

```
expand_polycell_years(support, years)
```

Arguments

`support`	A `build_polycell_support()` table in the interval grain, carrying `cell_id`, `polity_code`, `start_year` and `end_year`. The first two are what identify successive intervals of one polity in one cell, and without them the open end cannot be told from a succession.
`years`	Integer vector of calendar years.

Value

A `tibble` with one row per polycell-year, `year` placed after `area_code`.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  build_polycell_support(geometries = polycell_example_geometries()) |>
    expand_polycell_years(2010L:2012L)
}
```

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)
```

feed_characteristics *Feed characteristics by diet quality.*

Description

DE%, NDF%, GE content, and crude protein percentage for High/Medium/Low diet quality levels.

Usage

```
feed_characteristics
```

Format

A tibble with `diet_quality`, `de_percent`, `ndf_percent`, `ge_content_mj_kg_dm`, `cp_percent`.

Source

IPCC 2019, Vol 4, Ch 10.

Examples

```
feed_characteristics
```

`feed_taxonomy` *Feed taxonomy.*

Description

Maps each feed item to its feed group, feed quality class, per consumer feed type labels and an allocation priority. Granivores get a restricted feed type set, so only grazers take fibrous roughage. Migrated from the afsetools `Codes_coefs.xlsx` workbook.

Usage

```
feed_taxonomy
```

Format

A tibble with one row per feed item:

item_cbs_code FAOSTAT commodity balance item code.

item_cbs Item name.

feed_group Feed group (crop or material class).

feed_quality Feed quality class: lactation, high_quality, low_quality, residues, grass, zoot_fixed or non_feed.

feed_quality_rank Allocation priority rank (lower is allocated first): lactation and high_quality 1, low_quality 2, residues 3, grass 4.

granivore_feedtype Feed type the item provides to granivores.

grazer_feedtype Feed type the item provides to grazers.

zoot_fixed Whether intake equals demand regardless of supply (compound feed ingredients that are not substitutable).

Source

afsetools `Codes_coefs.xlsx`.

`fertiliser_n2o_modifiers` *Fertiliser-type modifying factors for direct soil N2O.*

Description

Multiplicative modifying factors applied to the disaggregated direct N2O emission factor by fertiliser type and climate zone. They scale the climate-by-irrigation emission factor up or down according to the nitrogen source. Mediterranean factors follow the Cayuela et al. (2017) meta-analysis for synthetic, solid and liquid sources and the IPCC (2019) dry-area values for excreta; Atlantic factors follow the IPCC (2019) wet-area values. Missing factors mark sources whose N2O is not modelled through this pathway in the Mediterranean (recycled organic fertilisers, soil organic matter, human-population N).

Usage

```
fertiliser_n2o_modifiers
```

Format

A tibble with columns:

fert_type Nitrogen source: one of `"Synthetic"`, `"Solid"`, `"Liquid"`, `"Recycling"`, `"Excreta_cattle_monog"`, `"Excreta_other"`, `"SOM"`, `"Human"` (the human-population N term, keyed `"Urban"` before it was renamed; the former key is still read, with a deprecation warning).

climate Climate zone: `"MED"` or `"ATL"`.

mf Multiplicative modifying factor on the N2O emission factor; `NA` where the source is not modelled through this pathway.

source Provenance note transcribed from the coefficient workbook (Cayuela et al. 2017 or IPCC 2019).

Source

Cayuela, M. L. et al. (2017). Direct nitrous oxide emissions in Mediterranean climate cropping systems. *Agriculture, Ecosystems & Environment*, 238, 25-35. doi:10.1016/j.agee.2016.10.006. IPCC (2019), 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol. 4, Chapter 11.

Examples

```
fertiliser_n2o_modifiers
```

`fill_linear` *Fill gaps by linear interpolation, or carrying forward or backward.*

Description

Fills gaps (`NA` values) in a time-dependent variable by linear interpolation between two points, or carrying forward or backwards the last or initial values, respectively. It also creates a new variable indicating the source of the filled values.

Usage

```
fill_linear(
  data,
  value_col,
  time_col = year,
  interpolate = TRUE,
  log_space = FALSE,
  fill_forward = TRUE,
  fill_backward = TRUE,
  value_smooth_window = NULL,
  .by = NULL,
  .copy = TRUE
)
```

Arguments

`data`	A data frame containing one observation per row.
`value_col`	The column containing gaps to be filled.
`time_col`	The column containing time values. Default: `year`.
`interpolate`	Logical. If `TRUE` (default), performs linear interpolation.
`log_space`	Logical. If `TRUE`, interior interpolation is performed in log space (constant compound growth rate) for each gap segment whose two bracketing anchors are both finite and strictly positive; any segment with a non-positive or non-finite anchor falls back to linear interpolation. Default: `FALSE`, i.e. linear interpolation everywhere. Log-space fills are labelled `"Log-linear interpolation"` in the source column, distinct from `"Linear interpolation"`, so the choice survives downstream. Carrying forward or backward is unaffected.
`fill_forward`	Logical. If `TRUE` (default), carries last value forward.
`fill_backward`	Logical. If `TRUE` (default), carries first value backward.
`value_smooth_window`	An integer specifying the window size for a centered moving average applied to the variable before gap-filling. Useful for variables with high inter-annual variability. If `NULL` (default), no smoothing is applied.
`.by`	A character vector with the grouping variables (optional).

`.copy` Logical. If `TRUE` (default), data.table inputs are defensively copied before mutation. Set to `FALSE` when the caller owns the data and does not need the original preserved.

Value

A tibble data frame (ungrouped) where gaps in value_col have been filled, and a new "source" variable has been created indicating if the value is original or, in case it has been estimated, the gapfilling method that has been used. Rows come back sorted by `.by` and then `time_col`, which is the order the filling is defined in: carrying a value forward or backward, and the moving average behind `value_smooth_window`, all read neighbouring rows, so the result cannot depend on the order the rows were handed over in.

Examples

```
sample_tibble <- tibble::tibble(
  category = c("a", "a", "a", "a", "a", "a", "b", "b", "b", "b", "b", "b"),
  year = c(
    "2015", "2016", "2017", "2018", "2019", "2020",
    "2015", "2016", "2017", "2018", "2019", "2020"
  ),
  value = c(NA, 3, NA, NA, 0, NA, 1, NA, NA, NA, 5, NA),
)
fill_linear(sample_tibble, value, .by = c("category"))
fill_linear(
  sample_tibble,
  value,
  interpolate = FALSE,
  .by = c("category"),
)
fill_linear(sample_tibble, value, log_space = TRUE, .by = c("category"))
```

`fill_proxy_growth` *Fill gaps using growth rates from proxy variables*

Description

Fills missing values using growth rates from a proxy variable (reference series). Supports regional aggregations, weighting, and linear interpolation for small gaps.

Usage

```
fill_proxy_growth(
  data,
  value_col,
  proxy_col,
  time_col = year,
  .by = NULL,
  max_gap = Inf,
```

```
  max_gap_linear = 3,
  fill_scope = NULL,
  value_smooth_window = NULL,
  proxy_smooth_window = 1,
  output_format = "clean",
  verbose = TRUE
)
```

Arguments

`data`	A data frame containing time series data.
`value_col`	The column containing values to fill.
`proxy_col`	Character or vector. Proxy variable(s) for calculating growth rates. Supports multiple syntax formats: • **Simple numeric proxy** (e.g., `"population"`): Auto-detects numeric columns and uses them as proxy variable. Inherits the `.by` parameter to compute proxy values per group. • **Simple categorical proxy** (e.g., `"region"`): Auto-detects categorical columns and interprets as `value_col:region`. Aggregates `value_col` by the specified groups. • **Advanced syntax** (e.g., `"gdp:region"`): Format is `"variable:group1+group2"`. Aggregates variable by specified groups. • **Hierarchical fallback** (e.g., `c("population", "gdp:region")`): Tries first proxy, falls back to second if first fails. • **Weighted aggregation** (e.g., `"gdp[population]"`): Weight variable by specified column during aggregation.
`time_col`	The column containing time values. Default: `year`.
`.by`	A character vector with the grouping variables (optional).
`max_gap`	Numeric. Maximum gap size to fill using growth method. Default: Inf.
`max_gap_linear`	Numeric. Maximum gap size for linear interpolation fallback. Default: 3.
`fill_scope`	Quosure. Filter expression to limit filling scope. Default: NULL.
`value_smooth_window`	Integer. Window size for a centered moving average applied to the value column before gap-filling. Useful for variables with high inter-annual variability. If `NULL` (default), no smoothing is applied.
`proxy_smooth_window`	Integer. Window size for moving average smoothing of proxy reference values before computing growth rates. Default: 1.
`output_format`	Character. Output format: ”clean” or ”detailed”. Default: ”clean”.
`verbose`	Logical. Print progress messages. Default: TRUE.

Details

Combined Growth Sequence (Hierarchical Interpolation):

When using multiple proxies with hierarchical fallback, the function implements an intelligent combined growth sequence strategy:

1. Better proxies (earlier in hierarchy) are tried first for each gap.
2. If a better proxy has partial coverage within a gap, those growth rates are used for the covered positions.
3. Fallback proxies fill only the remaining positions where better proxies are not available.
4. Values filled by better proxies are protected from being overwritten.

Value

A data frame with filled values. If output_format = "clean", returns original columns with updated value_col and added source column. If "detailed", includes all intermediate columns.

See Also

`fill_linear()`, `fill_sum()`

Examples

```
# Fill GDP using population as proxy
data <- tibble::tibble(
  country = rep("ESP", 4),
  year = 2010:2013,
  gdp = c(1000, NA, NA, 1200),
  population = c(46, 46.5, 47, 47.5)
)

fill_proxy_growth(
  data,
  value_col = gdp,
  proxy_col = "population",
  .by = "country"
)
```

`fill_sum` *Fill gaps summing the previous value of a variable to the value of another variable.*

Description

Fills gaps in a variable with the sum of its previous value and the value of another variable. When a gap has multiple observations, the values are accumulated along the series. When there is a gap at the start of the series, it can either remain unfilled or assume an invisible 0 value before the first observation and start filling with cumulative sum.

Usage

```
fill_sum(
  data,
  value_col,
  change_col,
  time_col = year,
  start_with_zero = TRUE,
  .by = NULL
)
```

Arguments

`data`	A data frame containing one observation per row.
`value_col`	The column containing gaps to be filled.
`change_col`	The column whose values will be used to fill the gaps.
`time_col`	The column containing time values. Default: `year`.
`start_with_zero`	Logical. If TRUE (default), assumes an invisible 0 value before the first observation and fills with cumulative sum starting from the first change_col value. If FALSE, starting NA values remain unfilled.
`.by`	A character vector with the grouping variables (optional).

Value

A tibble dataframe (ungrouped) where gaps in value_col have been filled, and a new "source" variable has been created indicating if the value is original or, in case it has been estimated, the gapfilling method that has been used.

Examples

```
sample_tibble <- tibble::tibble(
  category = c("a", "a", "a", "a", "a", "a", "b", "b", "b", "b", "b", "b"),
  year = c(
    "2015", "2016", "2017", "2018", "2019", "2020",
    "2015", "2016", "2017", "2018", "2019", "2020"
  ),
  value = c(NA, 3, NA, NA, 0, NA, 1, NA, NA, NA, 5, NA),
  change_variable = c(1, 2, 3, 4, 1, 1, 0, 0, 0, 0, 0, 1)
)
fill_sum(
  sample_tibble,
  value,
  change_variable,
  start_with_zero = FALSE,
  .by = c("category")
)
fill_sum(
  sample_tibble,
  value,
```

```
  change_variable,
  start_with_zero = TRUE,
  .by = c("category")
)
```

`folded_reporting_areas` *List the reporting areas whose data is folded into another area code*

Description

A FAOSTAT reporting area is *folded* when polity_area_crosswalk gives it a `polity_area_code` that is not its own `area_code`. Every row the area reports is then summed into that other bucket, so the area disappears from WHEP output without a single row being dropped or left unresolved. This lists those areas, because the coverage reports cannot: a fold resolves perfectly well, only to a territory that did not report the data.

Three kinds exist, and they are not equally defensible:

- `"fabio_rest_of_world"`: FABIO collapses the area into its single Rest-of-World row (`polity_area_code` 999, `ROW-1850-2025`) because its own region list does not enumerate the area either. 57 areas, all flagged `cbs FALSE` in regions_full. Several still report substantial data of their own – Reunion, Guadeloupe, Palestine, the Faroe Islands – which is attributed to Rest of World.
- `"cbs_reporter_folded"`: the area is flagged `cbs TRUE`, so regions_full says it has a commodity balance sheet of its own, and it is folded into 999 anyway. Four areas: 153 New Caledonia, 154 North Macedonia, 209 Eswatini and 212 Syria, the last being the largest single contributor to the fold. **FABIO does not fold these**: its published region list enumerates all four as regions in their own right (see the section below), so this fold is WHEP's, not a FABIO convention, and the `"fabio"` label the other 57 carry does not apply.
- `"successor_state"`: the area is summed into the bucket of the state that succeeded it, which is a deliberate territorial identity rather than a loss: FAOSTAT area 62 "Ethiopia PDR" into 238 Ethiopia, and areas 276 Sudan and 277 South Sudan into 206 Sudan (former).

Whether to lift the Rest-of-World fold is an open decision recorded in issue 419; this function only makes the current state visible and changes nothing. A build also warns, naming the areas and the row counts it actually folded.

Usage

```
folded_reporting_areas(crosswalk = NULL)
```

Arguments

`crosswalk`	Crosswalk to inspect. Defaults to polity_area_crosswalk.

Value

A tibble with one row per folded reporting area, ordered by `area_code`:

- `area_code`: The reporting area whose data is folded away.
- `area_name`, `area_iso3c`: Its name and ISO3-like code.
- `polity_area_code`: The bucket its rows are summed into.
- `polity_code`, `polity_name`: The polity the fold attributes them to.
- `fold_kind`: `"fabio_rest_of_world"`, `"cbs_reporter_folded"`, `"predecessor_bucket"` or `"successor_state"`.

What FABIO's own region list says

FABIO (Bruckner et al. 2019) publishes the region list it uses, and it contains all four `"cbs_reporter_folded"` areas as regions of their own:

- `io_codes.csv` of the FABIO v1.1 release (Zenodo record 2577067, the file `inst/scripts/compare_fabio.R` already downloads) enumerates 192 areas x 125 commodities. Areas 153, 154, 209 and 212 each have their own 125-row block, distinct from area 999 `RoW`.
- The FABIO source repository (https://github.com/fineprint-global/fabio) folds an area into Rest of World exactly when it is absent from `inst/regions_full.csv` with `current == TRUE`. All four carry `current TRUE` there, and the 192 codes that file flags `cbs TRUE` are precisely the 192 areas of `io_codes.csv`.

So `fabio_code == 999` for these four is a statement WHEP makes, not one FABIO makes. Correcting it in `regions_full` would move published values, because `polity_area_code` is derived from `fabio_code`, so the contradiction is left standing and reported here instead (issue 556).

The Rest-of-World fold is no longer applied

WHEP models every reporting member of bucket 999 in its own right. FABIO's 192-country layout is a methodology this package compares against, not a constraint on which territories it represents, and the choice of country set is WHEP's to make (issue 459).

That matters because the fold was never doing what its name suggests. Of the 61 members, only about a third report anything at all; the rest contribute no rows and folding them is arithmetically a no-op. Everything the bucket actually carried came from the members that DO file returns – Syria, Eswatini, North Macedonia, New Caledonia, the Faroe Islands, Palestine, Greenland and the like – and folding them discarded whose data it was. So promotion is self-limiting: an area with no rows is unaffected either way.

Bucket 999 survives as a genuine residual for the territories that report nothing. Measured on a full-range `get_wide_cbs()` (1850-2023), promotion takes the published area count from 195 to 216 and moves global totals by at most 0.99% (`stock_addition`), with every other column inside 0.4%.

`options(whep.unfold_rest_of_world = "none")` restores the fold, which is what reproducing a number published before this change requires. Because that no longer matches the published series, every read of the crosswalk warns while it is set. `"cbs_reporters"` re-folds all but the four `"cbs_reporter_folded"` areas and warns for the same reason. The

`"successor_state"` folds are never lifted by any mode, since those are territorial identities rather than a FABIO convention.

The predecessor-bucket fold, and the switch that lifts it

`"predecessor_bucket"` is the other direction and a separate switch. FAOSTAT retired area 206 "Sudan (former)" at the 2011 secession and reports 276 Sudan and 277 South Sudan from 2012, so here the *bucket* is the dead code and its members are live – the reverse of 62 Ethiopia PDR folding into its live successor 238, which stays a `"successor_state"` fold and is never lifted.

WHEP publishes the fold. `options(whep.unfold_predecessor_bucket = "all")` promotes the successors and warns on every crosswalk read, because unlike the Rest-of-World promotion it is **not** mass-neutral. Measured on `build_primary_production(2015, 2015)` it moves nothing at all outside areas 206, 276 and 277 – 48,678 rows, no key present in only one run, no matched value differing – and inside the region it withdraws exactly one series: item 651 Forage products, 1,432,940 t (4.22% of the region's tonnage) and 208,350 ha (0.25%). Heads, livestock units and slaughtered heads are conserved; item 1052 Chickens, layers splits exactly, 9,439,000 head to Sudan and 4,679,716 to South Sudan.

That tonnage is not reported data that stops being joined. It is `DM_yield_estimate_carried_forward`, WHEP's own extrapolation of the `faostat-production-old` fodder series for area 206, a source that carries no row for 276 or 277 in any year; the fold is what keeps bucket 206 a live key to extrapolate onto after FAOSTAT retired it in 2011. Restoring it under the promotion would need a rule for apportioning a predecessor's series between its successors, which this package has no source for. Whether to publish the promotion is issue 680.

An earlier measurement recorded in issue 419 reported this change at up to 13.7x on `feed`; that comparison predates the `dcast()` duplicate-key fix in `.select_best_source()` (issue 425) and does not reproduce.

References

Bruckner, M., Wood, R., Moran, D., Kuschnig, N., Wieland, H., Maus, V., Borner, J. (2019). FABIO - The Construction of the Food and Agriculture Input-Output Model. Environmental Science & Technology 53(19), 11302-11312. doi:10.1021/acs.est.9b03554

Examples

```
folded <- folded_reporting_areas()
nrow(folded)
head(folded[folded$fold_kind == "successor_state", ], 4)
folded[folded$fold_kind == "predecessor_bucket", ]
```

`footprint_scope` *Describe the scope of a footprint result.*

Description

Build a machine-readable scope record for a footprint: what is measured, in which units, by which method, and under which system boundary, allocation rule, data vintage and known limitations. This is the ISO 14044 goal-and-scope made into an attachable data object rather than prose, so a footprint result always travels with the assumptions behind it.

Attach it to a result with `attach_scope()` and read it back with `get_scope()`.

Usage

```
footprint_scope(stressor, units, method, details = list())
```

Arguments

`stressor`	What is measured, e.g. `"cropland"`.
`units`	Units of the footprint value, e.g. `"ha"`.
`method`	Estimation method used, e.g. `"FABIO-MRIO"`. This mirrors the multi-method `method_<quantity>` columns recorded elsewhere in the package.
`details`	Optional named list overriding any of: `boundary` (system boundary), `allocation` (allocation rule), `vintage` (data years) and `limitations` (free text).

Value

A one-row tibble with columns `stressor`, `units`, `method`, `boundary`, `allocation`, `vintage` and `limitations`.

Examples

```
footprint_scope(
  stressor = "cropland",
  units = "ha",
  method = "FABIO-MRIO",
  details = list(vintage = "1850-2023", allocation = "mass")
)
```

---

footprint_sensitivity

Local sensitivity of a footprint to each extension.

Description

One-at-a-time sensitivity analysis: nudge each sector's extension by a small relative step and measure the elasticity of the total footprint. High-elasticity sectors are where data quality matters most and where a result is most fragile.

Usage

```
footprint_sensitivity(run_fn, extensions, options = list())
```

Arguments

`run_fn`	Function taking an extension vector and returning a footprint tibble with a `value` column.
`extensions`	Numeric vector of base extensions per sector.
`options`	Named list overriding `delta` (relative step, default 0.05) and `which` (sector indices to test; default all non-zero extensions).

Value

A tibble with `sector` (index into `extensions`) and `elasticity`, ordered by descending absolute elasticity.

Examples

```
run_fn <- function(ext) tibble::tibble(value = sum(ext))
footprint_sensitivity(run_fn, extensions = c(60, 40))
```

---

get_arable_permanent_land

Physical arable and permanent-crop land base (fallow-inclusive).

Description

Return FAO's physical land-use split of cropland into **arable land** (annual/temporary crops plus their rotational fallow and temporary meadows) and **permanent-crop land** (orchards, plantations, vineyards), keyed by `(area_code, year)`.

whep's other crop-area paths (`get_crop_land_extension()`, `build_cropgrids_land_extension()`) are all derived from crop *production* / harvested area and therefore cannot recover the physical fallow-inclusive arable land of rain-fed, fallow-prone economies: in a drought year a country's cereal harvest collapses while its arable land (which counts the resting fallow) is unchanged, so a harvested-area method assigns that land to perennials and over-states the permanent share (e.g. Tunisia 2020 permanent share 0.73 from harvested area vs 0.43 physical). FAO's RL land-use survey (`Cropland` = `Arable land` + `Permanent crops`) is the physical land base; this function ingests it.

From 1961 the split is FAO's own (`source == "fao"`). Before 1961 (FAOSTAT's start) it is backcast from LUH2 land use: LUH2's annual vs. perennial crop functional types give a perennial fraction and a cropland shape that are spliced onto the FAO 1961 level so the series is continuous (`source == "luh2"`). See Details.

Usage

```
get_arable_permanent_land(
  years = NULL,
  input_dir = NULL,
  data = NULL,
  luh2_data = NULL,
  example = FALSE
)
```

Arguments

`years`	Integer vector of years to return, or `NULL` (default) for all available (1700-2025). The pre-1961 LUH2 backcast is computed only when `years` is `NULL` or requests a year before 1961.
`input_dir`	Optional directory holding a local FAOSTAT RL land-use file (`faostat_land_use.csv` or a parquet with the FAOSTAT RL columns). If `NULL` (default) the pinned `faostat-landuse` dataset is read via `whep_read_file()`.
`data`	Optional in-memory FAOSTAT RL table in the raw pin schema (columns `Area Code`, `Item Code`, `Element`, `Unit`, `Year`, `Value`), used instead of the pin (chiefly for testing). The two vocabularies this read selects on – `Element == "Area"` and the `"1000 ha"` unit label – are checked before the filter runs: either one moving takes every row with it, and a zero-row land base is indistinguishable downstream from a world with no cropland, so it is refused with a `whep_absent_label` error naming the labels the table does carry (#1034).
`luh2_data`	Optional in-memory LUH2 land-use table (columns `ISO3`, `Year`, `Land_Use`, `Area_Mha`) used for the pre-1961 backcast instead of the pinned `luh2-areas` dataset (chiefly for testing).
`example`	If `TRUE`, return a small illustrative table without reading remote data. Defaults to `FALSE`.

Details

The FAO identity `Cropland = Arable land + Permanent crops` holds in the source to rounding for essentially all country-years; `permanent_ha` is taken as `Cropland - Arable land` (clamped at 0) so `arable_ha + permanent_ha` reconstructs `cropland_ha` exactly wherever FAO reports `Arable <= Cropland`. Where FAO reports `Arable land` but not `Permanent crops` (924 country-years, mostly arable-only economies) this yields the permanent land the survey implies; where it reports `Permanent crops` but not `Arable land` (a few coconut atolls) `arable_ha` is filled from `Cropland - Permanent crops`.

Pre-1961 backcast: LUH2 annual cropland is `c3ann + c4ann + c3nfx`, perennial is `c3per + c4per`. For each country the perennial fraction and the cropland level are rescaled by their ratio to the LUH2 value at 1961 and multiplied by the FAO 1961 perennial fraction and cropland, so both match FAO exactly at the 1961 splice point and carry LUH2's earlier dynamics backwards. Countries without a FAO 1961 anchor receive no backcast.

Value

A tibble with one row per `(area_code, year)`:

- `area_code`: integer FAOSTAT area code, harmonised onto the `polity_area_code` bucket the rest of the pipeline aggregates on, at the fold state `options(whep.unfold_rest_of_world)` selects (the FAOSTAT "China" aggregate 351 is dropped). See polity_area_crosswalk.
- `year`: integer.
- `arable_ha`, `permanent_ha`, `cropland_ha`: physical land area in hectares.
- `source`: provenance, `"fao"` (>= 1961) or `"luh2"` (pre-1961 backcast).

Plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
get_arable_permanent_land(example = TRUE)
```

---

get_bilateral_trade *Bilateral trade data*

Description

Reports trade between pairs of countries in given years.

Usage

```
get_bilateral_trade(
  example = FALSE,
  cbs = NULL,
  method_items_not_in_cbs = c("drop", "keep", "abort"),
  method_seed_unit = c("target", "tonnes")
)
```

Arguments

`example` If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

`cbs` Optional pre-computed wide CBS tibble from `get_wide_cbs()`. If `NULL` (default), it is built internally.

`method_items_not_in_cbs`
How to treat a traded item whose `item_cbs_code` has no commodity balance sheet row to balance it against. See the *Items with no CBS row* section. One of:

- `"drop"` (default): discard those flows, the historical behaviour. Published values are unaffected by this argument's existence as long as the default is kept.
- `"keep"`: keep the flows and take the row and column margins from the reported bilateral data itself instead of from the CBS.
- `"abort"`: fail, so that a refreshed pin cannot introduce unanchored items unnoticed.

Under **every** method, an item that survives this step whose tonnes are not masses is refused; see the *Items with no CBS row* section.

`example = TRUE` always returns the `"drop"` fixture.

`method_seed_unit`
Which reported bilateral rows seed each trade matrix before it is balanced. See the *Which unit seeds a matrix* section. One of:

- `"target"` (default): seed each (year, item) matrix from the rows reported in the unit of the CBS margins it is balanced onto, i.e. head counts for the live-animal rows of the livestock balance and tonnes for everything else.
- `"tonnes"`: seed every matrix from its tonnes rows only, the behaviour before whep#1031. A live-animal (year, item) with no tonnes rows gets no matrix at all.

`example = TRUE` always returns the `"target"` fixture.

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 from country `"A"` to country `"B"`, for the corresponding year and item, in the row's `unit` (tonnes, or head counts for live animals). 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.
- `has_cbs_totals`: `TRUE` when the matrix margins came from the commodity balance sheet, `FALSE` when the item had no CBS row for that year and its margins were taken from the reported bilateral flows themselves. Always `TRUE` unless `method_items_not_in_cbs = "keep"`.
- `method_items_not_in_cbs`: the treatment chosen for items with no CBS row, recorded so a downstream consumer can tell which variant it is holding.
- `unit`: the denomination of the matrix values, taken from the CBS margins it was balanced onto: `"tonnes"`, or `"heads"` for the live animals. A matrix balanced onto its own reported flows (`has_cbs_totals == FALSE`) is seeded from, and so denominated in, tonnes.
- `method_seed_unit`: the seed-unit method chosen, see `method_seed_unit`.

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.

Which unit seeds a matrix

Each matrix is balanced onto the CBS `export`/`import` margins of its (year, item), so it is denominated in the `unit` of those margins: tonnes for the FAO-style balance and **head counts** for the live-animal rows that `get_livestock_cbs()` adds, whose margins are summed from exactly the `Head` rows of the same bilateral data. The seed only supplies the partner structure; iterative proportional fitting imposes the level. The choice is therefore which observed rows supply that structure (whep#1031).

`"tonnes"` keeps the mass rows alone. On the `bilateral_trade` pin `20250714T123347Z-2c392`, FAOSTAT reports the live-animal items in `Head` only up to 2013, so 283 of the pin's 4,589

(year, item) groups, over 11 items and 1986-2013, have no tonnes row. Such a group has no seed cell at all, so it produces **no matrix**, and `build_io_model()` then treats the item as wholly domestically sourced (`.item_trade_shares()` returns the identity for an item with no matrix). In 2010 that is all 9 live-animal items with CBS trade: 62.9 million imported head with no country of origin. Over 2014-2021 a further 86 groups carry both units; their structure then comes from the tonnes rows while their margins are head counts.

`"target"` seeds each matrix in its margins' own unit, so seed and target agree and the observed head-count partner structure is kept. Measured on that pin against the 2010 and 2016 wide CBS:

- every tonnes-denominated matrix (110 in 2010, 111 in 2016) is identical under both methods;
- 2010: `"target"` returns 9 head-count matrices where `"tonnes"` returns none. Their export-weighted partner shares differ from the reported head flows by a total-variation distance of 0 to 0.096 (0.0001 for pigs, 0.0006 for sheep).
- 2016: both return the 10 head-count matrices with non-zero margins. The export-weighted total-variation distance between the two methods' partner shares is 0.11 (camels) to 0.82 (mules), 0.49 for pigs and 0.70 for sheep, and the `"tonnes"` shares sit that far from the reported head flows while the `"target"` ones sit within 0.056.

`"tonnes"` also returns 5 more 2016 matrices, all zero: chickens, turkeys, rabbits, geese and rodents, whose CBS head margins are zero because that pin carries their `1000 Head` rows nowhere (whep#1092). An all-zero matrix carries no partner structure either, so `"target"` drops them as seedless (with a warning) and nothing observed is lost.

The pin registered since whep#1122, `20261001T135651Z-ca7b3`, carries those rows. Against a 2009–2011 `get_livestock_cbs()`, `"target"` then returns 14 head-count matrices for 2010 instead of 9, holding 1,544,369,485 head instead of 62,946,485: the five added are chickens (1,385,753,000), turkeys (76,646,000), ducks (13,015,000), rabbits (4,791,000) and geese (1,218,000), and the nine already there are unchanged.

The seed is not only a structure. `.fill_missing_trade()` fills each unreported cell from the margins, capped by the gap between an exporter's margin and its reported row sum, and the balancing step then starts from reported and estimated cells together. Both comparisons are made in the margins' unit, so a tonnes seed under head-count margins leaves most of each gap unfilled by what was reported and lets the estimate outweigh it. In 2016 the pin's summed live-animal head counts exceed its summed tonnes by 4.6 times for non-dairy cattle and 31 times for sheep, consistent with how far the 2016 `"tonnes"` shares sit from the reported flows.

Converting head counts to mass is deliberately not offered. It would put the seed in tonnes under head-count margins, the mismatch just described, unless the margins were converted as well, and converting them needs a sourced live weight per species. The package holds none: `gleam_animal_weights` is an unsourced placeholder (whep#881, whep#182).

Items with no CBS row

The bilateral trade matrices are balanced against the total exports and imports reported in the commodity balance sheet, so an item with no CBS supply/use row has nothing to balance against. Historically those flows were discarded silently. Measured on the

`bilateral_trade` pin `20250714T123347Z-2c392` (after the export-preference deduplication and the tonnes filter, i.e. exactly what reaches this step): 12.51 Gt of 47.48 Gt, **26.4% of the traded tonnage over 1986-2021**, in 7 items, of which 99.1% is the FABIO-style aggregate placeholder `"Other"` (`item_cbs_code` 5001). The drop is now reported with `cli::cli_warn()` whichever method is chosen, because a quarter of world trade should not disappear without a message.

The share is far from constant: 9.1% over 1986-2003, **49.6% over 2004-2013** and 4.3% over 2014-2021. The middle block is not a real trade signal. It contains physically impossible flows booked in tonnes: Colombia to the United States, 2004, 2.58 Gt of `"Other"` in a single cell, more than world cereal production; Kenya to the Netherlands, 515-594 Mt/year over 2005-2009. Excluding item 5001 altogether, the whole drop is 112 Mt, 0.24% of traded tonnage.

Which treatment is right is therefore a methodological question, not a lookup: `"Other"` is an unallocated residual whose 2004-2013 values are demonstrably corrupt, so `"keep"` carries that corruption into the output, and mapping the residual onto real CBS items would need a sourced disaggregation key that does not exist in the package. Nothing downstream consumes the kept rows yet either: `build_io_model()` takes its item dimension from supply-use and the CBS, so an item absent from both is ignored by `.build_trade_shares()` regardless of this argument.

Those figures are not masses, and that has now been traced to the FAOSTAT source (whep#1023). CBS item 5001 is fed by FAOSTAT trade item 1293 (*Crude organic material n.e.c.*), for which FAOSTAT's aggregate *Trade: Crops and livestock products* domain publishes a value but no country-level mass, while its Detailed Trade Matrix reports tonnages worth USD 0.01-0.5 per tonne whose mirrored report of the same flow disagrees by factors of 356 to 838,000. What the large side counts is unverified and the implied units per tonne are not constant, so nothing can be rescaled. See `build_detailed_trade()`'s *Quantities FAOSTAT does not back with a mass* section for the full measurement.

This function therefore **aborts** with class `"whep_unbacked_mass_trade"` whenever such an item would survive this step, rather than distributing 2.58 Gt through a matrix. The test is on what is *kept*, under every method, and not on whether the item has a CBS row: those two coincide on today's data, and that coincidence was the only thing keeping the figures out of a published number. Give item 5001 a CBS row and the earlier, `"keep"`-only refusal let the whole 12.40 Gt through on the default method, silently.

No published number moves. Measured on the 2025-07-14 `bilateral_trade` pin `20250714T123347Z-2c392`, item 5001 carries 12.40 Gt of `tonnes` over 277,201 rows - 8.9% of 1986-2003, **49.4% of 2004-2013** and 3.9% of 2014-2021 - and no commodity balance sheet carries item 5001, so `"drop"` removes all of it exactly as before.

A caller who wants a trade matrix that carries item 5001 has to obtain a mass for it first. `build_detailed_trade()` screens the same rows at the producer, where the fix belongs, and the `bilateral_trade` pin has been regenerated from it (whep#1122): the registered version carries no item 5001 row at all, so this refusal is now a guard against a pin built some other way.

The batch pin on the build path

Two further aliases were published in that same 2025-07-14 batch — `crop_residues` and `bilateral_trade`, alongside the four above, between 12:33:43Z and 12:33:50Z. Unlike the

four, these two are read on the default build path, which is why no warning is attached to them: every commodity balance build reads them, so a warning at the read is noise rather than information. `bilateral_trade` has since been regenerated by this package (#1122), so `crop_residues` is the only one still registered at its 2025-07-14 version.

They did **not** share a provenance despite sharing a timestamp. Each was established separately (#1054), because the timestamp alone establishes nothing.

`crop_residues` is predecessor output:

Read by `get_primary_residues()`, and from there by `build_commodity_balances()`. All 475,688 of its `Product` rows equal the `primary_prod` pin's `tonnes` values exactly — no key unmatched on either side, no value differing at a relative tolerance of 1e-6 — so it is a downstream artifact of the same predecessor run, carrying that run's production series into the commodity balance.

Its residue quantities are those production numbers times a residue-to-product ratio that varies by year, and times a **recovery rate**: the pin holds the residue the predecessor recovered from the field, not the residue produced (#1195). The predecessor's code (`Global/R/crop_npp.r`, then `afsetools::residue_use()`) writes each row as

```
product * kg_residue_kg_product_FM * HI_changes(region, year) * residue_dm_product_dm(region, category) / residue_dm_product_dm("West Europe", category) * recovery_rates(region, category),
```

with region and category keyed as in `residue_recovery.csv`. Rebuilt from the predecessor's own coefficient tables, that formula reproduces all 425,934 pin rows whose region it can key, at a relative tolerance of 1e-6. The year variation is `HI_changes` alone: a harvest-index change factor per region, given at eight anchor years from 1910 to 2000, linear in between and flat after 2000 (East Europe 1.10 in 1962 to 1.00 in 2000), which the predecessor attributes to Krausmann et al. (2013), *PNAS* 110:10324, Table M2 (assumed, unverified). It is a regional historical adjustment, not a yield response, and after 2000 the ratio of every region and category is constant. That is why `biomass_coefs$kg_residue_kg_product_FM` alone reproduces only 3,189 of the 472,790 comparable keys. Of the 33,411 keys that carry a residue of exactly 0, 30,620 are cells whose recovery rate is 0 (fodder crops in seven of the eight regions; roots and tubers, cassava, sugar beet and dry beans in West Europe and in North America and Oceania) and 1,919 are crops whose `biomass_coefs` ratio is 0.

WHEP ships the recovery rate and the regional ratio the predecessor used (both equal in all 160 cells), but not the `HI_changes` table. So `get_primary_residues()` undoes the recovery exactly, dividing by the same rate, but cannot recompute the predecessor's residue from production. It then re-keys the regional ratio on Wirsenius's own region membership, and estimates cereal residue (#1448), and that of the non-cereal crops with their own IPCC Table 11.2 line (#1518), from the pin's production and harvested area instead of reading it.

It is not a small input. The pin holds 7.63 Gt of recovered residue for 2010 (Straw 3.60 Gt, Other crop residues 2.49 Gt, Firewood 1.54 Gt) and 327.7 Gt over 1961–2021, which `get_primary_residues()` turns into 7.64 Gt (Straw 3.92 Gt, Other crop residues 2.82 Gt, Firewood 0.90 Gt) and 366 Gt of residue produced. 3,998 of its 249,095 output rows carry `NA` polity columns because the pin is name-keyed.

Driving the same residue model off a fresh `get_primary_production()` would move those numbers. Measured for 2010 at (`area_code`, `item_prod`): 1.958 Gt of current production sits on 6,425 keys the pin never sees (809 Mt of it primary crops, including the modelled

temporary-grassland item), 64.8 Mt on 296 keys is pin-only, and of the 8,010 shared keys 538 disagree, putting 1.102 Gt — 11.01% of the pin's shared-key mass — more than 1% apart. The largest class is rice, 673.9 Mt in the pin against 463.3 Mt fresh, a ratio of 0.6876: current code puts rice on a milled-equivalent basis while the pin's is paddy, which is a basis difference rather than an error, since straw scales with the field crop. The second is fodder, where the current build is close to twice the pin on every forage and silage item, the pin predating that work. Replacing the pin therefore means choosing a residue model, which is a science decision and not a refresh. For cereals that model is chosen by `get_primary_residues(cereal_residue = )`, and for the non-cereals with an IPCC line by `non_cereal_residue =`; both still run on the pin's production.

`bilateral_trade` is built by `build_detailed_trade()`:

Read by `get_bilateral_trade()` and by the live-animal branch of `build_commodity_balances()`. The registered version, `20261001T135651Z-ca7b3`, is `build_detailed_trade()`'s output verbatim, with its defaults and `extend_time = FALSE`, run on the `faostat-trade-bilateral` pin `20260407T095142Z-b3f81` (#1122): 10,038,220 rows over 1986–2021, polity-keyed, carrying 64,450,133,319 tonnes and 87,840,869,640 head.

The 2025-07-14 version it replaces, `20250714T123347Z-2c392`, held the same FAOSTAT Detailed Trade Matrix values in the predecessor pipeline's name-keyed schema — against the raw pin over 2010, all 296,642 shared `tonnes` keys and all 5,102 shared `Head` keys agree exactly — but without FAOSTAT's `1000 Head` rows: 76,141,882 thousand head of live broiler chicken, turkey, duck, rabbit and goose trade over 1986–2021, which PR #1113 recovered in the producer. The regenerated pin carries 7.50x its 11,707,083,640 head and 22.5% less than its 83,147,095,412 tonnes, all of that tonnage difference in CBS items 5001-5003, which no commodity balance carries: item 1293's unbacked tonnage, screened by `method_unbacked_quantity` (#1023), and three CBS names with no `item_cbs_code`. On the tonnes that do reach a trade matrix the two versions differ by 0.056%. `.clean_bilateral_trade()` reads either schema.

Examples

```
get_bilateral_trade(example = TRUE)
```

`get_crop_land_extension` *Get the per-crop physical cropland extension from spatialization inputs.*

Description

Convenience wrapper that loads the gridded land-use inputs, spatializes crop harvested area with `build_gridded_landuse()` (crop-level, no CFT aggregation), and converts it to a per-crop physical land extension with `build_crop_land_extension()`. The result is keyed by `(year, area_code, item_cbs_code)` and ready to use as `extensions` in `compute_footprint()`.

Usage

```
get_crop_land_extension(
  input_dir = NULL,
  years = NULL,
  method = c("cropland_apportion", "intensity_divide"),
  use_type_constraint = FALSE,
  fill_missing_patterns = TRUE,
  example = FALSE
)
```

Arguments

`input_dir`	Directory holding the spatialization inputs (`country_areas.parquet`, `crop_patterns.parquet`, `gridded_cropland.parquet`, `country_grid.parquet`, and optionally `multicropping.parquet`). Typically `<l_files_dir>/whep/inputs`. If `NULL` or unset, the pinned WHEP spatialization inputs are used.
`years`	Numeric vector of years to compute, or `NULL` for all available.
`method`	Physical-area conversion method passed to `build_crop_land_extension()`.
`use_type_constraint`	If `TRUE`, restrict each crop to cells of its LUH2 type (requires `type_cropland.parquet`). Defaults to `FALSE`.
`fill_missing_patterns`	If `TRUE` (default), crops that have harvested area but no `crop_patterns` rows (e.g. Barley, absent from the Monfreda layer) are placed with a uniform fallback pattern over each producing country's cropland, so their land is not silently dropped.
`example`	If `TRUE`, return a small example output without reading remote/large data. Defaults to `FALSE`.

Value

A tibble with columns `year`, `area_code`, `item_cbs_code`, `impact_u` (physical land in hectares), and `method_land`, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.

- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
get_crop_land_extension(example = TRUE)
```

---

get_feed_intake *Livestock feed intake*

Description

Get amount of items used for feeding livestock.

Usage

```
get_feed_intake(
  example = FALSE,
  grain = c("national", "local"),
  demand_tier = c("ipcc", "fcr"),
  feed_mode = c("historical", "scenario"),
  years = NULL,
  feed_eligibility = c("feed_table", "none"),
  feed_loss = c("wirsenius", "flat", "none")
)
```

Arguments

`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.
`grain`	Spatial grain of the feed allocation. `"national"` (default, one allocation per country) or `"local"` (the per-cell 0.5-degree engine, which is heavy and run via `build_feed_intake_local()`; calling it here redirects there).
`demand_tier`	Demand-estimation tier. `"ipcc"` (default, the rigorous IPCC Tier-2 energy demand for the ruminant species it covers, Bouwman FCR for pigs and poultry, Krausmann per-head for draft / other species) or `"fcr"` (the Bouwman / Krausmann feed-conversion magnitude for every species). Both grains allocate with `redistribute_feed()`.
`feed_mode`	Whether to distribute surplus feed availability. `"historical"` (default) suppresses the surplus-distribution pass: the CBS feed element is treated as realised consumption, so leftover availability is not dumped onto variable-demand livestock (which would inflate non-grass intake). `"scenario"` distributes the surplus.
`years`	Integer vector of years to build, or `NULL` (default) for every year in the production data (1850-2023 via the LUH2 extension). Restricting the range cuts run time proportionally; allocation is independent per year, so a subset returns exactly the same rows for those years.
`feed_eligibility`	Which feeds each livestock category may receive. `"feed_table"` (default) follows `feed_taxonomy`: an item with a grazer feed type but no granivore feed type (straw and green fodder) is fibrous roughage that pigs and poultry do not eat, so it is never allocated to them, at any allocation level. Granivore demand that the feeds they may eat cannot cover stays unmet rather than being filled with roughage. `"none"` lets any category receive any item (the behaviour before whep#1218), for sensitivity analysis.
`feed_loss`	Share of the Commodity Balance Sheet `feed` element lost between distribution and intake. FAOSTAT books storage and transport losses in a separate element, so the `feed` element is already net of them. `"wirsenius"` (default) applies the per-class losses of Wirsenius (2000, sections 3.1.2 and 3.1.5): none on Food Balance Sheet products and processing by-products, 10% on crop residues (`Straw`, `Other crop residues`), 20% on harvested grass, legume and cereal fodder (hay-making and ensiling, harvest to intake) and 5% on fodder vegetables and roots; temporary grassland, booked as grazed grass, carries none. `"flat"` applies 10% to every item, the unsourced factor used before whep#1420. `"none"` takes the `feed` element as intake. Grass from the grassland sink and scavenging are not Commodity Balance Sheet items and carry no loss under any method.

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`.

This is not the input `estimate_n_excretion()` reads: that function takes the per-livestock-category rows of `redistribute_feed()` (`territory`, `livestock_category`, `feed_quality`, `intake_dm_t`), and refuses this table.

Examples

```
get_feed_intake(example = TRUE)
```

---

get_polity_geometries

Get WHEP polity geometries

Description

Returns the periodized polity database, including geometry. Pass `polity_codes` to retrieve a subset that can be joined to outputs from `add_polity_code()`.

Usage

```
get_polity_geometries(polity_codes = NULL)
```

Arguments

`polity_codes`	Optional character vector of WHEP polity codes. Subsetting by code needs the suggested package `sf` to be installed; the whole table is returned without it.

Value

An sf data frame.

Examples

```
# sf is only suggested, and its methods are what make the geometry column
# printable, so guard the example on it.
if (requireNamespace("sf", quietly = TRUE)) {
  codes <- add_polity_code(
    tibble::tibble(area_code = c(203L, 68L), year = c(2000L, 2000L))
  )$polity_code
  geometries <- get_polity_geometries(codes)
  print(geometries[, c("polity_code", "polity_name", "polygon_source")])
}
```

---

get_primary_production

Primary items production

Description

Get amount of crops, livestock and livestock products.

Usage

```
get_primary_production(years = NULL, example = FALSE)
```

Arguments

`years`	Optional integer vector of years to build. When `NULL` (default) the whole series is built. Supplying a window builds only that range rather than building 1850-2023 and discarding the rest, and caches it under a window-specific key. A window is not guaranteed to reproduce the full-range result value for value, because some steps look across the years present. Measured for 2010 against the full range, `area_code`-level totals agree exactly for `ha`, `t_ha`, `LU` and `heads`; the largest disagreement is 3.0e-04, in the livestock ratios (`t_head`, `t_LU`) and `slaughtered_heads`, tracked in issue #625. Use `NULL` when exact agreement with the published series matters.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

Value

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

- `year`: The year in which the recorded event occurred.
- `area_code`: Legacy numeric reporting area code.
- `polity_area_code`: Numeric WHEP reporting polity code used for matrix workflows. This currently matches `area_code`.
- `reporting_polity_code`: WHEP polity code for the reporting polygon.
- `reporting_polity_name`: WHEP polity name for the reporting polygon.
- `reporting_polity_has_geometry`: Whether the reporting polity has a polygon in the WHEP polity database.
- `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 (stocks), available for livestock.
  - `slaughtered_heads`: Number of animals slaughtered, 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`.
- `source`: Where the value came from, e.g. `"FAOSTAT_prod"`, `"EuropeAgriDB"`, `"LUH2_cropland"`, `"imputed_yield"`.
- `fao_flag`: FAOSTAT's observation-status code for the value (`"A"` official, `"E"` estimated, `"I"` imputed, `"M"`, `"X"`), or `NA` where the number is not one FAOSTAT published under a flag. It describes the value rather than the item or area, so it is `NA` on WHEP's computed yields, on the livestock-unit conversions, and on every gap-filled or back-cast row. See `build_primary_production()` for the full rule.

Examples

```
get_primary_production(example = TRUE)
```

`get_primary_residues` *Crop residue items*

Description

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

Usage

```
get_primary_residues(
  example = FALSE,
  cereal_residue = c("ipcc", "ensemble", "ratio", "wirsenius"),
  oil_palm_residue = c("per_product", "per_hectare", "pin"),
  non_cereal_residue = c("ipcc", "ensemble", "ratio", "pin")
)
```

Arguments

`example` If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

`cereal_residue`

How cereal residue is estimated. One of:

- `"ipcc"` (default): the IPCC yield-dependent line, above-ground residue dry matter = slope x grain dry-matter yield + intercept, through `calculate_crop_residues()` with its modern-variety correction.
- `"ensemble"`: the same estimator's mean of that line and the static biomass_coefs ratio, as the soil carbon and nitrogen chain uses it.
- `"ratio"`: the static biomass_coefs ratio alone.
- `"wirsenius"`: the predecessor's model, Wirsenius (2000) Table 3.16 regional ratios, applied directly on Wirsenius's own region membership.

See the "Cereal residue" section. Other crops are not affected.

`oil_palm_residue`

How oil palm residue (fronds and trunks) is obtained. One of:

- `"per_product"` (default): 60 Mt DM / 83.09 Mt of fresh fruit bunches = 0.722 t DM per tonne of fruit, Malaysia's 2010 fronds and trunks over its 2010 FAOSTAT production. It keeps the pin's production-times-ratio form and is the cautious rule where the per-hectare one is weakest: low-yield, semi-wild palm systems such as Nigeria's (2.6 t of fruit per hectare against Malaysia's ~17), which `"per_hectare"` would give several times the residue. West African semi-wild systems stay uncertain under either rule and would need their own source. Maintainer decision (whep#1424).

- `"per_hectare"`: 60 Mt DM / 4.85 Mha = 12.37 t DM per harvested hectare, the same 2010 anchor. It treats frond and trunk production as a property of the standing palm, and the rate barely moves between the strategy's two Malaysian years (12.26 t DM/ha at 2012, from its "over 83 million dry tonnes" times "about 75 percent" on FAOSTAT's 5.08 Mha), where the per-tonne ratio drifts (0.656 at 2012).
- `"pin"`: the pin's own oil palm residue, with the recovery rate undone and the ratio re-keyed like every other non-cereal crop.

See the "Oil palm residue" section.

`non_cereal_residue`

How the residue of a non-cereal crop with its own line in IPCC (2006) Table 11.2 is estimated: soybean, groundnut, dry beans, peas, chickpeas, lentils and the other pulses, potato, and, through IPCC's "Tubers" line, cassava, sweet potato, yams and the other roots. One of `"ipcc"` (default), `"ensemble"` and `"ratio"`, the same estimators as for `cereal_residue`, or `"pin"`, the pin's ratio as for every other crop. See the "Non-cereal residue" section.

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 – all of it, before any is recovered from the field – in tonnes of **fresh matter**, like every other commodity-balance quantity.
- `value_dm`: The same residue in tonnes of **dry matter**: each crop's fresh residue times its own residue dry-matter content, `Residue_kgDM_kgFM` in biomass_coefs, summed per row. `NA` where a crop with residue mass carries no such coefficient, so the gap stays visible rather than reading as zero.
- `method_residue`: how the row's residue was estimated: the `cereal_residue` method for a cereal, the `non_cereal_residue` method for a non-cereal crop with its own IPCC line, `"malaysia_nbs_per_product"` or `"malaysia_nbs_per_hectare"` for oil palm unless `oil_palm_residue` is `"pin"`, and `"pin"` for every other crop.

The pin's residue quantities are fresh matter. Across its crops the ratio of pinned residue to product tracks the fresh-matter residue:product ratio `kg_residue_kg_product_FM` of biomass_coefs (about 0.8 of it for nearly every crop), not the residue's dry-matter content, which runs from 0.13 (tomato) to 1.0 (rapeseed) (whep#1215). Use `value_dm` wherever a quantity is defined per unit of dry matter, such as a residue nitrogen content.

The pin does not hold the residue produced but the share of it recovered from the field: the predecessor pipeline multiplied each residue row by the legacy recovery rate of its region and crop category (`recovery_rates` in `residue_recovery.csv`, see `calculate_residue_destinies()`) before writing it. Every reader of `value` treats it as the whole residue and applies a recovery rate of its own, so `value` is the pin divided by that same rate (whep#1195). The division is exact: WHEP ships the rate the predecessor used, and on the real pin the result equals the predecessor's gross residue on all 427,584 rows where both exist. Where that rate is 0 (fodder crops in seven of the eight regions; roots and tubers, cassava, sugar beet and dry beans in West Europe and in North America and Oceania) the pin holds 0 and the gross residue cannot be recovered from it. Fodder crops and sugar beet carry no residue there, as before; roots and tubers, cassava and dry beans are estimated from yield (see "Non-cereal residue") and so carry the residue they produce, of which the recovery rate of 0 downstream recovers nothing.

The ratio behind the pin's gross residue is the fresh-matter residue:product ratio of biomass_coefs, scaled by the region's residue:product ratio relative to West Europe (`residue_dm_product_dm` in the same table, Wirsenius (2000) Table 3.16) and by a harvest-index change factor per region and year. That factor is what makes the ratio vary by year. The predecessor's table carries it at eight anchor years from 1910 to 2000, interpolates linearly between them and holds 2000 constant after it, falling for instance from 1.10 in 1962 to 1.00 in 2000 in East Europe. Its code attributes the table to Krausmann et al. (2013), *PNAS* 110:10324, Table M2 (assumed, unverified), and WHEP does not ship it.

The predecessor looked the regional ratio up by `regions_full$region_HANPP`, which carries Wirsenius's eight region names but not his membership (Wirsenius 2000, Table 3.1): Southeast Asia, Russia, Belarus, the Caucasus and Sudan took another region's ratio (whep#1430). For every crop that reads the pin, the ratio is re-keyed here on Wirsenius's own membership, through the (HANPP region, UN M49 sub-region) pairs listed in `residue_feed_regions.csv`. That lowers world non-cereal residue dry matter by 0.2% in 2010.

Cereal residue

Cereals are the one part of the base with a published global series: Smerald, Rahimi & Scheer (2023), *A global dataset for the production and usage of cereal residues in the period 1997-2021*, Scientific Data 10:685, doi:10.1038/s41597023025870, the mean of three methods (constant regional ratios, and two yield-dependent ones). The pin's model put world cereal residue 16.1% above that mean over 1997-2021 (3899 against 3357 Tg dry matter), and 25% above their constant-ratio method in every year with the same grain production (whep#1448). Besides the membership above, two things caused it. The West Europe anchor presumes that biomass_coefs holds the West Europe ratio, and for cereals it does not (wheat 1.34, barley 1.18, sorghum 1.70 and maize 0.96 kg dry matter per kg grain dry matter, against Wirsenius's 1.0, 1.0, 1.2 and 1.2). And Wirsenius's ratios are early-1990s harvest indices, held constant after 2000 while yields rose.

Cereal residue is therefore estimated from the pin's own production and harvested area (its `Product` rows, which equal the `primary_prod` pin) with `calculate_crop_residues()`, the estimator the soil carbon and nitrogen chain already uses. Its modern-variety harvest-index correction applies, keyed on the HANPP region its adoption table is written in. The `"ipcc"` line is IPCC (2019), *2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories*, Vol. 4, Ch. 11, Table 11.2 (p. 11.19), whose cereal slopes and intercepts are those of IPCC (2006) Table 11.2. Measured against Smerald et al. (`validation/residue_base_dm.R`), world cereal residue in dry matter over 1997-2021 and in 2010 is:

`cereal_residue`	1997-2021 mean, Tg	vs Smerald	years outside band	2010, Tg
`"ipcc"`	3385	+0.8%	0 of 25	3333
`"ensemble"`	3059	-8.9%	5 of 25	3008
`"ratio"`	2732	-18.6%	25 of 25	2684
`"wirsenius"`	3656	+8.9%	9 of 25	3598
the pin's model	3899	+16.1%	16 of 25	3826

The band is the spread of Smerald et al.'s three methods, widened by 5%; their 2010 mean is 3334 Tg. `"ipcc"` matches the world total, but per crop it books maize about 19% and sorghum about 20% below their mean and wheat about 23% above it (2010 and 2020). `"wirsenius"` keeps the pin's harvest-index factor and corrects the anchor by applying Table 3.16 directly: residue dry matter is grain dry matter times the ratio of the area's Wirsenius region.

Non-cereal residue

The West Europe anchor is not right for non-cereals either, and the error runs both ways (whep#1518). Against the line IPCC (2006) Table 11.2 gives each crop (Vol. 4, Ch. 11, pp. 11.17-11.18), the pin's ratio put world residue dry matter in 2010 at 1.42 times the line for soybean, 1.36 for potato, 1.95 for cassava and 1.88 for sweet potato, but 0.53 for dry beans, 0.41 for the other pulses and 0.34 for chickpeas. Applying Wirsenius (2000) Table 3.16 directly does not cure it: soybean would fall to 0.86 of the line.

So the crops that Table 11.2 gives a line of their own are estimated from the pin's production and harvested area exactly as cereals are, with `calculate_crop_residues()` and its modern-variety correction:

crops	Table 11.2 row	slope	intercept, t DM/ha
soybean	Soyabean	0.93	1.35
groundnut (with shell)	Peanut (w/pod)	1.07	1.54
dry beans, broad beans	Dry bean	0.36	0.68
peas, chickpeas, lentils, other pulses	Beans & pulses	1.13	0.85
potato	Potato	0.10	1.06
cassava, sweet potato, yams, other roots	Tubers	0.10	1.06

The 2019 Refinement's Table 11.2 (updated, p. 11.19) repeats these values. "Tubers" is IPCC's own crop class, "modelled after potatoes"; the crops reach it through `ipcc_crop_mapping.csv`,

as in the soil carbon and nitrogen chain. The other non-cereal lines in `ipcc_residue_coefs.csv` (rapeseed, sunflower, sugar beet, sugar cane, cotton) are not in Table 11.2, so those crops, tree crops and vegetables keep the pin's ratio, as does every crop under `non_cereal_residue = "pin"`.

World residue dry matter of the crops estimated from yield, Tg, the pin's ratio against `"ipcc"` (the default):

	1961	2010	2020
soybean	51 -> 63	500 -> 385	673 -> 494
groundnut	33 -> 48	90 -> 93	112 -> 110
dry beans	7 -> 23	14 -> 34	15 -> 37
peas	8 -> 17	14 -> 17	20 -> 22
other pulses	18 -> 57	31 -> 79	40 -> 97
potato	26 -> 35	36 -> 28	42 -> 27
cassava	19 -> 16	65 -> 37	79 -> 46
sweet potato	24 -> 20	20 -> 12	19 -> 11
all non-cereal residue	533 -> 625	2159 -> 2070	2755 -> 2595
all residue	2672 -> 2764	5492 -> 5402	6622 -> 6462

Soybean, groundnut and potato then sit at 1.05-1.11 of their IPCC line in 2010 and 2020 (`validation/residue_base_dm.R`): the modern-variety correction and the product dry-matter content of `bio_coefs`. No published global non-cereal series exists to check them against independently (whep#1399).

Oil palm residue

Oil palm residue is replaced rather than taken from the pin (whep#1424). Read as gross residue, the pin's oil palm residue is far above an oil palm plantation's whole above-ground net primary production, fruit included: 81% of 17.3 Mg C, so about 30 t DM at the 0.47 kg C per kg DM of biomass_coefs, per hectare and year in the industrial plantation of Wakhid, Hirano, Dariah & Agus (2022), *Mires and Peat* 28:02, doi:10.19189/map.2021.snpg.sta.2288. In Malaysia it is several times the country's whole solid oil palm biomass, mill residues included.

The anchor is Malaysia's field residue: fronds (46 Mt) and trunks (14 Mt) of dry matter on 4.85 Mha planted in 2010, Exhibit 1 and p. 10 of Agensi Inovasi Malaysia (2013), *National Biomass Strategy 2020*, version 2.0 (FAOLEX mal228571). Mill residues (empty fruit bunches, fibre, shells) are part of the fruit bunch and are not field residue, so they are left out. The anchor is a gross residue, so it replaces the pin's figure after the recovery rate has been undone and the ratio re-keyed; neither touches it. The anchored dry matter is converted to fresh mass with the oil palm `Residue_kgDM_kgFM` of biomass_coefs, so `value_dm` is the anchor exactly. That coefficient is 0.396, the dry matter of fresh oil palm fronds (39.6% as fed, n = 3) in Heuzé, Tran, Sauvant & Lebas (2015), *Oil palm fronds and oil palm crop residues*, Feedipedia, `https://www.feedipedia.org/node/6916` (whep#1516). Fronds are 46 of the 60 Mt; trunks, whose sap is 75-80% of their weight, are wetter, so the fresh mass is if anything understated. It sets the fresh mass booked to `Firewood`, not the dry matter.

The batch pin on the build path

Two further aliases were published in that same 2025-07-14 batch — `crop_residues` and `bilateral_trade`, alongside the four above, between 12:33:43Z and 12:33:50Z. Unlike the four, these two are read on the default build path, which is why no warning is attached to them: every commodity balance build reads them, so a warning at the read is noise rather than information. `bilateral_trade` has since been regenerated by this package (#1122), so `crop_residues` is the only one still registered at its 2025-07-14 version.

They did **not** share a provenance despite sharing a timestamp. Each was established separately (#1054), because the timestamp alone establishes nothing.

`crop_residues` is predecessor output:

Read by `get_primary_residues()`, and from there by `build_commodity_balances()`. All 475,688 of its `Product` rows equal the `primary_prod` pin's `tonnes` values exactly — no key unmatched on either side, no value differing at a relative tolerance of 1e-6 — so it is a downstream artifact of the same predecessor run, carrying that run's production series into the commodity balance.

Its residue quantities are those production numbers times a residue-to-product ratio that varies by year, and times a **recovery rate**: the pin holds the residue the predecessor recovered from the field, not the residue produced (#1195). The predecessor's code (`Global/R/crop_npp.r`, then `afsetools::residue_use()`) writes each row as

```
product * kg_residue_kg_product_FM * HI_changes(region, year) * residue_dm_product_dm(region, category) / residue_dm_product_dm("West Europe", category) * recovery_rates(region, category),
```

with region and category keyed as in `residue_recovery.csv`. Rebuilt from the predecessor's own coefficient tables, that formula reproduces all 425,934 pin rows whose region it can key, at a relative tolerance of 1e-6. The year variation is `HI_changes` alone: a harvest-index change factor per region, given at eight anchor years from 1910 to 2000, linear in between and flat after 2000 (East Europe 1.10 in 1962 to 1.00 in 2000), which the predecessor attributes to Krausmann et al. (2013), *PNAS* 110:10324, Table M2 (assumed, unverified). It is a regional historical adjustment, not a yield response, and after 2000 the ratio of every region and category is constant. That is why `biomass_coefs$kg_residue_kg_product_FM` alone reproduces only 3,189 of the 472,790 comparable keys. Of the 33,411 keys that carry a residue of exactly 0, 30,620 are cells whose recovery rate is 0 (fodder crops in seven of the eight regions; roots and tubers, cassava, sugar beet and dry beans in West Europe and in North America and Oceania) and 1,919 are crops whose `biomass_coefs` ratio is 0.

WHEP ships the recovery rate and the regional ratio the predecessor used (both equal in all 160 cells), but not the `HI_changes` table. So `get_primary_residues()` undoes the recovery exactly, dividing by the same rate, but cannot recompute the predecessor's residue from production. It then re-keys the regional ratio on Wirsenius's own region membership, and estimates cereal residue (#1448), and that of the non-cereal crops with their own IPCC Table 11.2 line (#1518), from the pin's production and harvested area instead of reading it.

It is not a small input. The pin holds 7.63 Gt of recovered residue for 2010 (Straw 3.60 Gt, Other crop residues 2.49 Gt, Firewood 1.54 Gt) and 327.7 Gt over 1961–2021, which `get_primary_residues()` turns into 7.64 Gt (Straw 3.92 Gt, Other crop residues 2.82 Gt, Firewood 0.90 Gt) and 366 Gt of residue produced. 3,998 of its 249,095 output rows carry `NA` polity columns because the pin is name-keyed.

Driving the same residue model off a fresh `get_primary_production()` would move those numbers. Measured for 2010 at (`area_code`, `item_prod`): 1.958 Gt of current production sits on 6,425 keys the pin never sees (809 Mt of it primary crops, including the modelled temporary-grassland item), 64.8 Mt on 296 keys is pin-only, and of the 8,010 shared keys 538 disagree, putting 1.102 Gt — 11.01% of the pin's shared-key mass — more than 1% apart. The largest class is rice, 673.9 Mt in the pin against 463.3 Mt fresh, a ratio of 0.6876: current code puts rice on a milled-equivalent basis while the pin's is paddy, which is a basis difference rather than an error, since straw scales with the field crop. The second is fodder, where the current build is close to twice the pin on every forage and silage item, the pin predating that work. Replacing the pin therefore means choosing a residue model, which is a science decision and not a refresh. For cereals that model is chosen by `get_primary_residues(cereal_residue = )`, and for the non-cereals with an IPCC line by `non_cereal_residue =`; both still run on the pin's production.

`bilateral_trade` is built by `build_detailed_trade()`:

Read by `get_bilateral_trade()` and by the live-animal branch of `build_commodity_balances()`. The registered version, `20261001T135651Z-ca7b3`, is `build_detailed_trade()`'s output verbatim, with its defaults and `extend_time = FALSE`, run on the `faostat-trade-bilateral` pin `20260407T095142Z-b3f81` (#1122): 10,038,220 rows over 1986–2021, polity-keyed, carrying 64,450,133,319 tonnes and 87,840,869,640 head.

The 2025-07-14 version it replaces, `20250714T123347Z-2c392`, held the same FAOSTAT Detailed Trade Matrix values in the predecessor pipeline's name-keyed schema — against the raw pin over 2010, all 296,642 shared `tonnes` keys and all 5,102 shared `Head` keys agree exactly — but without FAOSTAT's `1000 Head` rows: 76,141,882 thousand head of live broiler chicken, turkey, duck, rabbit and goose trade over 1986–2021, which PR #1113 recovered in the producer. The regenerated pin carries 7.50x its 11,707,083,640 head and 22.5% less than its 83,147,095,412 tonnes, all of that tonnage difference in CBS items 5001-5003, which no commodity balance carries: item 1293's unbacked tonnage, screened by `method_unbacked_quantity` (#1023), and three CBS names with no `item_cbs_code`. On the tonnes that do reach a trade matrix the two versions differ by 0.056%. `.clean_bilateral_trade()` reads either schema.

Examples

```
get_primary_residues(example = TRUE)
```

`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(
  years = NULL,
  trade_recovery = .cbs_trade_recovery_choices(),
  example = FALSE
)
```

Arguments

`years`	Optional integer vector of years to build. When `NULL` (default) the whole series is built. Supplying a window calibrates the coefficients on that range of the full-range commodity balances (see `get_wide_cbs()` for why those are always built over the whole series) and caches them under a window-specific key.
`trade_recovery`	One of `"byproduct_residual"` (default), `"pass_through"`, `"net_import"` or `"none"`, selecting the CBS the coefficients are calibrated on. See `build_commodity_balances()` and `get_wide_cbs()`. Pass the same value here as to `get_wide_cbs()`: coefficients calibrated on one CBS do not describe the other.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`.

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_co` 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

```
get_processing_coefs(example = TRUE)
```

---

get_provenance *Retrieve a result's provenance record.*

Description

Return the provenance record attached by `attach_provenance()`, or `NULL` when none is present.

Usage

```
get_provenance(x)
```

Arguments

`x`	An object that may carry a `whep_provenance` attribute.

Value

The provenance tibble, or `NULL`.

Examples

```
get_provenance(tibble::tibble(value = 1))
```

get_scope *Retrieve a result's scope record.*

Description

Return the scope record attached by `attach_scope()`, or `NULL` when none is present.

Usage

```
get_scope(x)
```

Arguments

`x`	An object that may carry a `whep_scope` attribute.

Value

The scope tibble, or `NULL`.

Examples

```
get_scope(tibble::tibble(value = 1))
```

get_soc_climate_drivers *Assemble monthly SOC climate drivers from CRU climate and LPJmL hydrology.*

Description

Builds the monthly per-cell climate drivers the soil-organic-carbon decomposition modifiers consume: air temperature, topsoil soil-water saturation, monthly precipitation and potential evapotranspiration (the Century modifier drivers), the monthly water-minus-potential-evapotranspiration surplus (the RothC/HSOC driver), the annual water balance (the AMG driver), the volumetric soil water content with its field-capacity, wilting-point and porosity references (the ICBM moisture drivers) and clay content. Air temperature comes from CRU TS 4.09 (`read_cru_climate()` `"tmp"`, degrees Celsius); potential evapotranspiration from CRU `"pet"` (mm/day), converted to a monthly total by multiplying by the days in the month; the water input (precipitation plus irrigation) from the LPJmL run so it is consistent with the hydrology that produced the soil water content; and soil water content from LPJmL directly (`read_lpjml_hydrology()` `"swc"`, topmost layer). The soil hydraulic references (field capacity, wilting point, porosity) come from the dominant HWSD texture class of each cell via `read_soil_hydraulic()`, and the volumetric soil water content is `theta = swc_topsoil * porosity` (the LPJmL fractional saturation scaled by the cell porosity). Clay content is a soil-texture covariate supplied via `data$clay`; the polity key comes from a cell-polity crosswalk (`data$cell_polity`).

Usage

```
get_soc_climate_drivers(
  run_dir = NULL,
  years = NULL,
  polity_validity = c("keep", "flag", "drop"),
  partial_year = c("abort", "warn", "drop"),
  data = list(),
  example = FALSE
)
```

Arguments

`run_dir`	Path to the LPJmL run output directory. `NULL` (default) uses `WHEP_LPJML_RUN_DIR` when set, and the pinned `lpjml-soc-hydrology` artifact otherwise, so running LPJmL is not a prerequisite. That artifact holds only the three LPJmL monthly drivers (topsoil saturation, precipitation, irrigation); air temperature still comes from CRU and the texture products from HWSD, both downloadable, so neither is pinned.
`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year the inputs cover.
`polity_validity`	What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.
`partial_year`	What to do when a cell-year does not carry all twelve months by the time `water_balance_mm` is summed. A cell-month survives to that point only if CRU temperature, CRU PET, LPJmL precipitation and LPJmL irrigation all carry it, so an inner join can strip one and the annual surplus is then summed over eleven with nothing `NA` and no row visibly lost. `"abort"` (default) refuses, naming the absent cell-months; `"warn"` returns the short sums; `"drop"` excludes the affected cell-years.
`data`	Optional named list of pre-loaded inputs, each falling back to its reader when absent: `temp` (CRU `tmp`, `lon`, `lat`, `year`, `month`, `value` degrees Celsius), `pet` (CRU `pet`, same schema, mm/day), `prec` and `irrig` (LPJmL monthly, `lon`, `lat`, `year`, `month`, `value` mm/month), `swc` (`read_lpjml_hydrology()` soil water content), `clay` (`lon`, `lat`, `clay_pct`, required), `cell_polity` (`lon`, `lat`, `area_code`, the polity crosswalk, required) and `soil_hydraulic` (`lon`, `lat`, `t_field`, `t_wilt`, `porosity`; falls back to `read_soil_hydraulic()`, cropped to `cell_polity` when supplied). `cell_polity` is used only to **label** each cell with an `area_code` and to restrict the grid to the cells it covers; no quantity here is ever multiplied by

an area. It therefore decides this function's **footprint**, and callers that pass different crosswalks get different footprints from one function. The carbon path passes the polycell support (`build_polycell_support()`, via `read_polycell_support()`); the water path still passes `build_cell_polity()` until it migrates, so the two footprints differ by the crosswalks' own difference until then.

`example` If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Details

The monthly PET total is `pet_mm = pet_mm_day * days_in_month`; the monthly water surplus is `water_minus_pet_mm = (precip_mm + irrig_mm) - pet_mm`; and the annual water balance `water_balance_mm` is the per-cell-year sum of that surplus, repeated across every month of the cell-year so it can drive the AMG modifier that expects one annual scalar per cell-year. `precip_mm` carries precipitation only (irrigation excluded), as the Century moisture factor expects. The volumetric soil water content `theta = swc_topsoil * porosity` varies by month with the LPJmL saturation, while its `t_field`, `t_wilt` and `porosity` references are static per cell (the dominant HWSD texture class' properties), together driving the ICBM piecewise moisture response. The water input basis is recorded in `method_water_input` (`"lpjml_prec_irrig"`, LPJmL precipitation plus irrigation). Air temperature (CRU) and the soil texture products (clay, hydraulic properties) are not LPJmL outputs, hence the mixed sources.

The drivers cover the intersection of the CRU and LPJmL grids. CRU TS 4.09 masks as water 22 of the 58,795 cells the `lpjml-soc-hydrology` pin carries (small islands, coasts and large lakes), so those cells are excluded; the function reports the count every time it drops any (a message of class `whep_socd_cell_shortfall`) and aborts (class `whep_socd_cell_loss`) when more than 0.1% of the LPJmL cells in any year have no climate series.

Value

A tibble with `lon`, `lat`, `area_code`, `year`, `month`, `temp_c`, `swc_topsoil`, `precip_mm` and `pet_mm` (monthly, the Century modifier drivers), `water_minus_pet_mm` (the monthly RothC/HSOC surplus), `water_balance_mm` (the annual sum of `water_minus_pet_mm`, the AMG modifier driver, repeated across a cell-year's months), `clay_pct`, `theta`, `t_field`, `t_wilt` and `porosity` (the ICBM moisture drivers: the monthly volumetric soil water content and its static field-capacity, wilting-point and porosity references) and `method_water_input`, plus the polity columns below, plus `reporting_polity_out_of_span` when `polity_validity = "flag"`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.

- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
get_soc_climate_drivers(example = TRUE)
```

`get_wide_cbs` *Commodity balance sheet data.*

Description

Retrieve supply and use parts for each commodity balance sheet (CBS) item. Stock variations are split into two non-negative columns following the FABIO methodology.

Usage

```
get_wide_cbs(
  years = NULL,
  trade_recovery = .cbs_trade_recovery_choices(),
  example = FALSE
)
```

Arguments

`years`	Optional integer vector of years to return. When `NULL` (default) the whole series is returned. A window returns exactly those years of the full-range result. The balances themselves are always built over the whole series, once per session and then cached, because several of their gap fills carry an observation across the whole year axis: a balance built over the window alone would differ from the same years of the full build (whep#833). A window therefore costs as much as the full series the first time, and nothing after that.
`trade_recovery`	One of `"byproduct_residual"` (default), `"pass_through"`, `"net_import"` or `"none"`, passed to `build_commodity_balances()`, which documents what each does and what the default moves. Each method is built and cached under its own slot, so asking for one never serves the other's result.
`example`	If `TRUE`, return a small example output without downloading remote data. Default is `FALSE`. The example is the same fixture under either `trade_recovery`.

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()`.
- `unit`: The denomination of every quantity in the row. `"tonnes"` for the rows of the FAO-style balance sheet and `"heads"` (number of animals) for the live-animal rows added by the livestock balance. Set by the builder that produced the row, so a derived row carries the same unit as a reported one. Never sum quantities across rows of different units.

The other columns are quantities where total supply and total use should be balanced, in the row's `unit`.

For supply:

- `production`: Produced locally.
- `import`: Obtained from importing from other countries.
- `stock_withdrawal`: Biomass taken out of storage (non-negative). Positive when stocks decrease.

For use:

- `food`: Food for humans.
- `feed`: Food for animals.

- `export`: Released as export for other countries.
- `seed`: Intended for new production.
- `processing`: Used to obtain other subproducts.
- `processing_primary`: Used to obtain other subproducts, for the handful of primary items (palm fruit, hops, seed cotton, coconuts, hemp, kapok fruit, linum) whose entire domestic supply is destined for processing. Zero for every other item.
- `other_uses`: Any other use not included above.
- `stock_addition`: Biomass placed into storage (non-negative). Positive when stocks increase.

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

The live-animal rows also carry `method_cull`, saying where the slaughter of culled dairy cattle and laying hens was booked (see the argument of that name in `get_livestock_cbs()`); it is `NA` on the tonnes rows.

Examples

```
get_wide_cbs(example = TRUE)
```

---

gleam_animal_weights *GLEAM animal weights.*

Description

Typical live weights by region, species, system, and cohort.

Usage

```
gleam_animal_weights
```

Format

A tibble with `region`, `species`, `system`, `cohort`, `weight_kg`.

Source

The shipped values are unsourced placeholders, but the real GLEAM table has been located (whep#881): FAO. 2018. *Global Livestock Environmental Assessment Model, Model description, Version 2.0, Revision 5.* Rome, FAO, Supplement S1, "Live weights (kg)" block of Tables 2.4 (dairy cattle), 2.5 (beef cattle), 2.6 (feedlot cattle), 2.7 (buffaloes), 2.10 (sheep), 2.11 (goats) and 2.14-2.16 (backyard, intermediate and industrial pigs), workbook https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_2.0_Supplement_S1.xlsx (md5 `72fd2ea477dfe8b30cd3657b2baa4af1`, retrieved 2026-08-26). The Version 3.0 description drops those regional herd-parameter tables, which is why the committed GLEAM 3.0 Supplement S1 workbook has no sheet for them.

The shipped values do **not** match that source and have not been replaced here, because doing so is a science decision that moves published numbers: `.join_weights()` in `R/livestock_energy.R` uses this table as the Tier 2 live weight, and gross energy scales as `weight^0.75` for maintenance and `weight` for activity. Measured per cohort against the GLEAM values (system-averaged the way `.join_weights()` averages them), gross energy – hence Tier 2 enteric CH4 – would move by: Global cattle adult female 400 -> 457 kg, +6.9%; Global cattle fattening 300 -> 399 kg, +14.2%; Western Europe cattle adult male 1000 -> 732 kg, -16.4%; North America cattle adult female 615 -> 700 kg, +7.3%; Global sheep adult female 45 -> 53 kg, +7.2%. GLEAM also publishes no live weight for the Replacement cohorts (its herd module derives them), so a re-ingest cannot remove every assumption. Separately, the `"Latin America"` rows are unreachable: `.gleam_region_of()` emits GLEAM 3.0 region labels, in which that region is `"Central & South America"`, so those rows silently take the Global weights.

Examples

```
gleam_animal_weights
```

---

gleam_crop_residue_nitrogen

Nitrogen parameters for crop residues of feed materials.

Description

Nitrogen content of above- and below-ground residues and root-to-shoot ratios for feed materials.

Usage

```
gleam_crop_residue_nitrogen
```

Format

A tibble with columns:

material_number Sequential material identifier.

material Feed material code.

n_ag Nitrogen content of above-ground residues.

rbg_bio Ratio of below-ground residues to above-ground biomass.

n_bg Nitrogen content of below-ground residues.

species_group `"ruminant"` or `"monogastric"`.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Tables S.6.7 and S.6.8: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_crop_residue_nitrogen
```

`gleam_crop_residue_params`
GLEAM crop residue parameters.

Description

Dry matter content and parameters for calculating crop residue yield by crop type.

Usage

```
gleam_crop_residue_params
```

Format

A tibble with columns:

crop Crop name.

dry_matter_pct Dry matter content (percent).

slope Slope for residue yield calculation.

intercept Intercept for residue yield calculation.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Table S.3.1: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_crop_residue_params
```

`gleam_dressing_percentages` *GLEAM dressing percentages.*

Description

Carcass weight as percentage of live weight by species, production system, cohort, and GLEAM region. Includes country-specific overrides for industrial pig systems in Western Europe.

Usage

```
gleam_dressing_percentages
```

Format

A tibble with columns:

species Animal species (Cattle, Buffaloes, Sheep, Goats, Pigs, Chicken).

production_system Production system (Dairy, Beef, Backyard, Intermediate, Industrial, Layers, Broilers). NA for species without system breakdown.

cohort Cohort (e.g. Adult and replacement female). NA for species without cohort breakdown.

country Country name for country-specific values. NA for regional values.

gleam_region GLEAM region abbreviation (NA, RUS, WE, EE, NENA, ESEA, OCE, SA, LAC, SSA).

dressing_percent Dressing percentage.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Table S.9.1: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_dressing_percentages
```

`gleam_energy_use_ef` *Energy use emission factors for livestock production.*

Description

Emission factors for embedded (feed-production) and direct (on-farm) energy use in livestock production, from GLEAM 3.0 tables S.7.1 through S.7.7. Note that the factors are expressed per kilogram of *live weight*, *milk* or *egg* depending on the species and herd: see the `denominator` column. The GLEAM footnotes are materialised as derived rows: embedded energy for meat (non-dairy) cattle and all buffalo is half of dairy cattle (S.7.1 note a); embedded energy for non-dairy small ruminants is half of the listed values (S.7.2 note a); direct energy for dairy small ruminants is double the dairy cattle values (S.7.5 note a).

Usage

```
gleam_energy_use_ef
```

Format

A tibble in long format with columns:

species Animal species or group (`"cattle"`, `"buffalo"`, `"large_ruminants"`, `"small_ruminants"`, `"pigs"`, `"chickens"`).

herd Herd or product line (`"dairy"`, `"non_dairy"`, `"broilers"`, `"layers"`, `"all"`). `NA` for pigs.

grouping Country or country group as reported by GLEAM (e.g. `"OECD"`, `"EU 27"`, `"Least developed countries"`).

grouping_scheme Which country grouping the `grouping` belongs to: `"development3"` (OECD / least developed / others), `"region5"` (OECD / four non-OECD regions) or `"detailed15"` (individual OECD members plus world regions).

system Production system (e.g. `"grassland_based"`, `"industrial"`). `NA` when not applicable.

climate Climate zone (`"arid"`, `"humid"`, `"temperate"`). `NA` when not applicable.

energy_type `"embedded"` or `"direct"`.

denominator Reporting basis of the factor: `"lw"` (live weight), `"milk"` or `"egg"`.

emission_factor Emission factor in kg CO2-eq per kg of the `denominator`.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Tables S.7.1 through S.7.7: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_energy_use_ef
```

`gleam_enteric_params` *Enteric methane conversion factors (IPCC defaults, not GLEAM).*

Description

Ym (% of gross energy converted to CH4) by species and production system. Despite the object name, the values are IPCC defaults, mixed across the 2006 and 2019 editions, plus one unsourced row; none is a GLEAM value. No function in `R/` reads this object; the Tier 2 enteric chain uses ipcc_tier2_ym_values instead.

Usage

```
gleam_enteric_params
```

Format

A tibble with `species`, `system`, `ym_percent`, `notes`.

Source

Traced row by row against the published tables (whep#959):

- Cattle and buffalo 6.5% (grazing, mixed) and feedlot cattle 3.0% are the 2006 IPCC Guidelines, Vol 4, Ch 10, Table 10.12, which gives 6.5% for every non-feedlot cattle and buffalo class and 3.0% for cattle fed diets of 90 percent or more concentrates. They are not the 2019 Refinement's Table 10.12 (Updated), which resolves Ym by production level and digestibility (5.7 / 6.0 / 6.3 / 6.5 for dairy cows and buffalo, 7.0 / 6.3 for non-dairy, 4.0 for grain feedlots and 3.0 only for steam-flaked-corn feedlots).
- Sheep 6.5% is the 2006 Table 10.13 "Mature Sheep" value (lambs under one year are 4.5% there). The 2019 Refinement's Table 10.13 (Updated) replaces it with a single 6.7% for all sheep.
- Goats 5.5% is the 2019 Refinement, Vol 4, Ch 10, Table 10.13 (Updated); the 2006 edition has no goat Ym.
- Pigs 0.0% ("negligible") appears in no IPCC table. **Assumed, unverified.** IPCC gives swine no Ym, only a Tier 1 enteric factor of 1.5 (developed) and 1.0 (developing countries) kg CH4 head-1 yr-1 (2006 Table 10.10; the same in the 2019 Table 10.10 (Updated)).

GLEAM does not publish fixed Ym values of this shape. FAO. 2018. *Global Livestock Environmental Assessment Model, Model description, Version 2.0, Revision 5*, Table 4.12, p. 67, computes Ym for non-feedlot cattle, buffalo and adult small ruminants as 9.75 - 0.05 x ration digestibility (7.75 - 0.05 x digestibility for young small ruminants), uses 3 for feedlot cattle, and gives pigs 1.01 (adult reproductive) and 0.39 (replacement and fattening), not 0. The GLEAM 2.0 and 3.0 Supplement S1 workbooks in `data-raw/` contain no Ym table. The Version 3.0 model description was not checked.

Examples

```
gleam_enteric_params
```

---

gleam_feed_categories

GLEAM feed categories.

Description

Feed classification used in GLEAM 3.0.

Usage

```
gleam_feed_categories
```

Format

A tibble with `feed_category`, `feed_type`, `description`.

Source

Not traced to any GLEAM document (whep#881). Searched and ruled out: the GLEAM 3.0 Supplement S1 workbook (no sheet holds it); FAO. 2022. *Model Description, Version 3.0*, Table 3.1/3.3 (ruminants, 27 feed materials grouped as Roughages / Cereals / By-products / Concentrates, pp. 32-37) and Table 3.5 (monogastrics, 42 materials grouped as Swill and scavenging / Locally-produced / Non-local, p. 45); and Tables 3.2, 3.3 and 3.14 of the Version 2.0 Revision 5 description. None of those groupings is the six-way Grass / Crop residues / Concentrates / Fodder crops / Processed feeds / Animal products split shipped here, and "Animal products - Milk, fish meal" has no counterpart in GLEAM's ruminant material list at all. Treat the classification as a WHEP-local convenience taxonomy, not a GLEAM table. No function in `R/` reads it.

Examples

```
gleam_feed_categories
```

---

gleam_feed_composition

GLEAM feed use efficiency.

Description

Regional feed use efficiency (FUE) values for forages and crop residues of ruminant species.

Usage

```
gleam_feed_composition
```

Format

A tibble with columns:

feed_group Feed material group (1-6 or 9-15).

feed_type Feed type (mixed, grassland, or all).

gleam_region GLEAM geographic region.

feed_use_efficiency FUE value (0-1 fraction).

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Table S.3.2: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_feed_composition
```

---

gleam_feed_conversion_ratios

GLEAM feed conversion ratios for monogastrics.

Description

Nutritional values for feed materials of monogastric species (chicken and pigs).

Usage

```
gleam_feed_conversion_ratios
```

Format

A tibble with columns:

number Feed material number.

material Feed material code.

gross_energy_j_kg Gross energy (J per kg).

n_content_g_kg Nitrogen content (g per kg DM).

me_chicken_j_kg Metabolisable energy for chicken (J per kg).

me_pigs_j_kg Metabolisable energy for pigs (J per kg).

digestibility_pct Digestibility (percent).

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Table S.3.4: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_feed_conversion_ratios
```

`gleam_feed_digestibility`
GLEAM feed digestibility for ruminants.

Description

Nutritional values for feed materials of ruminant species, including gross energy, nitrogen content, and digestibility.

Usage

```
gleam_feed_digestibility
```

Format

A tibble with columns:

number Feed material number.

material Feed material code.

gross_energy_mj_kg Gross energy (MJ per kg DM).

n_content_g_kg Nitrogen content (g per kg DM).

digestibility_pct Digestibility (percent).

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Table S.3.3: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_feed_digestibility
```

gleam_field_operation_ef

Emission factors for field operations on feed materials.

Description

CO2-equivalent emissions per hectare from field operations for ruminant and monogastric feed materials.

Usage

```
gleam_field_operation_ef
```

Format

A tibble with columns:

material_number Sequential material identifier.

material Feed material code (e.g. `"GRASSF"`, `"WHEAT"`).

emission_factor_kg_co2eq_ha Emission factor in kg CO2-eq per hectare.

species_group `"ruminant"` or `"monogastric"`.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Tables S.6.1 and S.6.2: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_field_operation_ef
```

gleam_fracremove

Country-level fraction of crop residues removed.

Description

Countries whose FracReMove value differs from the GLEAM default.

Usage

```
gleam_fracremove
```

Format

A tibble with columns:

country Country name.

continent Continent.

region GLEAM region.

fracremove Fraction of crop residues removed (0 to 1).

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Table S.6.9: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_fracremove
```

---

gleam_geographic_hierarchy

GLEAM geographic hierarchy.

Description

Maps countries (ISO3) to GLEAM regions, FAOSTAT regions, and classification indicators.

This is GLEAM's own registry of the countries that exist today, so `polity_identity_conventions()` types it `"present_day_polity"` and it carries the polity its `iso3` resolves to in the present day, as regions_full does. All 204 rows now resolve: ATF, SGS and WLF used to keep `NA` for want of any WHEP polity, and upstream supplied one for each. A gap would still stay visible rather than being guessed at. The 204 GLEAM regions themselves are unchanged.

`oecd` is the one column not carried through verbatim. Cell G41 of the published sheet flags Comoros as an OECD member; it is not one, so `data-raw/livestock_coefficients.R` rebuilds the whole column from the OECD's own list of its 38 Members (oecd.org, "Members and partners", read 2026-08-25; the URL is in the builder, which is not URL-checked, because the host answers `R CMD check --as-cran` with a 403) and aborts if the sheet ever stops listing a Member. Every other flagged code already matched that list, Colombia (2020) and Costa Rica (2021) included, and `eu27` beside it is exactly the 27 real EU members, so the cell reads as a data-entry slip rather than a GLEAM grouping that borrows the name. The flag is load-bearing: `build_energy_co2_extension()` reads it for two of GLEAM's three country-grouping schemes, and as shipped it priced Comoros' meat at OECD energy intensity, up to 129 percent high (whep#574).

Usage

```
gleam_geographic_hierarchy
```

Format

A tibble with columns:

iso3 ISO3 country code.

country Country name.

continent Continent.

faostat_region FAOSTAT regional grouping.

gleam_region GLEAM regional grouping.

eu27 1 for an EU-27 member, 0 otherwise.

oecd 1 for an OECD member, 0 otherwise.

reporting_polity_code The polities code `iso3` resolves to in the present day, `NA` where WHEP has no polity for the territory.

reporting_polity_name The name of that polity.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Tables S.A1-S.A2: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_geographic_hierarchy
```

`gleam_livestock_categories` *GLEAM livestock categories.*

Description

Species, production systems, and cohort definitions from GLEAM 3.0.

Usage

```
gleam_livestock_categories
```

Format

A tibble with columns:

species Animal species.

production_system Dairy, Beef, Meat, etc.

cohort Age/sex cohort.

description Cohort description.

Source

Partly traced (whep#881). The cohort vocabulary corresponds to FAO. 2022. *Global Livestock Environmental Assessment Model, Model Description, Version 3.0*. Rome, FAO, Table 2.1 "Summary of cohorts in GLEAM", p. 10 (document code `cd8425en`, served from `www.fao.org/3/` and `openknowledge.fao.org`), identical to Table 2.1 p. 9 of the Version 2.0 Revision 5 description (https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_2.0_Model_description.pdf). The shipped table is **not** a transcription of it: GLEAM names cohorts AF/RF/AM/RM/MF/MM (plus MFr/MMr/MFf/MMf for feedlots) within one herd, whereas this table renames them, crosses them with a Dairy/Beef/Meat/Other production system GLEAM's Table 2.1 does not have, and supplies its own `description` strings. GLEAM publishes no table of cohort shares; it derives herd structure from the replacement, fertility, mortality and age-at-first-calving rates in Supplement S1 of the Version 2.0 description. `calculate_cohorts_systems()` instead splits a herd equally, `1 / n` per row of this table, so the row COUNT per (species, production system) is a result-affecting, unsourced assumption: Cattle Dairy 6 cohorts (16.7% each), Cattle Beef 5 (20% each). Treat the layout and the implied equal split as unverified.

Examples

```
gleam_livestock_categories
```

---

gleam_mechanization_levels

Country-level mechanization levels for feed materials.

Description

Mechanization level by country for each feed material, for ruminant and monogastric species.

Usage

```
gleam_mechanization_levels
```

Format

A tibble in long format with columns:

country Country name.

continent Continent.

region GLEAM region.

feed_material Feed material code in lowercase.

mechanization_level Numeric mechanization level.

species_group `"ruminant"` or `"monogastric"`.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Tables S.6.3 and S.6.4: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_mechanization_levels
```

---

gleam_milk_production

GLEAM milk production.

Description

Average annual milk yields and lactation lengths by region.

Usage

```
gleam_milk_production
```

Format

A tibble with `region`, `species`, `system`, `milk_kg_head_yr`, `lactation_days`.

Source

Not traced to any GLEAM document, and GLEAM appears to publish no such table (whep#881). Searched and ruled out: the GLEAM 3.0 Supplement S1 workbook; the herd-parameter tables of Supplement S1 of the Version 2.0 Revision 5 description (Tables 2.4-2.21 give live weights and replacement, fertility, mortality and age-at-first-calving rates, but no milk yield and no lactation length); and both model descriptions, in which `MILKyield` (Equation 9.1) and the lactation period are country-level inputs taken from FAOSTAT and national data rather than defaults – Table 1.2 of the Version 2.0 description lists them under national/sub-national resolution. GLEAM-i likewise asks the user for "annual average milk yield per milking cow". Treat the values as unverified placeholders. No function in `R/` reads this object: the Tier 2 lactation energy term takes `milk_yield_kg_day` from the caller, and `.build_demand_energy()` derives it from the FAOSTAT `t_head` rows.

Examples

```
gleam_milk_production
```

`gleam_mms_shares` *GLEAM manure management system shares.*

Description

Regional MMS allocation by species and system.

Usage

```
gleam_mms_shares
```

Format

A tibble with `region`, `species`, `system`, `mms`, `share_percent`.

Source

The shipped values are unsourced placeholders, but the real GLEAM table has been located (whep#881): FAO. 2018. *Global Livestock Environmental Assessment Model, Model description, Version 2.0, Revision 5.* Rome, FAO, Supplement S1, Tables 4.2-4.11 (regional MMS averages for dairy cattle, beef cattle, feedlot cattle, dairy and non-dairy buffalo, small ruminants, backyard/intermediate/industrial pigs and chickens), workbook https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_2.0_Supplement_S1.xlsx (md5 72fd2ea477dfe8b30cd3 retrieved 2026-08-26). The Version 3.0 description and its Supplement S1 publish MMS *definitions* (Tables 4.1-4.5) but no regional shares. The shipped values disagree materially with Table 4.2/4.3: Western Europe dairy cattle is liquid/slurry 42%, solid storage 30%, pasture 27%, daily spread 1% in GLEAM against 60/30/10 here, and the Sub-Saharan Africa (90/10), Latin America (95/5) and South Asia (60/30/10) rows omit the drylot share GLEAM gives as 35%, 42% and 54% respectively. Not re-ingested here because no function in `R/` reads this object. The manure chain reads regional_mms_distribution instead, which whep#958 re-ingested from the same GLEAM 2.0 tables.

Examples

```
gleam_mms_shares
```

`gleam_processing_transport_ef` *Processing and transport emission factors for feeds.*

Description

Emission factors for processing and transport of feed materials, for ruminant and monogastric species.

Usage

```
gleam_processing_transport_ef
```

Format

A tibble with columns:

material_number Sequential material identifier.

material Feed material code.

processing_g_co2eq_kg_dm Processing emission factor in g CO2-eq per kg dry matter.

transport_g_co2eq_kg_dm Transport emission factor in g CO2-eq per kg dry matter.

species_group `"ruminant"` or `"monogastric"`.

Source

FAO (2022) GLEAM version 3.0, Supplement S1 (an FAO workbook; no DOI is issued for it), Tables S.6.5 and S.6.6: https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_3.0_Supplement_S1.xlsx

Examples

```
gleam_processing_transport_ef
```

---

glwd_water_fraction *Derive the lake-and-river fraction of each 0.5-degree cell from GLWD*

Description

Aggregates the Global Lakes and Wetlands Database rasters to the 0.5-degree grid as a fraction of the whole cell. This is the one implementation of that derivation in WHEP: `read_glwd_water()` calls it for `build_polycell_support()`, and `inst/scripts/prepare_spatialize_all.R` calls it to write LPJmL's `lakes_rivers` input. It used to live only in that script, so the polycell producer read a hand-made `.clm` artefact of an LPJmL run instead and the two answers were free to diverge.

Usage

```
glwd_water_fraction(glwd_dir, cells = NULL)
```

Arguments

`glwd_dir`	Directory holding the GLWD rasters, as `inst/scripts/download/download_hydrology.R` lays them out: `GLWD_v2/` for v2, or `glwd_3/hdr.adf` / `glwd_3.tif` for v1.
`cells`	Optional `tibble` of `lon`/`lat` cell centres to sample at. `NULL` (default) returns every cell of the 0.5-degree grid the rasters cover.

Value

A `tibble` with `lon`, `lat` and `water_frac`, carrying a `"glwd_version"` attribute of `"v1"` or `"v2"`.

Which classes count as inland water

GLWD v2 is a **33-class wetland map**, not a water fraction, so a subset has to be chosen and the choice is a judgement rather than a lookup. Taken here: lakes as classes 1-3 (freshwater lake, saline lake, reservoir) and rivers as class 7 (small streams). Everything else – palustrine and riverine wetland, peatland, mangrove, saltmarsh, rice paddies – is **excluded**: those are land that is wet, not surface water, and `build_polycell_support()` books them under `land_area_ha`.

Under GLWD v1 the equivalent classes are 1 (lakes) and 3 (rivers). The two vintages are not interchangeable and give totals about 20% apart; see `read_glwd_water()`.

Class membership is multiplied by the companion `area_pct` raster where one is present, so a partially covered source pixel contributes its own fraction rather than counting whole.

Examples

```
# Requires the GLWD download; not run without it.
if (nzchar(Sys.getenv("WHEP_LPJML_INPUT_DIR"))) {
  glwd_water_fraction(
    file.path(Sys.getenv("WHEP_LPJML_INPUT_DIR"), "GLWD")
  )
}
```

---

grass_access_shares *Accessibility and conversion parameters for grazable grass availability.*

Description

Accessibility and conversion parameters for grazable grass availability.

Usage

```
grass_access_shares(aboveground = 0.46, grazable = 1, w_c_dm = 0.45)
```

Arguments

`aboveground`	Fraction of total grass NPP that is above ground (the grazable compartment; LPJmL `pft_npp` is whole-plant). Default 0.46.
`grazable`	Sustainable fraction of above-ground forage that can be grazed. Default 1 (the full above-ground ceiling); set below 1 to impose a sustainable-offtake share.
`w_c_dm`	Carbon-to-dry-matter mass fraction. Default 0.45.

Value

A named list with `aboveground`, `grazable` and `w_c_dm`.

Examples

```
grass_access_shares(grazable = 0.6)
```

---

grazing_energy_coefs *Grazing energy coefficients.*

Description

Walking energy cost for grazing animals (MJ/kg body weight/km).

Usage

```
grazing_energy_coefs
```

Format

A tibble with `parameter`, `value_mj_kg_km`, `source`.

Source

NRC 2001 (0.00045 Mcal/kg/km converted to MJ).

Examples

```
grazing_energy_coefs
```

---

gridded_fallow_weights *Build agro-climatic, rainfed-gated fallow allocation weights.*

Description

Compute a per-(area, item) weight for distributing reported fallow to crops, from gridded crop placement and a crop x agro-climatic-zone propensity. For each grid cell, rainfed crop area is multiplied by the crop's propensity in the cell's agro-climatic zone (derived from GAEZ length of growing period and thermal climate), then summed to country x item. Dryland cereals/pulses score high in arid/semi-arid zones, rainfed rice high in the humid tropics (rice-fallow); perennials and the irrigated share score ~zero.

Usage

```
gridded_fallow_weights(gridded_crops, grid_aez = NULL, propensity = NULL)
```

Arguments

`gridded_crops`	Tibble keyed by grid cell and `item_cbs_code` with columns `lon`, `lat`, `area_code`, `rainfed_ha`.
`grid_aez`	Tibble of `lon`, `lat`, `lgp` (length of growing period in days), `thermal` (GAEZ thermal-climate class). If `NULL`, the `grid-aez` pin is read via `whep_read_file()` (built by `data-raw/grid_aez.R`).
`propensity`	Tibble of `item_cbs_code`, `zone`, `fallow_propensity`. If `NULL`, the packaged `fallow_propensity.csv` is used.

Value

A tibble with `area_code`, `item_cbs_code`, `weight`.

Examples

```
gridded_crops <- tibble::tribble(
  ~lon, ~lat, ~area_code, ~item_cbs_code, ~rainfed_ha,
  0.25, 50.25, 1L, 2511L, 500
)
grid_aez <- tibble::tribble(~lon, ~lat, ~lgp, ~thermal, 0.25, 50.25, 100, 7L)
propensity <- tibble::tribble(
  ~item_cbs_code, ~zone, ~fallow_propensity,
  2511L, "semiarid", 0.8
)
gridded_fallow_weights(gridded_crops, grid_aez, propensity)
```

`harmonize_interpolate`

Harmonize advanced cases with interpolation for 1:N groups

Description

Harmonize data containing `"simple"` and `"1:n"` mappings. `"simple"` covers both 1:1 and N:1 relationships (values are summed). For `"1:n"` groups (one original item splits into several harmonized items) this function computes value shares across the full year range, interpolates missing shares, and applies them to split values.

Important for 1:n mappings: For each original item that splits into multiple harmonized items (e.g., "wheatrice" into "wheat" and "rice"), provide **one row per target** `item_code_harm`. Each row should have the same `item`, `year`, and `value`, differing only in `item_code_harm`. For example, to disaggregate "wheatrice":

- Row 1: `item` = "wheatrice", `item_code_harm` = 1
- Row 2: `item` = "wheatrice", `item_code_harm` = 2

Do not provide a single row; the function will not create duplicates automatically.

A 1:n contribution can turn out to be unvaluable, because its `value` is missing or because its share could neither be computed nor interpolated (e.g. every year of the group has a zero

total). Such contributions are dropped with a warning rather than summed, so that one missing value cannot turn the observed `"simple"` values of the same `(item_code, year)` cell into `NA`.

Usage

```
harmonize_interpolate(data, ...)
```

Arguments

`data`	A data frame containing at least columns: • `item`: String, original item name. • `item_code_harm`: Numeric, code for harmonized item. • `year`: Numeric, year of observation. • `value`: Numeric, value of observation. • `type`: String, `"simple"` or `"1:n"`.
`...`	Additional grouping columns provided as bare names.

Value

A tibble with columns:

- `item_code`: Numeric, code for harmonized item.
- `year`: Numeric, year of observation.
- `value`: Numeric, summed value of observation.
- and any additional grouping columns.

Examples

```
# Simple-only data (no 1:n rows)
df_simple <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type,
  "wheat", 1, 2000, 5, "simple",
  "barley", 2, 2000, 3, "simple",
  "oats", 2, 2000, 2, "simple"
)
harmonize_interpolate(df_simple)

# Mixed simple + 1:n data
df_mixed <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type,
  "wheatrice", 1, 2000, 20, "1:n",
  "wheatrice", 2, 2000, 20, "1:n",
  "wheat", 1, 2000, 8, "simple",
  "rice", 2, 2000, 12, "simple"
)
harmonize_interpolate(df_mixed)

# Multiple years with share interpolation
# Shares are known in 2000 and 2002; 2001 is interpolated.
```

```
df_years <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type,
  "wheat", 1, 2000, 6, "simple",
  "rice", 2, 2000, 4, "simple",
  "wheatrice", 1, 2001, 10, "1:n",
  "wheatrice", 2, 2001, 10, "1:n",
  "wheat", 1, 2002, 8, "simple",
  "rice", 2, 2002, 2, "simple"
)
harmonize_interpolate(df_years)

# With extra grouping columns
df_grouped <- tibble::tribble(
  ~item, ~item_code_harm, ~year, ~value, ~type, ~country,
  "wheat", 1, 2000, 6, "simple", "usa",
  "rice", 2, 2000, 4, "simple", "usa",
  "wheatrice", 1, 2001, 10, "1:n", "usa",
  "wheatrice", 2, 2001, 10, "1:n", "usa",
  "wheat", 1, 2002, 8, "simple", "usa",
  "rice", 2, 2002, 2, "simple", "usa",
  "wheat", 1, 2002, 8, "simple", "germany"
)
harmonize_interpolate(df_grouped, country)
```

`harmonize_simple` *Harmonize rows labeled "simple" by summing values*

Description

Sum `value` for rows where `type == "simple"`. This covers both 1:1 and N:1 item mappings, since in both cases the values are simply summed. The results are grouped by `item_code_harm`, `year` and any additional grouping columns supplied via `...`.

Usage

```
harmonize_simple(data, ...)
```

Arguments

`data`	A data frame containing at least columns: • `item_code_harm`: Numeric, code for harmonized item. • `year`: Numeric, year of observation. • `value`: Numeric, value of observation. • `type`: String, harmonization type. Only `"simple"` rows are used.
`...`	Additional grouping columns supplied as bare names.

Value

A tibble with columns:

- `item_code_harm`: Numeric, code for harmonized item.
- `year`: Numeric, year of observation.
- `value`: Numeric, summed value of observation.
- and any additional grouping columns.

Examples

```
# 1:1 mapping: one original item -> one harmonized code
df_one_to_one <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type,
  1, 2000, 10, "simple",
  2, 2000, 3, "simple",
  1, 2001, 12, "simple",
  2, 2001, 5, "simple"
)
harmonize_simple(df_one_to_one)

# N:1 mapping: multiple items map to the same code
df_many_to_one <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type,
  1, 2000, 4, "simple",
  1, 2000, 6, "simple",
  2, 2000, 3, "simple"
)
harmonize_simple(df_many_to_one)

# With an extra grouping column (e.g. country)
df_grouped <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type, ~country,
  1, 2000, 4, "simple", "usa",
  1, 2000, 6, "simple", "usa",
  1, 2000, 9, "simple", "germany",
  2, 2000, 3, "simple", "usa"
)
harmonize_simple(df_grouped, country)

# Rows with type != "simple" are ignored
df_mixed <- tibble::tribble(
  ~item_code_harm, ~year, ~value, ~type,
  1, 2000, 10, "simple",
  1, 2000, 99, "1:n",
  2, 2000, 3, "simple"
)
harmonize_simple(df_mixed)
```

`human_kgn_cap_reference`
Human-population nitrogen rate per urban inhabitant.

Description

The calibration nitrogen per URBAN inhabitant, `human_n_reference$human_n_gg * 1e6` over the calibration country's urban population, at each human_n_reference benchmark year a verified rate could be computed for. It is the rate `build_human_n()` applies under `population_basis = "urban"`. See `data-raw/build_human_kgn_cap.R` for the derivation: every human_n_reference benchmark year, including 1860, 1900 and 1950, has its own verified denominator. Most rows use real gridded HYDE baseline-scenario urban population (summed over the calibration country's cell_polity footprint from `whep::build_cell_polity()`); the 2018, 2020 and 2022 rows keep the World Bank `SP.URB.TOTL` urban-population denominator instead, because the local HYDE mirror used to build this table only extends through 2017. Every row is real, verified data; not all rows share the same source.

`urban_kgn_cap_reference` is the deprecated former name of this table, kept for one release. It holds the same rows, with `human_kgn_cap` under its former name `urban_kgn_cap`.

Usage

```
human_kgn_cap_reference

urban_kgn_cap_reference
```

Format

A tibble with columns:

year Benchmark calendar year.

human_kgn_cap Human-population nitrogen applied to agriculture per urban inhabitant (kg N per person per year).

Source

Derived from human_n_reference and the calibration country's urban population: HYDE baseline-scenario gridded population (1860-2016 rows) and World Bank indicator `SP.URB.TOTL` (2018-2022 rows); see `data-raw/build_human_kgn_cap.R`.

Examples

```
human_kgn_cap_reference
```

`human_kgn_cap_total_reference`
Human-population nitrogen rate per inhabitant.

Description

The calibration nitrogen per inhabitant, `human_n_reference$human_n_gg * 1e6 / calibration_population`, where `calibration_population` is the calibration country's UN WPP 2024 total population. It is the rate `build_human_n()` applies under its default `population_basis = "total"`, whose population level is the UN WPP total (`build_total_population_grid()`), and the counterpart of human_kgn_cap_reference (kg N per URBAN inhabitant). Pairing each population with the rate on its own basis is what lets either basis regenerate its calibration total; applying the per-urban-inhabitant rate to a total population would scale the term by the inverse urban fraction (global WPP total over HYDE urban population: 3.02 in 1960, 2.01 in 2010, 1.91 in 2017).

The series starts at 1950, the first year UN WPP covers, so the 1860 and 1900 benchmarks of human_n_reference have no row here. The same calibration series is applied as a global default under either basis; see `build_human_n()`.

Usage

```
human_kgn_cap_total_reference
```

Format

A tibble with columns:

year Benchmark calendar year.

human_kgn_cap Human-population nitrogen applied to agriculture per inhabitant (kg N per person per year).

calibration_population The calibration country's total population that year (persons), the denominator, from `read_wpp_population()` with `by = "total"`.

Source

Derived from human_n_reference and United Nations, Department of Economic and Social Affairs, Population Division (2024), World Population Prospects 2024, medium variant (`WPP2024_PopulationByAge5GroupSex_Medium.csv.gz`); see `data-raw/build_human_kgn_cap.R`.

Examples

```
human_kgn_cap_total_reference
```

`human_n_reference` *Historical human-population nitrogen applied to agriculture.*

Description

National-total nitrogen from a population's municipal solid waste, sewage sludge and human excreta actually applied to agricultural land, at benchmark years: the Spanish series taken as reference. It calibrates the per-capita rates human_kgn_cap_total_reference and human_kgn_cap_reference that `build_human_n()` applies everywhere, as a documented placeholder (see that function's Details) rather than a globally calibrated estimate.

`urban_n_reference` is the deprecated former name of this table, kept for one release. It holds the same rows, with `human_n_gg` under its former name `urban_n_gg`.

Usage

```
human_n_reference

urban_n_reference
```

Format

A tibble with columns:

area_code Numeric FAOSTAT area code, as everywhere else in this package; currently a single national series. The vendored CSV records an ISO3 string and it is resolved to a code through polity_area_crosswalk at build time, so this series joins to area-keyed tables without a hand conversion. It held the string itself until 0.3.0.9000, which made it the one column named `area_code` in this package that was not one (whep#401). Keep using it to join; it is an aggregation key, not the territory's identity.

polity_code The polity the row's territory IS: the identifier every place in WHEP that names a territory is meant to carry (whep#458). Resolved at build time against the polity active in that benchmark year, so a series spanning a succession would carry more than one code. Added alongside `area_code` rather than replacing it (whep#495), so callers keying on the numeric are unaffected.

year Benchmark calendar year.

human_n_gg National-total human-population nitrogen applied to agriculture (Gg N/year).

Source

Aguilera, E. (WHEP project team). Own estimation from unpublished project data (no public DOI): workbook `Urban_waste.xlsx`, sheet `UrbanN`, and its update `UrbanN_update.csv`.

Examples

```
human_n_reference
```

hwsd_texture_usda *HWSD topsoil USDA texture code to texture-class crosswalk.*

Description

Maps the HWSD (Harmonized World Soil Database) topsoil USDA texture code `t_usda_tex` (the `TEXTURE_USDA` field, integers 1 to 13) to the canonical USDA texture-class name keying soil_hydraulic_by_texture. `read_soil_hydraulic()` uses it to translate each grid cell's dominant HWSD texture code into its hydraulic properties. HWSD splits clay into a heavy-clay (code 1) and a light-clay (code 3) class; both map to the single USDA `"clay"` class of the standard 12-class system, matching how the WHEP HWSD preparation pipeline collapses them.

Usage

```
hwsd_texture_usda
```

Format

A tibble with columns:

t_usda_tex HWSD topsoil USDA texture code (integer 1 to 13).

usda_texture_class Canonical USDA texture class (snake_case), one of the classes in soil_hydraulic_by_texture.

Source

HWSD topsoil USDA texture-code legend as documented for the local HWSD extract in the WHEP spatial-input preparation pipeline (`inst/scripts/prepare_spatialize_all.R`), following the Harmonized World Soil Database version 2.0 `D_TEXTURE_USDA` class ordering. FAO & IIASA (2023). *Harmonized World Soil Database version 2.0*. Rome and Laxenburg. doi:10.4060/cc3823en.

Examples

```
hwsd_texture_usda
```

indirect_n2o_ef *Indirect N2O emission factors.*

Description

Parameters for indirect N2O emissions from manure management: EF4 (volatilization), EF5 (leaching), FracGasMS, FracLeach, one block per IPCC edition. The manure engine reads the block its `indirect_n2o_source` option names (see `apply_management_losses()`); the default is `"ipcc_2019"`.

Verified against both editions of Vol 4, Ch 11, Table 11.3 (whep#1245):

- `ef4_volatilization` 0.010 in both (aggregated value).
- `ef5_leaching` 0.011 in the 2019 Refinement, 0.0075 in the 2006 Guidelines.
- `frac_leach` is FracLEACH-(H): 0.24 in the 2019 Refinement, 0.30 in the 2006 Guidelines. Both editions apply it only where leaching occurs (wet climates in 2019; where the soil water-holding capacity is exceeded in 2006) and take it as zero elsewhere; WHEP applies it everywhere.

Until whep#1245 the table cited 2019 while holding the 2006 EF5 and FracLEACH-(H).

- `frac_gasm` is FracGASM, the volatilised fraction of organic N applied and of dung and urine deposited by grazing animals: 0.21 in the 2019 Refinement, 0.20 in the 2006 Guidelines (whep#1365). The default per-MMS indirect manure N2O prices pasture N at it.

`frac_gasms` 0.20 is the same in both blocks and is **assumed, unverified**: Table 10.22 of either edition publishes FracGasMS per animal category and manure system, not one number, and 0.20 equals the 2006 FracGASM of Table 11.3 (the 2019 FracGASM is 0.21). Since whep#1365 only the `indirect_n2o_fractions = "single"` form reads it; the default reads the per-system Table 10.22 values instead.

Usage

```
indirect_n2o_ef
```

Format

A tibble with `edition` (`"ipcc_2019"` or `"ipcc_2006"`), `parameter`, `value`, `description`.

Source

IPCC 2019 Refinement, Vol 4, Ch 11, Table 11.3 (Updated), p. 11.26; IPCC 2006 Guidelines, Vol 4, Ch 11, Table 11.3, p. 11.24; Vol 4, Ch 10, Table 10.22 of either edition for FracGasMS.

Examples

```
indirect_n2o_ef
```

`interp_vec` *Interpolate anchor points at arbitrary output positions.*

Description

Vector-level interpolation primitive: anchor positions and values in, one interpolated value per requested output position out. With `log_space = TRUE` an output point is filled in log space (constant compound growth rate) whenever both of its bracketing anchors are finite and strictly positive; every other point falls back to ordinary linear interpolation.

This is the same rule, and the same internal implementation, that `fill_linear()` applies with `log_space = TRUE`. `fill_linear()` is a data-frame gap filler that needs the target rows to already exist; `interp_vec()` is for callers that hold plain vectors and need values at positions with no pre-existing row, or that call the primitive once per gap segment inside their own guard logic.

Usage

```
interp_vec(x, y, xout, log_space = FALSE, rule = 1)
```

Arguments

`x`	Numeric vector of anchor positions, for example years. It does not need to be sorted.
`y`	Numeric vector of anchor values, the same length as `x`.
`xout`	Numeric vector of positions to interpolate at. It does not need to be sorted; results follow the order of `xout`.
`log_space`	Logical. If `TRUE`, each output point bracketed by two finite and strictly positive anchors is interpolated in log space (constant compound growth rate); any other point falls back to linear interpolation. Default: `FALSE`, i.e. linear interpolation everywhere.
`rule`	Either `1` or `2`, passed to `stats::approx()` to handle output positions outside the anchor range. `1` (default) returns `NA` there, `2` carries the nearest anchor value. Log space never applies outside the anchor range.

Details

Anchors with a non-finite position or a missing value are dropped, and anchors sharing a position are averaged, so that the linear and log-space paths always see the same anchor set. Anchors whose value is infinite are kept, but a segment bounded by one is never eligible for log space. If fewer than two usable anchors remain there is nothing to interpolate between, and every element of `y` and `method` is `NA`.

An output position that coincides with an anchor position returns that anchor's value unchanged, labelled `"linear"`, because nothing needed interpolating there. In particular log space never rebuilds an anchor value it was handed, which would perturb it in the last bits.

Value

A list of two vectors, each as long as `xout`:

- `y`: the interpolated values.
- `method`: `"loglinear"` where the value came from log space, `"linear"` where it came from linear interpolation, and `NA_character_` where no value could be produced.

See Also

`fill_linear()`

Examples

```
# Constant compound growth: the 2005 midpoint of 1 and 1024 is 32, whereas
# linear interpolation returns the arithmetic midpoint 512.5.
interp_vec(c(2000, 2010), c(1, 1024), xout = 2005, log_space = TRUE)
interp_vec(c(2000, 2010), c(1, 1024), xout = 2005)

# A non-positive anchor makes log space undefined, so the point stays linear.
interp_vec(c(2000, 2010), c(0, 10), xout = 2005, log_space = TRUE)

# Several output positions at once, from an unsorted anchor set. With
# `rule = 2` the position beyond the last anchor carries that anchor value.
interp_vec(
  x = c(2010, 2000, 2005),
  y = c(400, 100, 200),
  xout = c(2002, 2007, 2015),
  log_space = TRUE,
  rule = 2
)
```

`ipcc_2006_enteric_ef` *IPCC 2006 Tier 1 enteric emission factors.*

Description

Table 10.11 (2006): Tier 1 regional EFs for enteric fermentation, with the non-cattle species of Table 10.10 appended under a `"Global"` region.

Usage

```
ipcc_2006_enteric_ef
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`.

Source

IPCC 2006, Vol 4, Ch 10, Table 10.11 for cattle and Table 10.10 (developed-countries column) for the non-cattle rows. Two departures from the published table: Oceania dairy cattle is stored as 90 where Table 10.11 gives 100, and the published table groups Africa **and** the Middle East in one row (46 dairy / 31 other) which is repeated here as two regions. The Indian Subcontinent row of Table 10.11 (58 dairy / 27 other) is absent. Tracked in whep#601.

Examples

```
ipcc_2006_enteric_ef
```

`ipcc_2006_manure_ef` *IPCC 2006 Tier 1 manure emission factors.*

Description

Table 10.14 (2006): Tier 1 regional EFs for manure CH4.

Usage

```
ipcc_2006_manure_ef
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`, `temp_zone`.

Source

IPCC 2006, Vol 4, Ch 10, Table 10.14 for cattle, swine and buffalo and Table 10.15 for sheep, goats and poultry. Table 10.14 is resolved per degree Celsius, and the value taken for a `temp_zone` is not always the bound of that class, nor always present in the row: North American dairy cows 53 is the 12 degree column rather than the 48 of the cool class, North American other cattle 2 is the 15 degree value where the whole cool class of that row is 1, Asian dairy cows 16 is the 18 degree column rather than the 31 of the warm class, Latin American dairy cows 1 is the cool value where the warm class gives 2, and Western European dairy cows 20 appears in no column of that row (its cool value is 21). Tracked in whep#601.

Examples

```
ipcc_2006_manure_ef
```

`ipcc_2006_mcf_temp` *IPCC 2006 MCF by temperature.*

Description

Table 10.17 (2006): MCF values by MMS type and annual temperature.

Usage

```
ipcc_2006_mcf_temp
```

Format

A tibble with `system`, `temp_c`, `mcf_percent`.

Source

Loosely IPCC 2006, Vol 4, Ch 10, Table 10.17, but re-resolved onto a 10/15/20/25 degree Celsius grid the published table does not use, and the `temp_c == 25` value of every one of the four rows appears in no column of Table 10.17. Verified against the published table:

- Liquid/slurry is the only row Table 10.17 resolves per degree, as 17/19/20/22/25/27 percent for 10 to 15 degrees rising to 65 percent at 25 (without a natural crust cover). Stored 17 is its 10 degree column, but 25 is the 14 degree column rather than the 27 of 15 degrees, 35 is the 18 degree column rather than the 42 of 20 degrees, and 48 is in no column at all (46 at 21 degrees, 50 at 22).
- Pasture/range/paddock, daily spread and solid storage are published only by climate class – 1.0/1.5/2.0, 0.1/0.5/1.0 and 2.0/4.0/5.0 percent for cool/temperate/warm. Stored 10 and 15 degrees take the cool and temperate figures, but 20 degrees takes the warm figure although 20 degrees falls inside the published temperate class, and 25 degrees takes 2.5, 1.5 and 6 percent, none of which the table publishes. **Assumed, unverified.** No function in `R/` reads this object – the Tier 2 manure path uses climate_mcf and the Tier 1 path ipcc_2019_mcf_manure – so nothing published depends on it. Tracked in whep#601.

Examples

```
ipcc_2006_mcf_temp
```

ipcc_2019_bo *IPCC 2019 Bo values (Table 10.16A).*

Description

Maximum CH4 producing capacity of manure (m3 CH4/kg VS). Dairy cattle (0.24) differs from other cattle (0.18).

Usage

```
ipcc_2019_bo
```

Format

A tibble with `category`, `bo_m3_kg_vs`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.16A (Updated). That edition has no Table 10.16: its list of tables jumps from Table 10.15 (Updated), the per-head manure CH4 factors for deer, reindeer, rabbits, ostrich and fur-bearing animals, straight to Table 10.16A. Table 10.16 is the 2006 Guidelines' number for that deer/reindeer/rabbit/fur-bearing table (0.22/0.36/0.08/0.68 kg CH4 head-1 yr-1, no ostrich row), and the 2006 edition has no Bo table at all – its Bo defaults live in Annex 10A.2. Every row matches its high-productivity column except `"Swine - Breeding"` 0.27: Table 10.16A publishes one swine Bo (0.48 North America, 0.45 other high-productivity regions, 0.29 low productivity) and the 2006 Annex 10A.2 derivation tables give breeding swine the same Bo as market swine, so 0.27 appears in neither edition; it coincides with the North American *market swine volatile-solids rate* of 0.27 kg VS head-1 day-1 in 2006 Annex 10A.2. `"Other Cattle"` 0.18 is the Western European non-dairy column (North America is 0.19, Eastern Europe and Oceania 0.17). Tracked in whep#601.

Examples

```
ipcc_2019_bo
```

ipcc_2019_cfi *IPCC 2019 Cfi values (Table 10.4).*

Description

Net energy maintenance coefficients (MJ/day/kg^0.75). Dairy (lactating) cattle use 0.386; non-dairy 0.322.

Usage

```
ipcc_2019_cfi
```

Format

A tibble with `category`, `subcategory`, `cfi_mj_day_kg075`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.4 (Updated), which repeats the 2006 Guidelines values and adds the goat row (0.315). Two rows published in both editions are absent here, and the `"Non-lactating/Bulls"` row conflates them: intact bulls take 0.370, not the 0.322 of non-lactating cows, steers and juveniles, and lambs under one year take 0.236 rather than the 0.217 of mature sheep. Tracked in whep#601.

Examples

```
ipcc_2019_cfi
```

`ipcc_2019_enteric_ef_cattle`

IPCC 2019 enteric EF for cattle.

Description

Tier 1 enteric fermentation emission factors for cattle by region (kg CH4/head/yr). Regional cattle factors are Table 10.11 in both the 2006 Guidelines and the 2019 Refinement; Table 10.10 holds the non-cattle species (see ipcc_2019_enteric_ef_other).

Usage

```
ipcc_2019_enteric_ef_cattle
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`.

Source

Predominantly the 2006 Guidelines, Vol 4, Ch 10, Table 10.11, not the 2019 Refinement's Table 10.11 (Updated). Verified against both published tables:

- 2006 values, differing from the 2019 Refinement: North America 128/53 (2019: 138/64), Western Europe 117/57 (126/52), Eastern Europe 99/58 (93/58), Latin America 72/56 (87/56), Asia 68/47 (78/54), Africa 46/31 (76/52), Middle East other cattle 31 (60).
- Matching neither edition: Oceania dairy 90 (2006: 100; 2019: 93), Middle East dairy 63 (2006 groups Africa and the Middle East at 46; 2019: 76), Indian Subcontinent 68/47 (2006: 58/27; 2019: 73/46).

- The "Global" fallback row (80/47) appears in no IPCC table in either edition. **Assumed, unverified.** The 2019 Refinement also moved buffalo into this table (78 Western Europe, 68 Eastern Europe / Latin America / Asia, 81 Africa, 67 Middle East, 85 Indian Subcontinent), which is not reflected here. Since whep#601 Tier 1 enteric CH4 reads the as-published 2019 factors in ipcc_enteric_ef_tier1 by default. This table is read only under the enteric_ef_source = "as_shipped" manure-engine option (see manure_engine_options), and as the fallback for rows the 2019 edition prices at nothing.

Examples

```
ipcc_2019_enteric_ef_cattle
```

---

ipcc_2019_enteric_ef_other

IPCC 2019 enteric EF for non-cattle.

Description

Tier 1 enteric fermentation emission factors for non-cattle species (kg CH4/head/yr). Non-cattle species are Table 10.10 in both the 2006 Guidelines and the 2019 Refinement; regional cattle factors are Table 10.11 (see ipcc_2019_enteric_ef_cattle).

Usage

```
ipcc_2019_enteric_ef_other
```

Format

A tibble with category, ef_kg_head_yr.

Source

The 2006 Guidelines, Vol 4, Ch 10, Table 10.10, developed-countries column (buffalo 55, sheep 8, goats 5, camels 46, horses 18, mules and asses 10, swine 1.5), not the 2019 Refinement. The Refinement's Table 10.10 (Updated) splits every ruminant and swine factor by productivity system (sheep 9 high / 5 low, goats 9 / 5, swine 1.5 / 1.0), leaves camels, horses and mules unchanged, and moves buffalo out of this table into the regional Table 10.11. Poultry is stored as 0; both editions say "insufficient data for calculation", so the zero is a project choice rather than a published factor. Since whep#601 Tier 1 enteric CH4 reads ipcc_enteric_ef_tier1 by default; this table is read only under enteric_ef_source = "as_shipped" and as the fallback for rows the 2019 edition prices at nothing (poultry, and buffalo in North America and Oceania).

Examples

```
ipcc_2019_enteric_ef_other
```

ipcc_2019_manure_ch4_ef_cattle

IPCC 2019 manure CH4 EF for cattle.

Description

Tier 1 manure management CH4 emission factors for cattle by region (kg CH4/head/yr).

Usage

```
ipcc_2019_manure_ch4_ef_cattle
```

Format

A tibble with `region`, `category`, `ef_kg_head_yr`.

Source

Not the 2019 Refinement. Its Table 10.14 (Updated) publishes manure CH4 per kilogram of volatile solids (g CH4 kg VS-1), by productivity class and ten climate zones; the Refinement contains no per-head Tier 1 manure CH4 table at all (its only per-head CH4 tables are the enteric Tables 10.10/10.11 and Table 10.15 for deer, reindeer, rabbits, ostrich and fur-bearing animals). The per-head quantity stored here is the form of the 2006 Guidelines Table 10.14, but the values do not match it either (North American dairy cattle 27/42/60 for cool/temperate/warm against 48/78/112 in 2006; Latin American dairy cattle 47 against 2). **The provenance of these values is unknown and unverified**; tracked in whep#601.

Examples

```
ipcc_2019_manure_ch4_ef_cattle
```

ipcc_2019_manure_ch4_ef_other

IPCC 2019 manure CH4 EF for non-cattle.

Description

Tier 1 manure management CH4 emission factors for non-cattle species (kg CH4/head/yr).

Usage

```
ipcc_2019_manure_ch4_ef_other
```

Format

A tibble with `category`, `ef_kg_head_yr`.

Source

2006 Guidelines, Vol 4, Ch 10, Tables 10.14 (buffalo, swine) and 10.15 (sheep, goats, poultry, horses, mules and asses, camels), not the 2019 Refinement, which publishes no per-head Tier 1 manure CH4 table. The temperature column each value is taken from varies by species (sheep 0.19 and goats 0.13 are the developed-country cool column, while horses 1.64, mules 0.90 and camels 1.92 are the developing-country temperate column). For buffalo 2.00 and swine 6.00 the source region is not recorded and cannot be recovered from the value: 2006 Table 10.14 is resolved by region and by degree Celsius, and more than one cell carries each of those two numbers. Tracked in whep#601.

Examples

```
ipcc_2019_manure_ch4_ef_other
```

`ipcc_2019_mcf_manure` *IPCC 2019 MCF for manure management.*

Description

Methane conversion factors (percent) by manure management system and climate zone, using `"All"` for the systems that take a single factor.

Usage

```
ipcc_2019_mcf_manure
```

Format

A tibble with `system`, `climate_zone`, `mcf_percent`.

Source

Predominantly the 2006 Guidelines, Vol 4, Ch 10, Table 10.17, whose cool/temperate/warm structure this table follows. The 2019 Refinement's Table 10.17 (Updated) is resolved by ten climate zones and by liquid-system retention time instead, and differs in level: it gives a single 0.47 percent for pasture/range/paddock against 1.0/1.5/2.0 here, and 1.0/2.0/2.5 for static-pile and passive-windrow composting against 0.5/0.5/0.5 and 1.0/1.0/1.5 here. Some cells match neither edition: dry lot 1.5/2.5/4.0 (both editions give 1.0/1.5/2.0), intensive-windrow composting 0.5/0.5/0.5 (both give 0.5/1.0/1.5), passive-windrow composting 1.0/1.0/1.5 (2006 gives 0.5/1.0/1.5, the 2019 Refinement 1.0/2.0/2.5), pit storage under one month 3/3/5 (2006 gives 3/3/30) and liquid/slurry with crust warm 47 percent, which is in no column of that 2006 row (its warm class runs 44 percent at 26 degrees, 48 at 27 and 50 at 28 or above) and in no cell of the 2019 table. `"Anaerobic Digester"` 0 is not a published default in either edition: 2006 Table 10.17 gives the range 0-100 percent and requires the compiler to calculate it, and the 2019 Refinement resolves it into six leakage-and-storage classes spanning 1.00 to 13.17 percent. **Assumed, unverified.** Where a 2006 row is resolved per degree Celsius the value taken is not always the mid-point of the

class: uncovered anaerobic lagoon temperate 73 percent is the 14 degree column, not the 78 percent of 20 degrees, and liquid/slurry without crust and pit storage over one month take 35 percent for temperate, which is the 18 degree column rather than the 42 percent of 20 degrees. Tracked in whep#601.

Examples

```
ipcc_2019_mcf_manure
```

`ipcc_2019_n_excretion`

IPCC 2019 nitrogen excretion rates.

Description

Default nitrogen excretion by animal category and region, stored as annual excretion per head (kg N/head/yr). That is the form the Tier 1 manure N2O path consumes, and the same quantity the Tier 2 path derives from the energy balance.

Usage

```
ipcc_2019_n_excretion
```

Format

A tibble with `region`, `category`, `nex_kg_n_head_yr`.

Source

Unverified. IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.19 (Updated) publishes the excretion *rate* per 1000 kg animal mass per day, not an annual per-head amount, and the values stored here do not follow from it. The Refinement does supply the missing conversion factor: Table 10A.1 (New) gives the regional typical weight of dairy cattle, so rate x weight x 365 is derivable and gives 140 kg N/head/yr for North America (0.59 x 650 kg x 365 / 1000) against the 105 stored, 118 for Western Europe (100 stored), 84 for Eastern Europe (80), 128 for Oceania (80), 72 for Latin America (50), 62 for Asia (50), 42 for Africa (40), 64 for the Middle East (40) and 68 for the Indian Subcontinent (50). Other cattle would need the cohort population mix of Table 10A.2 (New) to be weighted the same way. Tracked in whep#601.

Examples

```
ipcc_2019_n_excretion
```

`ipcc_2019_n2o_ef_direct`

IPCC 2019 direct N2O emission factors.

Description

Table 10.21: EF3 values (kg N2O-N per kg N excreted) by manure management system.

Usage

```
ipcc_2019_n2o_ef_direct
```

Format

A tibble with `system`, `ef_kg_n2o_n_per_kg_n`.

Source

Mixed, and not consistently the 2019 Refinement's Table 10.21. Verified against both editions of Vol 4, Ch 10, Table 10.21:

- Matching both editions: liquid/slurry with crust 0.005, in-vessel composting 0.006, poultry with and without litter 0.001.
- Matching the 2006 Guidelines but not the 2019 Refinement: solid storage 0.005 (2019: 0.010), static-pile composting 0.006 (2019: 0.010), passive-windrow composting 0.01 (2019: 0.005), anaerobic digester 0 (2019: 0.0006).
- Matching neither edition: daily spread 0.01 and liquid/slurry without crust 0.002 and uncovered anaerobic lagoon 0.001 (all three are 0 in both editions), dry lot 0.005 (0.02 in both), intensive-windrow composting 0.006 (2019: 0.005; 2006: 0.1). Pasture/range/paddock is not in Table 10.21 in either edition, which defers it to Ch 11. Its stored 0.01 is the 2006 Ch 11 Table 11.1 EF3PRP,SO for sheep and other animals; the 2019 Refinement's Table 11.1 (Updated) gives 0.004 for cattle, poultry and pigs and 0.003 for sheep and other animals. Four further rows are WHEP composite or fallback labels with no counterpart system in Table 10.21 of either edition, so no published value backs them: `"Liquid/Slurry"` 0.002 (the published liquid/slurry EF3 is 0.005 with a crust and 0 without; 0.002 is the pit-storage value), `"Solid Storage and Dry Lot"` 0.005 (the 2006 solid-storage value, while dry lot is 0.02 in both editions), `"Burned for Fuel"` 0 (both editions give no EF3 here and report those emissions under fuel combustion or waste incineration instead) and `"Other"` 0.005. **Assumed, unverified.** Since whep#601 the manure engine reads the as-published 2019 EF3 in ipcc_manure_ef3 by default; this table backs only that dataset's `"as_shipped"` edition.

Examples

```
ipcc_2019_n2o_ef_direct
```

`ipcc_2019_soil_n_ef` *IPCC 2019 Tier 1 soil nitrogen emission and leaching factors.*

Description

IPCC 2019 Refinement Tier 1 default factors for managed soils: direct N2O emission factors for N inputs (EF1), flooded rice (EF1FR) and urine and dung deposited by grazing animals (EF3PRP), the indirect N2O factors for volatilisation and redeposition (EF4) and for leaching and runoff (EF5), and the leaching fraction FracLEACH-(H). Many are given as an aggregated value and as wet and dry climate values. A climate is wet if annual precipitation / potential evapotranspiration (PET) is greater than 1 in temperate and boreal climates, or annual precipitation is greater than 1000 mm in tropical climates (Table 11.1, footnote 4); otherwise it is dry.

Usage

```
ipcc_2019_soil_n_ef
```

Format

A tibble with columns:

parameter IPCC parameter name: `"EF1"`, `"EF1FR"`, `"EF3PRP_CPP"`, `"EF3PRP_SO"`, `"EF4"`, `"EF5"` or `"FracLEACH_H"`.

disaggregation Climate or water-management stratum the value applies to, e.g. `"aggregated"`, `"wet"`, `"dry"`.

value The factor.

unit Unit of the factor.

source Locator of the value in the IPCC 2019 Refinement.

Source

IPCC (2019), 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol. 4, Chapter 11: Table 11.1 (Updated) p. 11.13; Table 11.3 (Updated) p. 11.26; Section 11.2.2.2 p. 11.25.

Examples

```
ipcc_2019_soil_n_ef
```

`ipcc_2019_ym` *IPCC Ym values.*

Description

Methane conversion rate (% GE) by species and feed situation.

- Cattle feedlot (>90% concentrate): 3.0%.
- Sheep: a single 6.7%, irrespective of feed quality (no body-weight split).

Usage

```
ipcc_2019_ym
```

Format

A tibble with `category`, `feed_situation`, `ym_percent`.

Source

Mixed across editions.

- Sheep 6.7% and goats 5.5% are the 2019 Refinement, Vol 4, Ch 10, Table 10.13 (Updated).
- Cattle and buffalo 6.5% on pasture/range and mixed rations are the 2006 Guidelines Table 10.12, which gives 6.5% for every non-feedlot cattle and buffalo class. The 2019 Refinement's Table 10.12 (Updated) resolves cattle and buffalo Ym by production level and feed digestibility instead: 5.7 / 6.0 / 6.3 / 6.5 for dairy cows by yield class, 7.0 for >75 percent forage non-dairy, 6.3 for mixed rations, 4.0 for grain feedlots and 3.0 for steam-flaked-corn feedlots. The stored feedlot 3.0% is therefore the 2006 ">=90 percent concentrate" value, which in the 2019 Refinement applies only to the steam-flaked-corn case.
- Camels 5.0% appears in no IPCC table. Both editions instead direct compilers to reuse the other-cattle or buffalo Ym for camels, which would be 6.5%. **Assumed, unverified.** Tracked in whep#601.

Examples

```
ipcc_2019_ym
```

`ipcc_enteric_ef_tier1` *IPCC Tier 1 enteric emission factors, as published.*

Description

Tier 1 enteric fermentation emission factors (kg CH4/head/yr) as the 2019 Refinement publishes them, which the `ipcc_2019_enteric_ef_*` tables did not hold (whep#601). Tier 1 enteric CH4 reads this table by default; the `enteric_ef_source` manure-engine option selects it (`"ipcc_2019"`) or the older tables (`"as_shipped"`), see manure_engine_options.

A category has one of three shapes:

- **regional** (`region` is an IPCC region, `productivity == "All"`): dairy cattle, other cattle and buffalo, Table 10.11 (Updated). Only the Tier 1 column is carried; the high/low productivity rows the table adds for five regions are Tier 1a, which needs the head count split by productivity system. North America and Oceania publish no buffalo factor ("no buffalo herds").
- **split by productivity** (`region == "All"`, `productivity` `"High"` or `"Low"`): sheep, goats and swine, Table 10.10 (Updated). Its footnote 1 says that "for all regions other than North America, Europe and Oceania the Tier 1 default values are the low productivity EFs", so Tier 1 takes `"High"` in North America, Western Europe, Eastern Europe and Oceania and `"Low"` in every other IPCC region.
- **flat** (`region == "All"`, `productivity == "All"`): horses, camels, mules and asses, and the deer, ostrich and llama/alpaca rows of the same table.

Poultry has no row: both editions say "insufficient data for calculation". A row the edition prices at nothing – poultry, buffalo in North America or Oceania, or a regional or productivity-split species on a row whose IPCC region does not resolve – keeps the as-shipped factor, and `method_enteric` says so (`enteric_ef_as_shipped_fallback`).

Usage

```
ipcc_enteric_ef_tier1
```

Format

A tibble with one row per `edition`, `region`, `category` and `productivity`:

- `edition`: `"ipcc_2019"`.
- `region`: IPCC Tier 1 region, or `"All"`.
- `category`: `"Dairy Cattle"`, `"Other Cattle"`, `"Buffalo"` or a general species name.
- `productivity`: `"All"`, `"High"` or `"Low"`.
- `ef_kg_head_yr`: emission factor, kg CH4 per head per year.
- `ipcc_table`: the table the value was read from.

Measured effect

On FAOSTAT 2020 heads at national grain, Tier 1 enteric CH4 goes from 110.8 Tg (as shipped) to 125.3 Tg: cattle 78.58 to 90.98 Tg, buffalo 11.21 to 16.55 Tg, sheep 10.30 to 7.31 Tg (low productivity outside the four high regions), swine 1.41 to 1.08 Tg and goats 5.47 to 5.56 Tg. Every row resolved an IPCC region; the only fallback rows were poultry and 163 buffalo head in North America and Oceania.

Source

IPCC. 2019. *2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories*, Vol 4, Ch 10, Table 10.10 (Updated), p. 10.38, and Table 10.11 (Updated), pp. 10.40-10.43 (PDF md5 `c1784e747af9bb307e93f4170c12679a`, downloaded from `ipcc-nggip.iges.or.jp` and checksummed on 2026-10-07). Every value is a transcription.

Examples

```
ipcc_enteric_ef_tier1
```

`ipcc_manure_ef3` *IPCC direct manure N2O emission factors (EF3), by edition.*

Description

EF3 (kg N2O-N per kg N) for the six manure-management labels WHEP's manure engine carries, in two editions. The `ef3_source` option of the manure engine (see manure_engine_options) and of `apply_management_losses()` selects one; `"ipcc_2019"` is the default.

- `"ipcc_2019"`: the 2019 Refinement, Vol 4, Ch 10, Table 10.21 (Updated): daily spread 0, solid storage 0.010, uncovered anaerobic lagoon 0, poultry manure with litter 0.001. Liquid/slurry is published as 0.005 with a natural crust, 0 without and 0.005 covered; the first two are carried as `"Liquid/Slurry - With Crust"` and `"Liquid/Slurry - No Crust"`, and the engine's single `"Liquid/Slurry"` label takes their mix at the crust share the `slurry_crust` option selects: `"half"` (default, 0.0025), `"none"` (0) or `"all"` (0.005). Half is GLEAM 2.0's assumption: Supplement S1 footnotes the "Liquid slurry*" of its pig and chicken tables (Tab. 4.8-4.11) "50% of this manure management system is assumed to present natural crust". Its cattle and buffalo tables (Tab. 4.2-4.6) state no share, so the half is extended to them by assumption. Pasture/range/paddock is deferred to Ch 11, whose Table 11.1 (Updated) gives the aggregated EF3PRP 0.004 for "cattle (dairy, non-dairy and buffalo), poultry and pigs" (`animal_class == "cattle_poultry_pigs"`) and 0.003 for "sheep and 'other animals'" (`"sheep_other"`), the chapter listing goats, horses, mules, donkeys, camels, reindeer and camelids as other animals. Its disaggregated wet/dry pair (0.006 / 0.002) needs a moisture regime per row and is not carried.

- `"as_shipped"`: the values the engine read before whep#601, a crosswalk onto ipcc_2019_n2o_ef_direct, which holds 2006 and unpublished values (see its `@source`). Its `"Liquid/Slurry"` 0.002 is not the no-crust value (both editions print 0 for that) but the EF3 of pit storage below animal confinements. `slurry_crust` does not apply to it. Kept so earlier figures stay reproducible.

`"Solid Storage"` also carries GLEAM's dry-lot shares (see regional_mms_distribution), which Table 10.21 prices at 0.02, not 0.010.

Usage

```
ipcc_manure_ef3
```

Format

A tibble with one row per `edition`, `mms_type` and `animal_class`:

- `edition`: `"ipcc_2019"` or `"as_shipped"`.
- `mms_type`: manure-management system label, plus the two published liquid/slurry crust rows of the 2019 edition.
- `animal_class`: `"all"`, or for 2019 pasture `"cattle_poultry_pigs"` or `"sheep_other"`.
- `ef3`: kg N2O-N per kg N.
- `ipcc_system`: the published row (or the shipped table's row) the value was read from.

Measured effect

On FAOSTAT 2020 heads at national grain, Tier 1 direct manure N2O goes from 1.523 Tg (as shipped) to 1.178 Tg, almost all of it the pasture factor (0.010 to 0.004 / 0.003): keeping pasture at 0.010 would give 1.830 Tg. The liquid/slurry crust share moves it between 1.120 Tg (`"none"`) and 1.236 Tg (`"all"`). In `apply_management_losses()`, which carries no management N2O on the grazing stream, the 2019 edition moves the nitrogen applied to land by -1.4 to +0.6 percent per species under the Global split.

Source

IPCC. 2019. *2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories*, Vol 4, Ch 10, Table 10.21 (Updated), pp. 10.91-10.93 (PDF md5 `c1784e747af9bb307e93f4170c12679a`), and Vol 4, Ch 11, Table 11.1 (Updated), p. 11.13 (PDF md5 `06c11d4f5a2b729564f8b3bd1ba11824`). Both downloaded from `ipcc-nggip.iges.or.jp` and checksummed on 2026-10-07.

Examples

```
ipcc_manure_ef3
```

`ipcc_tier2_bo_values` *Tier 2 Bo values.*

Description

Maximum CH4 producing capacity by detailed category. Dairy cattle 0.24 vs other cattle 0.18. Under the default `mcf_source = "ipcc_2019"` these price every manure stream except pasture/range/paddock, which takes the 0.19 the Refinement pairs with its pasture MCF (see climate_mcf_ipcc, whep#1137).

Usage

```
ipcc_tier2_bo_values
```

Format

A tibble with `category`, `bo_m3_kg_vs`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.16.

Examples

```
ipcc_tier2_bo_values
```

`ipcc_tier2_energy_coefs`
Tier 2 energy coefficients.

Description

Coefficients for IPCC Tier 2 GE calculation including Cfi (maintenance), Ca (activity), Cp (pregnancy), Cw (work), and energy content of weight gain. Now includes `subcategory` column to differentiate dairy (lactating) vs non-dairy cattle.

Usage

```
ipcc_tier2_energy_coefs
```

Format

A tibble with columns:

category Species (Cattle, Buffalo, Sheep, etc.).

subcategory Dairy, Non-Dairy, or All.

cfi_mj_day_kg075 NEm coefficient (MJ/day/kg^0.75).

ca_pasture Activity coefficient for grazing.

ca_feedlot Activity coefficient for confined.

cp Pregnancy coefficient.

cw Work coefficient.

energy_content_gain_mj_kg Energy per kg gain.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Eq 10.3-10.16, with `cfi_mj_day_kg075` from Table 10.4 (Updated) and `ca_pasture` from Table 10.5 (Updated). Sheep and goats take their own rows in both tables and must not be conflated: Cfi is 0.217 for sheep older than one year and 0.315 for goats, and Ca is 0.0107 for sheep grazing flat pasture and 0.019 for lowland goats. Neither goat value exists in the 2006 Guidelines, whose Table 10.4 has no goat row at all. `ca_pasture` takes the lowland/flat-pasture value as the global default; the published hill and mountain goat coefficient (0.024) is not stored and must be supplied per row by the caller.

Examples

```
ipcc_tier2_energy_coefs
```

`ipcc_tier2_manure_ash` *Tier 2 manure ash content.*

Description

Ash content of manure as percent of dry matter, used in VS calculation (Eq 10.24).

Usage

```
ipcc_tier2_manure_ash
```

Format

A tibble with `category`, `ash_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10.

Examples

```
ipcc_tier2_manure_ash
```

`ipcc_tier2_n_retention` *Tier 2 nitrogen retention fractions.*

Description

Fraction of N intake retained in animal products. Dairy cattle 0.20 vs other cattle 0.07.

Usage

```
ipcc_tier2_n_retention
```

Format

A tibble with `category`, `n_retention_frac`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.20.

Examples

```
ipcc_tier2_n_retention
```

`ipcc_tier2_ym_values` *Tier 2 Ym values.*

Description

Methane conversion rate by species and feed situation for Tier 2 enteric CH4. Includes feedlot distinction and sheep body weight differentiation.

Usage

```
ipcc_tier2_ym_values
```

Format

A tibble with `category`, `feed_situation`, `ym_percent`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10, Table 10.12.

Examples

```
ipcc_tier2_ym_values
```

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_full *Full CBS item table*

Description

Extended item reference table covering all CBS items, including their process and commodity codes, feed type classifications, and default material flow destinations.

Usage

```
items_full
```

Format

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

- `item_cbs`: Name of the CBS item.
- `item_cbs_code`: Numeric CBS item code.
- `comm_code`: Commodity code used in process-based modelling (may contain `"#N/A"` when not applicable).
- `proc_code`: Process code (may contain `"#N/A"` when not applicable).
- `proc`: Process name (may contain `"#N/A"` when not applicable).
- `unit`: Measurement unit (typically `"tonnes"`).
- `group`: Broad item group. Common values include `"Additives"`, `"Crop products"`, `"Crop residues"`, `"Draught"`, `"Fish"`, `"Forestry"`, `"Grass"`, `"Livestock"`, and others.
- `feedtype_graniv`: Feed type classification for granivores (e.g., `"additives"`, `"concentrates"`, `"roughages"`).
- `feedtype_grazers`: Feed type classification for grazers.
- `comm_group`: Sub-group of the commodity (e.g., `"Additives"`, `"Alcohol"`, `"Ethanol"`, `"Oil cakes"`, `"Other processing residues"`).
- `Cat_1`: Primary category label used in material flow accounting.
- `Name_biomass`: Corresponding item name in `biomass_coefs`, enabling joins with the biomass coefficient table; every non-missing value resolves. `NA` means the item has no biomass counterpart: the 22 live-animal items, which are heads rather than biomass, 4000 Animal draught, which is measured in work hours, and 2899 Miscellaneous, a residual aggregate. Upstream, the last two and 2775 Aquatic Plants carry the literal placeholder `"0"`; here 2775 bridges to the `Aquatic Plants` row of biomass_coefs instead (whep#970).
- `dbMFA_items`: Item identifier used in the material flow analysis database.
- `FEDNA`: Name of the FEDNA feed-composition table entry (https://fundacionfedna.org/ingredientes-para-piensos) paired with the item. It is a **stand-in, not a product identity**: it says which feed-table row the item was given, not which substance the item is, and must not be read as evidence of what a row represents (whep#1131). Why each entry was chosen is not recorded upstream. On the 28 `Additives` rows, 4008 Enzimes and 4016 Phytase share PROTEINA DE PATATA (potato protein), 4005 Calcium carbonate and 4013 Minerals share CARBONATO CALCICO, and several name a different substance outright: 4018 Potassium carbonate is CARBONATO SODIO, 4027 Vitamins AC. CITRICO, 4015 Pesticides AC. FORMICO, 4017 Pigment AC. COLZA, 4009 Flavours PROPIONATO SODICO, 4003 Anionic salts MET HIDROXI SAL CALCICA and 4011 Methionine HIDROXI-ANAL MET, a nitrogen-free analogue. Nor does the named entry fix the `biomass_coefs` nitrogen: Methionine and Lysine carry their dry commercial forms (DL-Metionina, L-Lisina HCl), not the entries named here, and the FEDNA entry each additive's nitrogen was actually read from is recorded in `inst/extdata/harmonization/biomass_coefs_additive_sources.csv` (whep#1074). No WHEP function reads this column.

- `default_destiny`: Default CBS use category for this item. One of `"Feed"`, `"Food"`, `"Other_uses"`, `"Processing"`, or `NA`. It is where the final balance books the domestic supply a row's reported destinies leave unexplained. The values come from the `items_full` sheet of afsetools' `Codes_coefs.xlsx`, which records no derivation for them. All nine oilseed cakes (`comm_group == "Oil cakes"`) are `"Feed"`: five of them (2591, 2592, 2594, 2595, 2596) were `"Food"` until whep#1066, unlike the other four.

Source

Derived from FAOSTAT data and internal commodity classification work.

Examples

```
head(items_full)
```

---

items_prim *Primary production items linked to CBS*

Description

Maps FAOSTAT primary production items and crop products to their CBS counterparts, along with farm and labour classifications.

Usage

```
items_prim
```

Format

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

- `item_prod`: Name of the production item (e.g., `"Wheat"`, `"Rice"`).
- `item_prod_code`: FAOSTAT production item code (character).
- `item_cbs`: Name of the corresponding CBS item.
- `item_cbs_code`: Numeric CBS item code.
- `Farm_class`: Farm system classification. Crop items use codes such as `"COP"` (cereals, oilseeds, protein crops), `"Vegetables"`, `"Fruits"`, `"Olive"`, `"Grapevine"`, `"Other_crops"`. Livestock items use `"Dairy_cows"`, `"Cattle"`, `"Monogastric"`, `"Sheep_goats"`, `"Bees"`, `"Game"`. `NA` for non-farm items.
- `Cat_Labour`: Labour category used in agricultural labour analyses.
- `Cat_FAO1`: Top-level FAO commodity category.
- `group`: Item group classification. One of `"Primary crops"`, `"Crop products"`, `"Livestock products"`, `"Grass"`, `"Crop residues"`, `"Scavenging"`, `"Livestock"`.

Source

Derived from FAOSTAT Production data.

Examples

```
head(items_prim)
```

`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.

`items_prod_full` *Full production item table*

Description

Comprehensive reference table for all production items, combining CBS linkages, biomass names, multiple classification schemes, and crop ecological traits.

Usage

```
items_prod_full
```

Format

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

- `item_prod`: Name of the production item.
- `item_prod_code`: FAOSTAT production item code (character).
- `item_cbs`: Name of the corresponding CBS item.
- `item_cbs_code`: Numeric CBS item code.
- `group`: Item group. One of `"Primary crops"`, `"Crop products"`, `"Livestock products"`, `"Crop residues"`, `"Grassland"`, `"Scavenging"`.
- `live_anim`: Name of the parent live animal for livestock-derived items (`NA` for crop items).
- `live_anim_code`: Numeric CBS code of the parent live animal (`NA` for crop items).
- `Cat_Krausmann`: Item category used in Krausmann et al. biomass flow accounting.
- `Name_biomass`: Corresponding item name in `biomass_coefs`, enabling joins for biomass coefficients.
- `Name_Eurostat`: Corresponding item name in Eurostat agricultural statistics.
- `Name`: Alternative or display name for the item.
- `Cat_Labour`: Labour category used in agricultural labour analyses.
- `Cat_FAO1`: Top-level FAO commodity category (e.g., `"Cereals"`, `"Oilcrops"`).
- `Cat_Origin`: Origin-based commodity category. One of `"Cereals"`, `"Vegetables and Fruits"`, `"Sugar and Stimulants"`, `"Oil Crops"`, `"Fodder crops"`, `"Fibres and Crude Materials"`, or `NA`.
- `Cat_Use`: Use-based commodity category (e.g., `"Grains"`, `"Oils and Fats"`, `"Fodder crops"`, `"Beverages, sugar and stimulants"`).
- `Order`: Numeric ordering field used for sorting items consistently across outputs.
- `Categ`: General category label used in some analyses.
- `Farm_class`: Farm system classification (see `items_prim` for values).
- `c3_c4`: Photosynthetic pathway. One of `"c3"`, `"c4"`, or `NA`.

- `ann_per_nfx`: Annual/perennial and nitrogen fixation trait. One of `"ann"` (annual), `"per"` (perennial), `"nfx"` (nitrogen-fixing), or `NA`.
- `Cat_1`: Primary category label for material flow accounting.
- `Cat_2`: Secondary category label.
- `Cat_3`: Tertiary category label.
- `Cat_4`: Quaternary category label.
- `Herb_Woody`: Plant growth form. One of `"Herbaceous"`, `"Woody"`, or `NA`.
- `Crop_irrig`: Irrigation category used in water-use analyses.
- `Cat_Org`: Organic farming category classification.
- `Cat_Ymax`: Maximum attainable yield category.
- `Cat_Ymax_leg`: Legend label for the `Cat_Ymax` category.

Source

Derived from FAOSTAT Production data and multiple classification schemes from the literature.

Examples

```
head(items_prod_full)
```

`key_lattice_gaps` *Report the expected keys that no row supplies*

Description

Enumerate the keys an expected lattice requires and the data does not supply, returning one row per missing key. A zero-row result means every expected key is present in every group.

This is the detector `check_keys_complete()` signals on, exposed separately because a caller that wants to *act* on the gaps – report them, drop the groups that carry them – needs the keys themselves rather than a condition.

Usage

```
key_lattice_gaps(data, expected, .by = NULL, max_gaps = 1e+06)
```

Arguments

`data`	A tibble to check.
`expected`	The expected key set: a named list of value vectors, a data frame of expected key rows, or a function of `data` returning either.
`.by`	Character vector of grouping columns. Every group must carry the whole expected key set. `NULL` (default) treats the table as one group.
`max_gaps`	Refuse rather than enumerate when more keys than this are missing. A result that large is not a partially incomplete lattice, it is an almost entirely absent one, and enumerating it helps nobody.

Value

A tibble with the `.by` columns and the expected key columns, one row per missing key, sorted by group. Zero rows when the lattice is complete.

Why a total, an identity or a row count cannot do this

An absent month drops its row, so a year sums **eleven** months, and the eleven-month total is a plausible number that no `na.rm` choice can question – there is no `NA` anywhere. A conservation check over the rows that arrived still balances, because both sides are built from the same eleven. A row count still looks reasonable, because it is reasonable: it is the count of the rows that came. Only an assertion against the key set that *should* have arrived distinguishes the two.

What the expected lattice may be

`expected` is one of three forms, and the form decides the cost.

- A **named list** of value vectors, e.g. `list(month = 1:12)`. The expected key set is the cross product of the elements, required in every group.
- A **data frame** of expected key rows, e.g. every `(year, area_code)` pair a polity vocabulary admits. Use this for a lattice that varies along one of its own keys: put the varying key in the frame and do not group on it, and the whole per-year vocabulary is one key set.
- A **function** of `data` returning either of the above, for a rule that has to read the data first – the years it actually spans, the model's cell list.

The contract is containment, not equality: every expected key must occur, and a key the data carries beyond the expected set is not a failure.

Cost, which is the reason the design is shaped this way

A full cross-join assertion is free on twelve months and ruinous on tens of millions of cell-months, so completeness is established in stages and each stage runs only when the cheaper one could not settle it.

1. **Scalar counts.** When the observed vocabulary of each key column lies inside the expected one, the lattice is complete if and only if the number of distinct key tuples equals the number of groups times the size of the expected key set. That is two `dplyr::n_distinct()` passes over column vectors: no frame is copied, no lattice is built and no grouped result is materialised.
2. **A distinct-key anti-join**, reached only when the counts disagree (or when the observed vocabulary is not inside the expected one, which makes the count identity unsound). This works from the *distinct* key tuples, not from every row.
3. **Enumeration of the incomplete groups only.** The lattice is expanded for the groups that already failed their count, never for the whole table. The enumeration is the diagnostic, not the detector.

One rule follows from stage 1 and is worth knowing before wiring a check on a large table: **narrow the data to the keys you are asserting about**. An expected set deliberately smaller than a column's observed vocabulary – "months 1 and 12 must be here", on a frame that also holds 2 to 11 – defeats the count identity and sends every call down stage 2, even when nothing is missing. Filtering to the two months first keeps the assertion identical and the cost scalar; `.wb_check_swc_boundary()` does exactly this.

Measured on the 86,781,420-row `lpjml-soc-hydrology` cell-month lattice (7,231,785 cell-year groups, `lon/lat/year/month`): stage 1 settles it in **13.9 s with no measurable allocation above the frame itself**. For comparison, a grouped distinct count of the same lattice takes 66.7 s and +166 MB, and materialising the full lattice and anti-joining it takes 36.6 s and +2.1 GB. Memory is the binding constraint on this chain (whep#624), which is why the cheapest stage is also the one that allocates nothing; stage 2 costs a further 6.4 s and +2.0 GB there, and is paid only by a lattice that is already broken.

What it cannot see

Three absences pass this check, and a caller must not read a pass as more than it is.

- A **group that is absent entirely**. No rows at all means no group, and `.by` can only name groups the data already carries. Assert the expected *group* set the same way: put the grouping keys in the data-frame form of `expected` and pass no `.by`.
- A key that is present but **wrong**. This is a completeness check, not a validity one; month 13 is not a failure here, only an absent month 12 is.
- A lattice that is complete while its **values** are absent as zeros. That is the other half of the family: assert the input was supplied, not that its keys are all there (whep#1034).

Examples

```
eleven <- tibble::tibble(year = 2000L, month = 1:11)
key_lattice_gaps(eleven, list(month = 1:12), .by = "year")

# Complete: zero rows.
key_lattice_gaps(
  tibble::tibble(year = 2000L, month = 1:12),
  list(month = 1:12),
  .by = "year"
)
```

`lassaletta_grassland_share` *Grassland share of synthetic nitrogen by country and year*

Description

Country-level time series of the share of synthetic nitrogen applied to grassland (versus cropland). Used to split national N totals between land use types in the WHEP nitrogen pipeline.

The series is the per-country grassland proportion that Lassaletta et al. (2014) subtracted from FAO national fertiliser use to obtain nitrogen input to cropland: "FAO data on annual per country synthetic fertilizer use refer to total use in agriculture and is provided without distinction between arable and grassland. We therefore had to subtract from these figures the proportion used for grassland fertilization, which in some European countries such as Ireland and the Netherlands accounts for a significant proportion." The proportions in that paper rest in turn on Richard (1951), Power and Alessi (1971), Anonymous (1992), FAO (2006) and Heffer (2013), detailed in its supplementary material S1.

Usage

```
lassaletta_grassland_share
```

Format

A tibble with one row per country-year combination containing:

- `Country`: Country name, in the FAO-era vocabulary of the source (`"Belgium-Luxemburg"`, `"Czechoslovakia"`, `"Ethiopia PDR"`, `"Yugoslav SFR"`, `"FSU"`).
- `year`: Year (numeric), 1961-2009 for every label.
- `grass_share`: Share of synthetic N applied to grassland, as a fraction in 0-1. The tracked CSV holds the same numbers as percentages.

141 labels x 49 years = 6,909 rows. 4,798 of them are exactly 0 and only 53 labels are ever non-zero; the largest shares are Ireland (0.88) and the Netherlands (0.71), the two countries the source singles out. Many series are linear ramps from 0 in 1961 up to the first documented value, so a value is not an observation for its own year.

The label set is **not** a partition: it carries a historical entity and its modern successors side by side for the whole span, so `Sudan`, `Sudan (former)` and `South Sudan` all appear for 1961-2009, with `Sudan` and `Sudan (former)` carrying identical values. A consumer that resolves these labels to area codes therefore needs a duplicate-key rule.

Source

Lassaletta, L., Billen, G., Grizzetti, B., Anglade, J., & Garnier, J. (2014). 50 year trends in nitrogen use efficiency of world cropping systems: the relationship between yield and nitrogen input to cropland. *Environmental Research Letters*, 9(10), 105011. doi:10.1088/17489326/9/10/105011. The per-country values are that paper's supplementary material S1 and its Annex 1 CSV.

The attribution is evidential, not byte-verified. The tracked CSV arrived from an upstream project that labelled it with this DOI; its span (1961-2009), its country vocabulary and its two extreme countries all agree with the paper, and at 19,531 bytes the wide original is within 2% of the 19.2 KB IOP reports for Annex 1. IOP blocks automated download of that file, so the two were never compared row by row, and the duplicated Sudan labels may be the paper's or a later addition.

Examples

```
head(lassaletta_grassland_share)
```

liv_lu_coefs *Livestock unit coefficients*

Description

Provides livestock unit (LU) conversion factors per head for each animal class, used to express heterogeneous livestock populations in comparable units.

Usage

```
liv_lu_coefs
```

Format

A tibble where each row corresponds to one animal class. It contains the following columns:

- `Animal_class`: Animal class identifier (e.g., `"Dairy_cows"`, `"Cattle"`, `"Sheep_goats"`, `"Broilers"`, `"Hens"`, `"Pigs"`).
- `LU_head`: Livestock units per head (numeric). Dairy cows have a value of 1.0 by convention; smaller animals have proportionally lower values.

Source

Based on standard livestock unit definitions from FAO and European agricultural statistics.

Examples

```
head(liv_lu_coefs)
```

livestock_constants *Livestock physical constants.*

Description

Named list of physical constants used in livestock emission calculations:

- `energy_content_ch4_mj_kg`: 55.65 MJ/kg CH4.
- `ch4_density_kg_m3`: 0.67 kg/m3.
- `vs_energy_content_mj_kg`: 18.45 MJ/kg DM.
- `n_to_n2o`: 44/28 (N to N2O molecular mass ratio).
- `days_in_year`: 365.
- `default_de_percent`: 65%.
- `default_ue_fraction`: 0.04 (urinary energy as fraction of GE).
- `ev_wool_mj_kg`: 24.0 MJ/kg clean wool.

Usage

```
livestock_constants
```

Format

A named list.

Source

IPCC 2019 Refinement, Vol 4, Ch 10.

Examples

```
str(livestock_constants)
```

---

livestock_emissions_to_kt

Convert IPCC per-animal livestock emissions from kilograms to kilotonnes.

Description

Bridge the two unit families the package carries. The IPCC calculators (`calculate_livestock_emissions()` and friends) emit kilograms of gas in `enteric_ch4_tier1`/`enteric_ch4_tier2`, `manure_ch4_tier1`/`manure_ch4_tier2` and `manure_n2o_total`. The FAOSTAT-shaped gridded and national artifacts carry kilotonnes in `enteric_ch4_kt`, `manure_ch4_kt` and `manure_n2o_kt`. Nothing converted between them, so the two families never met; this is the one place that conversion happens.

The factor is exact: 1 kilotonne is 1e6 kilograms.

Usage

```
livestock_emissions_to_kt(data, tier = 2)
```

Arguments

`data`	A tibble from the IPCC livestock calculators, carrying the kilogram columns for the requested `tier`. A missing column aborts rather than being treated as zero.
`tier`	IPCC tier the kilogram columns come from, `2` (default) or `1`.

Value

`data` with `enteric_ch4_kt`, `manure_ch4_kt` and `manure_n2o_kt` added.

Examples

```
tibble::tibble(
  enteric_ch4_tier1 = 8e7,
  manure_ch4_tier1 = 1e7,
  manure_n2o_total = 5e5
) |>
  livestock_emissions_to_kt(tier = 1)
```

---

livestock_production_defaults

Default production parameters.

Description

Default values for fat%, protein%, lactose%, weight gain, work hours, and pregnancy fraction by species.

`estimate_energy_demand()` applies the weight gain to growing cohorts only and the pregnancy fraction to mature-female cohorts only: following IPCC 2019 Vol 4 Ch 10 (pp. 10.18, 10.28), mature animals have no net weight gain over a year and only mature females gestate. Beef breeding cows take the regional IPCC 2019 Table 10A.2 rate instead of the `"Other Cattle"` value of 0. The `cohort_life_stage` and `method_pregnancy` columns of its output record which rule applied.

Usage

```
livestock_production_defaults
```

Format

A tibble with columns:

category Species or animal class.

fat_percent Milk fat content (percent).

protein_percent Milk protein content (percent).

lactose_percent Milk lactose content (percent).

weight_gain_kg_day Average daily weight gain of a growing animal (kg/day).

work_hours_day Hours of draft work per day.

pregnant_fraction Fraction of mature females giving birth in a year.

Source

NRC 2001; IPCC 2019, Vol 4, Ch 10.

Examples

```
livestock_production_defaults
```

`make_lpjml_covariate` *Build an LPJmL/WHEP spatial covariate function*

Description

Creates a `function(centroids_sf, year)` suitable for APIs that accept a spatial covariate density, such as `build_constant_territory_series()`. The covariate is built from prepared WHEP/LPJmL spatialization inputs, either from a local `input_dir` or from pinned WHEP spatialization inputs when `input_dir = NULL`.

Usage

```
make_lpjml_covariate(
  input_dir = NULL,
  years = NULL,
  weighting = c("total_cropland", "crop_pattern"),
  item_prod_code = NULL,
  cft_mapping = whep::cft_mapping
)
```

Arguments

`input_dir`	Directory holding prepared WHEP/LPJmL input parquets. If `NULL`, the function reads pinned spatialization inputs via `whep_read_file()`.
`years`	Integer vector of years to retain. If `NULL`, all years present in the required input are used.
`weighting`	Either `"total_cropland"`, `"crop_pattern"`, or a custom `function(centroids_sf, year)`. A custom function is returned unchanged.
`item_prod_code`	FAOSTAT/WHEP production item code. Required when `weighting = "crop_pattern"`.
`cft_mapping`	Mapping from WHEP item codes to CFT/LUH2 types. Defaults to cft_mapping.

Value

A function `function(centroids_sf, year)` returning non-negative density values aligned to `centroids_sf`.

Examples

```
# A custom covariate function is returned unchanged, ready to plug into
# build_constant_territory_series(). Here a uniform (area-weighting) density:
uniform <- function(centroids_sf, year) rep(1, nrow(centroids_sf))
covariate <- make_lpjml_covariate(weighting = uniform)
identical(covariate, uniform)
```

```
# The "total_cropland" and "crop_pattern" modes instead read prepared
# WHEP/LPJmL spatialization parquets from `input_dir` (or pinned inputs).
```

`manner_default_technique_mix` *Gross-default technique/incorporation-delay blend for MANNER.*

Description

A deliberate, permanent gross-assumption default for `calculate_manner_nh3()`'s `technique` and `incorporation_delay_h` organic-manure drivers, for use where real per-cell/per-era manure-application-technique survey data does not exist (which is everywhere right now). Every row fixes `technique = "Broadcast"`, matching the production MANNER run of the earlier regional historical reconstruction this model is ported from (which itself applies Broadcast nationally with no region/era variation). The four rows blend manner_incorporation_factor's `delay_bin` categories in equal shares: a quarter of applied nitrogen assumed never incorporated, a quarter incorporated within 2 hours, a quarter within 12-24 hours, and a quarter within 1-2 days. See `calculate_manner_nh3_default()`'s Details for the full reasoning; this is not derived from a region/era survey.

Usage

```
manner_default_technique_mix
```

Format

A tibble with columns:

technique Manure application technique: always `"Broadcast"`.

delay_bin Incorporation-delay bin label, one of manner_incorporation_factor's `delay_bin` values: `"No incorporation"`, `"<2 h"`, `"12-24 h"` or `"1-2 days"`.

incorporation_delay_h Hours between surface application and soil incorporation passed to `calculate_manner_nh3()`: `NA` for `"No incorporation"`, 2 for `"<2 h"`, 24 for `"12-24 h"` and 48 for `"1-2 days"`.

share Fraction of applied nitrogen assumed to follow this incorporation-delay bin; the four shares sum to 1 (0.25 each).

Source

Gross default assumption, not a region/era-specific survey; see `calculate_manner_nh3_default()`'s Details for the reasoning.

Examples

```
manner_default_technique_mix
```

`manner_incorporation_factor`
MANNER organic-manure incorporation-delay factor.

Description

Multiplicative ammonia-volatilisation factor for organic manure as a function of the delay between surface application and soil incorporation, by manure type. Used by `calculate_manner_nh3()`'s organic-manure path, distinct from manner_params's `incorporation` table (a land-use factor reused by the synthetic-fertiliser path; see that dataset's description). The delay bins are monotonically increasing; a supplied delay is assigned to the first (shortest) bin whose `delay_hours` is greater than or equal to it, and a missing or infinite delay maps to `"No incorporation"` (factor 1, no volatilisation reduction).

Usage

```
manner_incorporation_factor
```

Format

A tibble with columns:

manure_type Organic manure type: `"cattle_slurry"`, `"pig_slurry"`, `"FYM"` or `"poultry_manure"`.

delay_bin Incorporation-delay bin label (e.g. `"<2 h"`, `"6-12 days"`, `"No incorporation"`).

delay_hours Upper bound of the delay bin in hours; `NA` for `"No incorporation"`.

factor Numeric multiplicative incorporation factor.

Source

WHEP project-internal coefficient workbook (not a public DOI): the MANNER implementation of an earlier regional historical reconstruction, manure factors.

Examples

```
manner_incorporation_factor
```

`manner_params` *MANNER process-based ammonia-volatilisation factors.*

Description

Long-form lookup of the discrete factor tables of the MANNER ammonia volatilisation model. Each row gives one factor, identified by its table (`category`), the lookup key (`key`) and, where the table is two-dimensional, a second key (`sub_key`). The tables are: the per-fertiliser maximum ammonia-loss ceiling (`max_nh3`); the soil-pH factor for synthetic fertiliser (`ph`, sub-keyed by pH band); the incorporation / land-use factor (`incorporation`); the manure application-technique factor (`technique`); the wind-speed factor (`windspeed`); and the combined rainfall-by-wetness factor (`rainfall_wet`). Continuous fertiliser-rate, temperature and rainfall response surfaces are computed in the MANNER function and are not part of this table.

Usage

```
manner_params
```

Format

A tibble with columns:

category Factor table: one of `"max_nh3"`, `"ph"`, `"incorporation"`, `"technique"`, `"windspeed"`, `"rainfall_wet"`.

key Primary lookup key (fertiliser, technique, wind-speed class, or rainfall-wetness class).

sub_key Secondary lookup key (pH band for the `ph` table); `NA` for one-dimensional tables.

factor Numeric multiplicative factor.

Source

Nicholson, F. A., Bhogal, A., Chadwick, D., Gill, E., Gooday, R. D., Lord, E., Misselbrook, T., Rollett, A. J., Sagoo, E., Smith, K. A., Thorman, R. E., Williams, J. R. & Chambers, B. J. (2013). An enhanced software tool to support better use of manure nutrients: MANNER-NPK. *Soil Use and Management*, 29(4), 473-484. doi:10.1111/sum.12078. Underlying mass-flow ammonia framework: Webb, J. & Misselbrook, T. H. (2004). A mass-flow model of ammonia emissions from UK livestock production. *Atmospheric Environment*, 38(14), 2163-2176. doi:10.1016/j.atmosenv.2004.01.023. Values transcribed from the MANNER implementation of an earlier regional historical reconstruction.

Examples

```
manner_params
```

manner_rain_factor *MANNER synthetic-fertiliser rainfall factor.*

Description

Multiplicative ammonia-volatilisation factor for synthetic fertiliser as a function of soil pH class and the same rainfall/wetness classification used by the organic-manure `rainfall_wet` table in manner_params. Used by `calculate_manner_nh3()`'s synthetic-fertiliser path. AN and CAN are 1 for every pH class and rainfall class; AS is 1 for every `pH<7` row.

Usage

```
manner_rain_factor
```

Format

A tibble with columns:

fertiliser Synthetic fertiliser: `"Urea"`, `"AN"`, `"CAN"` or `"AS"`.

ph_class Soil pH class: `"pH<7"` or `"other pH"`.

rainfall_class Combined rainfall-level/wetness class (e.g. `"noraindry"`, `"heavyrainwet"`), the same 9 classes as manner_params's `rainfall_wet` table.

factor Numeric multiplicative rainfall factor.

Source

WHEP project-internal coefficient workbook (not a public DOI): the MANNER implementation of an earlier regional historical reconstruction, synthetic-fertiliser factors.

Examples

```
manner_rain_factor
```

manner_rate_factor *MANNER synthetic-fertiliser application-rate factor.*

Description

Multiplicative ammonia-volatilisation factor for synthetic fertiliser as a function of the nitrogen application rate, keyed by fertiliser, soil type (calcareous / non-calcareous) and application-rate bin (kg N/ha). Used by `calculate_manner_nh3()`'s synthetic-fertiliser path. AN and CAN are 1 for every soil type and rate bin; Urea follows the same rate-response curve on both soil types; AS follows the Urea curve on calcareous soils and is 1 on non-calcareous soils.

Usage

```
manner_rate_factor
```

Format

A tibble with columns:

fertiliser Synthetic fertiliser: `"Urea"`, `"AN"`, `"CAN"` or `"AS"`.

soil_type `"calcareous"` or `"non-calcareous"`.

rate_bin Application-rate bin (kg N/ha): one of `"0-30"`, `"30-60"`, `"60-90"`, `"90-120"`, `"120-150"`, `"150-180"`, `"180-200"`, `">200"`.

factor Numeric multiplicative rate factor.

Source

WHEP project-internal coefficient workbook (not a public DOI): the MANNER implementation of an earlier regional historical reconstruction, synthetic-fertiliser factors.

Examples

```
manner_rate_factor
```

---

manure_inorganic_n *Inorganic (mineral) nitrogen fraction of excreted manure by species.*

Description

Fraction of total excreted nitrogen that is in inorganic (ammoniacal) form and therefore available for ammonia volatilisation, by livestock species and manure stream (whole excreta, or after separation into a liquid and a solid fraction). Used by `calculate_manner_nh3()`'s organic-manure path to scale the realised emission factor down to the ammoniacal nitrogen actually applied. The mapping from `calculate_manner_nh3()`'s `manure_type` argument to this table's `species`/`manure_stream` keys (`cattle_slurry`/`pig_slurry` to the `"Liquid"` stream, `FYM` to Cattle `"Solid"`, `poultry_manure` to Poultry `"Solid"`) is a documented modelling choice made when porting this table, not a literal crosswalk from the source implementation; see `calculate_manner_nh3()`'s Details.

Usage

```
manure_inorganic_n
```

Format

A tibble with columns:

species Livestock species: `"Sheep"`, `"Goats"`, `"Poultry"`, `"Cattle"`, `"Pigs"`, `"Horses"`, `"Donkeys_mules"` or `"Rabbits"`.

manure_stream Manure stream: `"Excreta"` (whole, unseparated excreta), `"Liquid"` or `"Solid"` (after mechanical separation); not every species has a `"Liquid"` row.

inorganic_n_fraction Fraction of the stream's total nitrogen that is inorganic (ammoniacal).

source Short author-year provenance string as cited in the source livestock coefficient workbook for that coefficient. These are secondary citations transcribed from that workbook, not independently DOI-verified full bibliographic entries.

Source

WHEP project-internal coefficient workbook (not a public DOI): the livestock coefficient workbook of an earlier regional historical reconstruction, manure inorganic-nitrogen sheet. That sheet in turn cites: Van Soest, P. J. (1994); Nahm, K. H. (2003); Nahm, K. H. (2005); Smith, K. A. & Frost, J. P. (2000); Chambers, B. J. et al. (1999); Chambers, B. J. et al. (2000); Nicholson, F. A. et al. (1996); Canh, T. T. et al. (1997); Sommer, S. G. et al. (2004); Burton, C. H. & Turner, C. (2003); Martinez, J. & Burton, C. H. (2003); Rotz, C. A. (2004); Wheeler, E. F. et al. (2011); Gungor-Demirci, G. & Demirer, G. N. (2004); Lebas, F. (1975); Lebas, F. (2004). These secondary citations are transcribed as recorded in the workbook and have not been independently verified against the primary sources.

Examples

```
manure_inorganic_n
```

---

max_intake_share *Maximum intake shares.*

Description

Per livestock category diet share caps that limit how much of a diet a feed item or feed class may make up. Migrated from the afsetools `Livestock_coefs.xlsx` workbook.

Usage

```
max_intake_share
```

Format

A tibble with one row per cap:

livestock_category Livestock category the cap applies to.

var Cap key type: item_cbs (per item) or Cat_feed (per feed class).

var_value Item or feed class the cap applies to.

max_intake_share Maximum share of the diet (fraction).

Source

afsetools `Livestock_coefs.xlsx`.

`meisinger_denitrification`
Meisinger and Randall topsoil denitrification share matrix.

Description

Fraction of nitrogen surplus lost through topsoil denitrification, as a function of fertiliser category, soil organic matter content, drainage class and climate. The full matrix is keyed by fertiliser category (synthetic versus manure), tillage regime, soil organic matter class, climate category and drainage class. Where drainage is `"None"` (waterlogged), the entire surplus is denitrified (share 1.00).

Usage

```
meisinger_denitrification
```

Format

A tibble with columns:

fert_cat Fertiliser category: `"Synthetic"` or `"Manure"`.

tillage Tillage regime: `"Tillage"`, `"No_tillage"` or `"Not_specified"` (manure).

som_content Soil organic matter class: `"Low"`, `"Medium"` or `"High"`.

climate_cat Climate category label used in the source matrix: `"Semiarid"` or `"Humid"`.

drainage_rate Drainage class: `"Very_high"`, `"High"`, `"Medium"`, `"Low"`, `"Very_low"` or `"None"`.

denit_share Fraction of nitrogen surplus denitrified in the topsoil.

climate Climate zone the row applies to: `"MED"` or `"ATL"`.

Source

Meisinger, J. J. & Randall, G. W. (1991). Estimating nitrogen budgets for soil-crop systems. In R. F. Follett, D. R. Keeney & R. M. Cruse (Eds.), *Managing Nitrogen for Groundwater Quality and Farm Profitability* (pp. 85-124). Soil Science Society of America. doi:10.2136/1991.managingnitrogen.c5. Values transcribed from the nitrogen coefficient workbook of an earlier regional historical reconstruction.

Examples

```
meisinger_denitrification
```

melt_bilateral_trade *Melt a bilateral trade matrix to long format.*

Description

Convert the `bilateral_trade` list-column of `get_bilateral_trade()` (one square origin-by-destination matrix per year and item) into the tidy `from_code`/`to_code` long form consumed by `compute_footprint_balance()`. Self-trade (diagonal) entries are dropped.

Usage

```
melt_bilateral_trade(bilateral_trade)
```

Arguments

`bilateral_trade`	Tibble from `get_bilateral_trade()`, with `year`, `item_cbs_code` and a `bilateral_trade` matrix list-column.

Value

A tibble with `year`, `from_code`, `to_code`, `item_cbs_code` and `value`.

Examples

```
m <- matrix(
  c(0, 40, 0, 0),
  nrow = 2,
  dimnames = list(c("1", "2"), c("1", "2"))
)
bt <- tibble::tibble(
  year = 2010L, item_cbs_code = 10L, bilateral_trade = list(m)
)
melt_bilateral_trade(bt)
```

mueller_synthetic_n *Synthetic nitrogen application rates by crop and country*

Description

Country- and crop-process-level synthetic nitrogen application rates (kg N ha^{-1}), derived from Mueller et al. (2012). Used as reference crop-specific N rates in the WHEP nitrogen pipeline.

Usage

```
mueller_synthetic_n
```

Format

A tibble with one row per crop-process-country combination containing:

- `proc_code`: Internal process code (e.g. `"p001"`).
- `crop_process`: Descriptive crop process name (e.g. `"Rice production"`).
- `crop_original`: Crop name as in the source dataset.
- `unit`: Unit of the rate value (always `"kgN/ha"`).
- `iso3c`: ISO 3166-1 alpha-3 country code.
- `rate_value`: Nitrogen application rate (kg N ha^{-1}).

Source

Mueller, N. D. et al. (2012). Closing yield gaps through nutrient and water management. *Nature*, 490(7419), 254–257. doi:10.1038/nature11420

Examples

```
head(mueller_synthetic_n)
```

---

n_attenuation_constants

Nitrogen leaching-attenuation and indirect-N2O constants.

Description

Scalar constants for the nitrogen-loss cascade: the input C:N leaching attenuation parameters (`a_cn_*`) and the IPCC indirect-emission factors. The C:N attenuation reduces nitrate leaching for carbon-rich, low-availability inputs; the indirect factors convert volatilised ammonia and leached nitrate into indirect N2O and set the ammonia-volatilisation fractions.

Usage

```
n_attenuation_constants
```

Format

A tibble with columns:

constant Constant name: `"a_cn_min_cn"`, `"a_cn_span"`, `"a_cn_span_other"`, `"a_cn_max"`, `"ef5_no3_to_n2o"`, `"ef4_nh3_to_n2o_atl"`, `"nh3_frac_synthetic"`, `"nh3_frac_organic"`.

value Numeric value of the constant.

description Human-readable description of the constant.

Source

C:N attenuation parameters from the nitrogen pipeline of an earlier regional historical reconstruction. Indirect emission factors and ammonia volatilisation fractions: IPCC (2019), 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol. 4, Chapter 11 (EF5 = 0.011 for nitrate leaching to N2O; EF4 = 0.014 for Atlantic, i.e. wet-climate, ammonia to N2O, p. 11.26; NH3 volatilisation fractions 0.11 synthetic and 0.21 organic, Table 11.3).

Examples

```
n_attenuation_constants
```

---

n_boundary_params *Planetary reactive-nitrogen boundary parameters.*

Description

Parameters defining the planetary boundary for anthropogenic reactive nitrogen and its agri-food-system apportionment, used by the SJOS-N per-capita boundary axis. The low, high and top values bracket the published planetary reactive-nitrogen limit; the per-capita cap, the synthetic-to-total agricultural ratio and the food share of agricultural nitrogen scale that global limit to a comparable per-capita agricultural basis.

Usage

```
n_boundary_params
```

Format

A tibble in long form with columns:

parameter Parameter name (e.g. `"boundary_low"`, `"per_capita_cap"`, `"syn_tot_agri_ratio"`).

value Numeric parameter value.

unit Unit of the value (e.g. `"Tg N/yr"`, `"kg N/cap/yr"`, `"ratio"`, `"fraction"`).

description Human-readable description of the parameter.

Source

Planetary reactive-nitrogen boundary literature: de Vries, W., Kros, J., Kroeze, C. & Seitzinger, S. P. (2013). Assessing planetary and regional nitrogen boundaries related to food security and adverse environmental impacts. *Current Opinion in Environmental Sustainability*, 5(3-4), 392-402. doi:10.1016/j.cosust.2013.07.004; Campbell, B. M., Beare, D. J., Bennett, E. M., Hall-Spencer, J. M., Ingram, J. S. I., Jaramillo, F., Ortiz, R., Ramankutty, N., Sayer, J. A. & Shindell, D. (2017). Agriculture production as a major driver of the Earth system exceeding planetary boundaries. *Ecology and Society*, 22(4):8. doi:10.5751/ES09595220408; Springmann, M. et al. (2018). Options for keeping the food

system within environmental limits. *Nature*, 562, 519-525. doi:10.1038/s415860180594-0; regional agricultural-nitrogen boundaries: Schulte-Uebbing, L. F., Beusen, A. H. W., Bouwman, A. F. & de Vries, W. (2022). From planetary to regional boundaries for agricultural nitrogen pollution. *Nature*, 610, 507-512. doi:10.1038/s41586022051582. The specific low/high/top boundary values, the per-capita cap and the agricultural-apportionment ratios are transcribed from the Global SJOS-N analysis synthesising these sources; verify against Zotero before any manuscript use.

Examples

```
n_boundary_params
```

---

n2o_efs_disaggregated

Disaggregated direct soil N2O emission factors by climate and irrigation.

Description

Direct nitrous-oxide emission factors (fraction of applied nitrogen emitted as N2O-N) disaggregated by climate zone and irrigation type. Atlantic strata use the IPCC Tier 1 default of 0.01 (row `irrig_type == "Tier_1"`); Mediterranean strata use the Cayuela et al. (2017) meta-analytic factors that resolve the strong effect of water management on N2O. Atlantic non-flooded irrigation rows carry a missing factor because Atlantic strata are routed to the `"Tier_1"` factor; the Atlantic flooded row keeps its own value.

Usage

```
n2o_efs_disaggregated
```

Format

A tibble with columns:

irrig_type Irrigation / management stratum: one of `"Tier_1"`, `"Rainfed"`, `"Traditional"`, `"Drip"`, `"Sprinkler"`, `"Flooded"`, `"Med_average"`.

climate Climate zone: `"MED"` (Mediterranean) or `"ATL"` (Atlantic).

ef Direct N2O emission factor (kg N2O-N per kg applied N); `NA` for Atlantic non-flooded irrigation strata.

Source

Cayuela, M. L., Aguilera, E., Sanz-Cobena, A., Adams, D. C., Abalos, D., Barton, L., Ryals, R., Silver, W. L., Alfaro, M. A., Pappa, V. A., Smith, P., Garnier, J., Billen, G., Bouwman, L., Bondeau, A. & Lassaletta, L. (2017). Direct nitrous oxide emissions in Mediterranean climate cropping systems: emission factors based on a meta-analysis of available measurement data. *Agriculture, Ecosystems & Environment*, 238, 25-35. doi:10.1016/j.agee.2016.10.006. Precursor Mediterranean N2O review: Aguilera, E., Lassaletta, L.,

Sanz-Cobena, A., Garnier, J. & Vallejo, A. (2013). The potential of organic fertilizers and water management to reduce N2O emissions in Mediterranean climate cropping systems. A review. *Agriculture, Ecosystems & Environment*, 164, 32-52. doi:10.1016/j.agee.2012.09.006. Atlantic Tier 1 default: IPCC (2019), 2019 Refinement to the 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol. 4, Chapter 11.

Examples

```
n2o_efs_disaggregated
```

`n2o_efs_ipcc2006` *IPCC 2006 Tier 1 direct soil N2O emission factors by climate and irrigation.*

Description

Direct nitrous-oxide emission factors (fraction of applied nitrogen emitted as N2O-N) under the IPCC 2006 Guidelines Tier 1 defaults, disaggregated by irrigation/management stratum and climate zone. Unlike `n2o_efs_disaggregated()` (the Cayuela et al. 2017 meta-analytic Mediterranean factors plus the IPCC 2019 Atlantic default), every stratum here carries the flat IPCC 2006 default (0.010), except flooded rice (0.003) in both climate zones. Used by `calculate_soil_n2o()`'s `method = "ipcc2006"`.

Usage

```
n2o_efs_ipcc2006
```

Format

A tibble with columns:

irrig_type Irrigation / management stratum: one of `"Tier_1"`, `"Rainfed"`, `"Traditional"`, `"Drip"`, `"Sprinkler"`, `"Flooded"`, `"Med_average"`.

climate Climate zone: `"MED"` (Mediterranean) or `"ATL"` (Atlantic).

ef Direct N2O emission factor (kg N2O-N per kg applied N).

Source

WHEP project-internal coefficient workbook (not independently DOI-verified): `N_coefficients.xlsx`, sheet `N2O_EFs_IPCC2006`, itself transcribed from IPCC (2006), 2006 IPCC Guidelines for National Greenhouse Gas Inventories, Vol. 4, Chapter 11, Tier 1 defaults.

Examples

```
n2o_efs_ipcc2006
```

`normalize_nourishment` *Normalize and classify per-capita nourishment.*

Description

Adds a piecewise-normalized nourishment score `value_norm` and its Under / Adequate / Over class `nourish`. The normalization (the Global `Intake_normalization`) is `value / floor` below the floor, `1 + value / ceiling` above the ceiling and `1 + (value - floor) / (ceiling - floor)` in between, so the score is below 1 under the floor, exactly 1 at the floor, between 1 and 2 across the adequate band and 2 or above at or past the ceiling. The class is `"Under"` when `value_norm < 1`, `"Adequate"` when `value_norm < 2` and `"Over"` otherwise. Protein is the SJOS-N nourishment axis, so the defaults are the protein floor and ceiling (62.1 and 85.05 g/cap/day) from nourishment_thresholds.

Of those two defaults only the underlying 46 g/cap/day floor is sourced (WHO/FAO/UNU TRS 935 Table 46, the safe intake of a 55 kg adult, itself a 97.5th-percentile individual level rather than a population one). The 63 ceiling and the 1.35 factor that lifts both to a supply basis carry no source; `nourishment_thresholds$provenance` says so per row. `build_nourishment_band()` is the sourced replacement for both bounds and is not wired in here yet.

Passing `value_col = energy_kcal_cap_day` runs the same arithmetic on a different quantity and is **not** a second WHEP axis: the packaged energy bounds are unsourced, and WHEP's energy column is gross combustion energy where a dietary threshold is metabolisable. Supply your own bounds and your own metabolisable series if you want that comparison.

Usage

```
normalize_nourishment(x, value_col = protein_g_cap_day, thresholds = NULL)
```

Arguments

`x`	A tibble carrying the per-capita nourishment column named by `value_col` (for example a `build_food_supply()` output).
`value_col`	The unquoted nourishment column to normalize. Defaults to `protein_g_cap_day`.
`thresholds`	Either a named `floor`/`ceiling` pair applied to every row (a named numeric vector or list), or a **data frame of per-country-year bounds** keyed by `year` and `area_code` with either `floor_g_cap_day`/`ceiling_g_cap_day` or `floor`/`ceiling` — so a `build_nourishment_band()` output passes straight through. A row that matches no band is classified `NA` and named in a warning, never silently given the flat default. When `NULL` (default) the flat protein bounds from nourishment_thresholds are used.

Value

`x` with `value_norm` (numeric score) and `nourish` (`"Under"`, `"Adequate"` or `"Over"`) added.

Examples

```
normalize_nourishment(
  tibble::tribble(
    ~area_code, ~protein_g_cap_day,
    10L, 30,
    20L, 70,
    30L, 100
  )
)
```

`nourish_levels` *Nourishment classification levels and colours.*

Description

The ordered levels of the nourishment ("just") axis (`Over`, `Adequate`, `Under`) with a plotting colour per level, used to classify and colour polities by per-capita nourishment adequacy.

Usage

```
nourish_levels
```

Format

A tibble with columns:

level Nourishment level: `"Over"`, `"Adequate"` or `"Under"`.

order Integer plotting/factor order (1 to 3).

colour Plotting colour (an R colour name).

Source

Level labels and colours ported by value from `afsetools::load_vectors()` (`Nour_levels` and `Nourish_colours`). The nourishment adequacy framing follows Springmann, M. et al. (2018). Options for keeping the food system within environmental limits. *Nature*, 562, 519-525. doi:10.1038/s4158601805940; verify against Zotero before any manuscript use.

Examples

```
nourish_levels
```

`nourishment_thresholds` *Nourishment protein and energy thresholds.*

Description

Protein and dietary-energy floors and ceilings for the SJOS-N nourishment ("just") axis, plus the waste-and-inequality factor that lifts the raw per-capita protein bounds to supply-side thresholds and the normalised-score class cutoffs. Raw protein floor and ceiling (46 and 63 g/cap/day) are scaled by the 1.35 waste-and-inequality factor to the supply-side protein floor and ceiling (62.1 and 85.05 g/cap/day). The class cutoffs (1 and 2) split a normalised nourishment score into the Under, Adequate and Over classes.

Only one of these numbers is sourced. `provenance` records which, per row, so nothing here can look verified when it is not: the protein floor of 46 g/cap/day is WHO/FAO/UNU TRS 935 Table 46, the safe intake of a 55 kg adult at a PDCAAS of 1.0 — and that safe level is a 97.5th-percentile *individual* value, which TRS 935 (p.41) says is incorrect to apply to a population. The 63 ceiling, the 2300 and 2900 energy bounds and the 1.35 factor are `inherited_unsourced`: they come from the Global SJOS-N analysis and no source has been produced for them, the author having confirmed 1.35 was a preliminary presentation figure (whep#753).

The energy bounds are **not** a second axis WHEP publishes. Nothing in the package reads them, and WHEP's own energy column is gross combustion energy where a dietary kcal threshold is metabolisable, so the two are not comparable as they stand. See `normalize_nourishment()`.

Usage

```
nourishment_thresholds
```

Format

A tibble in long form with columns:

metric Metric name: `"protein_raw"`, `"protein"`, `"energy"`, `"waste_inequality"` or `"class"`.

bound Which bound the value is: `"floor"`, `"ceiling"`, `"factor"`, `"under"` or `"over"`. The upper protein and energy bound is named `"ceiling"` because `normalize_nourishment()` uses it as the top of the Adequate band, above which a country is classified Over — not as something to aim at (whep#753).

value Numeric threshold value.

unit Unit of the value (`"g/cap/day"`, `"kcal/cap/day"`, `"ratio"` or `"score"`).

provenance Where the value comes from: `"trs935_table46_55kg_safe_level"`, `"derived_raw_times_waste_i` `"inherited_unsourced"` or `"definition"`.

Source

The protein floor, and only the protein floor: WHO/FAO/UNU (2007). *Protein and amino acid requirements in human nutrition*, WHO Technical Report Series 935, Table 46.

This table previously cited Springmann, M. et al. (2018), doi:10.1038/s4158601805940, for its values. That attribution was **wrong** and has been removed: two independent full-text searches of the paper and its supplementary information found none of 46, 63, 62.1, 85.05, 1.35, 2300 or 2900 in it. Springmann's protein figures appear only as diet composition. Every value other than the floor is transcribed from the Global SJOS-N analysis with no source established, and carries `provenance == "inherited_unsourced"`; do not cite one without finding a source first.

Examples

```
nourishment_thresholds
```

---

plot_compart_factor *Plot each compartment's own factor breakdown*

Description

Plots panels 3-6 of the composite decomposition figure described in the decomposition proposal (section 14): the cumulative, year-on-year contribution of each factor *within* one compartment's own decomposition (e.g. Cropland's Size/Intensity/Inefficiency), as opposed to the AFS-wide panels 1-2 from `plot_loss_decomp()`, which only show each compartment's total contribution.

Usage

```
plot_compart_factor(
  cropland = NULL,
  semi_natural = NULL,
  manure = NULL,
  urban = NULL
)
```

Arguments

`cropland`	A tibble from `decompose_cropland_surplus()`. If `NULL`, computed automatically (slow).
`semi_natural`	A tibble from `decompose_semi_natural_surplus()`. If `NULL`, computed automatically (slow).
`manure`	A tibble from `decompose_manure_losses()`. If `NULL`, computed automatically (slow).
`urban`	A tibble from `decompose_urban_losses()`. If `NULL`, computed automatically (slow).

Value

A named list with ggplot objects `cropland`, `semi_natural`, `manure`, and `urban`.

Examples

```
# Each argument is a calculate_lmdi() table; the same shape is reused for
# all four compartments here.
lmdi <- tibble::tribble(
  ~period, ~factor_label, ~component_type, ~additive,
  "1861-1862", "N surplus", "target", 20000,
  "1861-1862", "Size", "factor", 9000,
  "1861-1862", "Intensity", "factor", 7000,
  "1861-1862", "Inefficiency", "factor", 4000,
  "1862-1863", "N surplus", "target", 10000,
  "1862-1863", "Size", "factor", 5000,
  "1862-1863", "Intensity", "factor", 3000,
  "1862-1863", "Inefficiency", "factor", 2000
)
plots <- plot_compart_factor(lmdi, lmdi, lmdi, lmdi)
```

---

plot_compart_factor_periods

Plot each compartment's own factor breakdown by period, as one combined panel plot

---

Description

Same factor-level breakdown as `plot_compart_factor()`, but using the four reference-period bars from `decompose_terr_losses_periods()` (matching `plot_loss_decomp_periods()`) instead of the year-on-year cumulative view, and combining all four compartments side by side into a single patchwork plot sharing one legend, instead of four separate ggplot objects. Each factor label has a fixed, unique color across the whole plot, so no two factors share a color.

Usage

```
plot_compart_factor_periods(
  cropland = NULL,
  semi_natural = NULL,
  manure = NULL,
  urban = NULL
)
```

Arguments

`cropland`	A tibble from `decompose_cropland_surplus()` with `by_period = TRUE`. If `NULL`, computed automatically (slow).

`semi_natural`	A tibble from `decompose_semi_natural_surplus()` with `by_period = TRUE`. If `NULL`, computed automatically (slow).
`manure`	A tibble from `decompose_manure_losses()` with `by_period = TRUE`. If `NULL`, computed automatically (slow).
`urban`	A tibble from `decompose_urban_losses()` with `by_period = TRUE`. If `NULL`, computed automatically (slow).

Value

A patchwork ggplot object: one panel per compartment, side by side, with a single shared legend.

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)
) {
  # Each argument is a by-period calculate_lmdi() table, as returned by
  # decompose_cropland_surplus(by_period = TRUE).
  lmdi <- tibble::tribble(
    ~period, ~period_years, ~factor_label, ~component_type, ~additive,
    "1865-1925", 60, "Size", "factor", 5400,
    "1865-1925", 60, "Intensity", "factor", 3600,
    "1865-1925", 60, "Inefficiency", "factor", -1200,
    "1925-1965", 40, "Size", "factor", 4000,
    "1925-1965", 40, "Intensity", "factor", 9200,
    "1925-1965", 40, "Inefficiency", "factor", 2800
  )
  panel <- plot_compart_factor_periods(lmdi, lmdi, lmdi, lmdi)
}
```

---

plot_compart_factor_roll

Plot each compartment's own factor breakdown, rolling mean

Description

Same as `plot_compart_factor_yearly()`, but smooths each year's own additive contribution with a centered rolling mean (`window` years wide, `NA`-padded at the edges) before plotting, one panel per compartment.

Usage

```
plot_compart_factor_roll(
  cropland = NULL,
  semi_natural = NULL,
  manure = NULL,
```

```
  urban = NULL,
  window = 10
)
```

Arguments

`cropland`	A tibble from `decompose_cropland_surplus()`. If `NULL`, computed automatically (slow).
`semi_natural`	A tibble from `decompose_semi_natural_surplus()`. If `NULL`, computed automatically (slow).
`manure`	A tibble from `decompose_manure_losses()`. If `NULL`, computed automatically (slow).
`urban`	A tibble from `decompose_urban_losses()`. If `NULL`, computed automatically (slow).
`window`	Width of the centered rolling-mean window, in years. Default `10`.

Value

A named list with ggplot objects `cropland`, `semi_natural`, `manure`, and `urban`.

Examples

```
# `window` must not exceed the number of periods available per factor.
lmdi <- tibble::tribble(
  ~period, ~factor_label, ~component_type, ~additive,
  "1861-1862", "N surplus", "target", 20000,
  "1861-1862", "Size", "factor", 9000,
  "1862-1863", "N surplus", "target", 10000,
  "1862-1863", "Size", "factor", 5000,
  "1863-1864", "N surplus", "target", 12000,
  "1863-1864", "Size", "factor", 6000
)
plots <- plot_compart_factor_roll(lmdi, lmdi, lmdi, lmdi, window = 3)
```

---

plot_compart_factor_roll_panel

Plot each compartment's own factor breakdown, rolling mean, as one combined panel plot

---

Description

Same factor-level breakdown as `plot_compart_factor_roll()`, combining all four compartments side by side into a single patchwork plot, matching `plot_compart_factor_periods()`'s style: one shared y-axis per axis-sharing pair (Cropland+Semi-natural share "N surpluses", Livestock+Urban share "N losses"), one shared legend, fixed unique colors per factor label.

Usage

```
plot_compart_factor_roll_panel(
  cropland = NULL,
  semi_natural = NULL,
  manure = NULL,
  urban = NULL,
  window = 10
)
```

Arguments

`cropland`	A tibble from `decompose_cropland_surplus()`. If `NULL`, computed automatically (slow).
`semi_natural`	A tibble from `decompose_semi_natural_surplus()`. If `NULL`, computed automatically (slow).
`manure`	A tibble from `decompose_manure_losses()`. If `NULL`, computed automatically (slow).
`urban`	A tibble from `decompose_urban_losses()`. If `NULL`, computed automatically (slow).
`window`	Width of the centered rolling-mean window, in years. Default `10`.

Value

A patchwork ggplot object: one panel per compartment, side by side, with a single shared legend.

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)
) {
  lmdi <- tibble::tribble(
    ~period, ~factor_label, ~component_type, ~additive,
    "1861-1862", "N surplus", "target", 20000,
    "1861-1862", "Size", "factor", 9000,
    "1862-1863", "N surplus", "target", 10000,
    "1862-1863", "Size", "factor", 5000,
    "1863-1864", "N surplus", "target", 12000,
    "1863-1864", "Size", "factor", 6000
  )
  panel <- plot_compart_factor_roll_panel(
    lmdi,
    lmdi,
    lmdi,
    lmdi,
    window = 3
  )
}
```

`plot_compart_factor_yearly` *Plot each compartment's own factor breakdown, year-on-year (non-cumulative)*

Description

Same as `plot_compart_factor()`, but plots each year's own additive contribution directly, without accumulating it over time — matching `plot_loss_decomp_yearly()`'s year-on-year view, one panel per compartment.

Usage

```
plot_compart_factor_yearly(
  cropland = NULL,
  semi_natural = NULL,
  manure = NULL,
  urban = NULL
)
```

Arguments

`cropland`	A tibble from `decompose_cropland_surplus()`. If `NULL`, computed automatically (slow).
`semi_natural`	A tibble from `decompose_semi_natural_surplus()`. If `NULL`, computed automatically (slow).
`manure`	A tibble from `decompose_manure_losses()`. If `NULL`, computed automatically (slow).
`urban`	A tibble from `decompose_urban_losses()`. If `NULL`, computed automatically (slow).

Value

A named list with ggplot objects `cropland`, `semi_natural`, `manure`, and `urban`.

Examples

```
lmdi <- tibble::tribble(
  ~period, ~factor_label, ~component_type, ~additive,
  "1861-1862", "N surplus", "target", 20000,
  "1861-1862", "Size", "factor", 9000,
  "1861-1862", "Intensity", "factor", 7000,
  "1862-1863", "N surplus", "target", 10000,
  "1862-1863", "Size", "factor", 5000,
  "1862-1863", "Intensity", "factor", 3000
)
plots <- plot_compart_factor_yearly(lmdi, lmdi, lmdi, lmdi)
```

plot_crop_livestock_conn

Plot the crop-livestock connectivity index

Description

Plots the national, unweighted average of local feed self-sufficiency and the manure-recycling ratio from `decompose_crop_livestock_conn()` as a line chart over time. Falling lines indicate growing crop-livestock disconnection. This is meant as a supplementary diagnostic reported alongside (not inside) the main additive decomposition, per the decomposition proposal.

Usage

```
plot_crop_livestock_conn(connectivity = NULL)
```

Arguments

`connectivity`	A named list from `decompose_crop_livestock_conn()`. If `NULL`, computed automatically (slow).

Value

A ggplot object.

Examples

```
connectivity <- list(
  national = tibble::tribble(
    ~year, ~self_sufficiency, ~recycling_ratio,
    1960, 0.95, 0.62,
    1980, 0.71, 0.48,
    2000, 0.54, 0.39
  )
)
p <- plot_crop_livestock_conn(connectivity)
```

plot_destiny_mix

Plot the cropland destiny mix over time

Description

Plots the national cropland destiny mix (domestic food / feed / exported food / non-food shares) from `decompose_destiny_mix()` as a line chart over time, showing the local-food -> feed + export transition described in the decomposition proposal (sections 3 and 14). This is meant as a supplementary diagnostic reported alongside (not inside) the main additive decomposition, since `decompose_cropland_surplus()` no longer carries a destiny factor.

Usage

```
plot_destiny_mix(destiny_mix = NULL)
```

Arguments

destiny_mix	A tibble from decompose_destiny_mix(). If NULL, computed automatically (slow).

Value

A ggplot object.

Examples

```
destiny_mix <- tibble::tribble(
  ~year, ~destiny_grp, ~share,
  1960, "domestic_food", 0.55,
  1960, "feed", 0.30,
  1960, "exported", 0.10,
  1960, "non_food", 0.05,
  2000, "domestic_food", 0.35,
  2000, "feed", 0.45,
  2000, "exported", 0.15,
  2000, "non_food", 0.05
)
p <- plot_destiny_mix(destiny_mix)
```

---

plot_finn_circularity

Plot Finn Cycling Index evolution and period comparison

Description

Creates three panels: (a) temporal evolution of mean FCI per typology with interquartile ribbon; (b) FCI distribution at four key periods as boxplots; (c) net change in mean FCI from first to last period.

Usage

```
plot_finn_circularity(
  periods = c(1860, 1920, 1960, 2010),
  finn_data = NULL,
  n_prov_destiny = NULL,
  typologies = NULL
)
```

Arguments

`periods`	Integer vector of years to mark as dashed reference lines on the `evolution` panel. The `periods` and `change` panels always compare the four fixed historical eras (1860-1870, 1920-1930, 1960-1970, 2010-2020) regardless of this argument.
`finn_data`	Pre-computed tibble from `create_finn_indicator()`. If `NULL`, computed automatically (slow).
`n_prov_destiny`	Passed to `create_finn_indicator()` when `finn_data` is `NULL`.
`typologies`	Typology assignment per year and province, with columns `year`, `province_name` and `Typology_base`. If `NULL`, derived from `create_typo_ts_plot()`.

Value

A named list with ggplot objects `evolution`, `periods`, `change`.

Examples

```
# `periods` must name years present in the data.
finn_data <- tibble::tribble(
  ~year, ~province_name, ~finn_index,
  1960, "A", 0.12,
  1960, "B", 0.18,
  2000, "A", 0.07,
  2000, "B", 0.09
)
typologies <- tibble::tribble(
  ~year, ~province_name, ~Typology_base,
  1960, "A", "Specialized cropping systems (intensive)",
  1960, "B", "Semi-natural agroecosystems",
  2000, "A", "Specialized cropping systems (intensive)",
  2000, "B", "Semi-natural agroecosystems"
)
plots <- plot_finn_circularity(
  periods = c(1960, 2000),
  finn_data = finn_data,
  typologies = typologies
)
```

`plot_footprint_sankey`

Interactive footprint Sankey viewer

Description

Create a browser-based Sankey viewer from a footprint table, such as the output of `compute_footprint()`. The viewer includes stage filters, a per-stage node limit controls, a minimum-flow threshold, hover tooltips, click-to-highlight interactions, and SVG download.

Each row is treated as one path through the selected `stages`, with `value_col` giving the path size. Rows are summed before plotting, so repeated paths are shown as one flow.

The default stages are `origin_area`, `origin_item`, `target_item`, and `target_area`; `target_fd` is appended automatically when present. If columns such as `origin_area_name` or `target_item_name` are present, they are used as display labels.

Usage

```
plot_footprint_sankey(
  footprints,
  stages = NULL,
  value_col = "value",
  label_cols = NULL,
  max_nodes = 10,
  other_label = "Other",
  stage_max_nodes = NULL,
  stage_other_labels = NULL,
  embed_max_nodes = Inf,
  stage_embed_max_nodes = NULL,
  min_share = 0,
  title = "Footprint Sankey Viewer",
  subtitle = NULL,
  value_label = "footprint",
  width = "100%",
  height = 680,
  file = NULL,
  open = FALSE
)
```

Arguments

`footprints`	A data frame with footprint paths and a numeric value column.
`stages`	Character vector of columns to use as Sankey stages. Defaults to the footprint origin and target columns.
`value_col`	Name of the numeric column containing flow values.
`label_cols`	Optional named character vector mapping stage columns to display-label columns, for example `c(origin_area = "origin_area_name")`. When omitted, `{stage}_name` columns are used where available.
`max_nodes`	Maximum number of individual nodes to show per stage in the current filtered browser view. Less important nodes are grouped into `other_label`. Use `Inf` to show all nodes.
`other_label`	Label used for grouped nodes when `max_nodes` is finite.

`stage_max_nodes`
: Optional named numeric vector overriding `max_nodes` for selected stages, for example `c(product = 75, target_area = 12)`.

`stage_other_labels`
: Optional named character vector overriding `other_label` for selected stages, for example `c(product = "Other products")`.

`embed_max_nodes`
: Maximum number of individual nodes to embed per stage before writing the viewer. Less important nodes are permanently grouped before serialization, reducing file size. Use `Inf` to keep all nodes available to the browser.

`stage_embed_max_nodes`
: Optional named numeric vector overriding `embed_max_nodes` for selected stages.

`min_share`
: Minimum visible path size as a percentage of the current filtered total. Set to 0 to keep diffuse trade links visible. Users can change this in the browser.

`title`
: Optional viewer title.

`subtitle`
: Optional viewer subtitle.

`value_label`
: Label used in tooltips and the summary line.

`width`
: Viewer width as a CSS value.

`height`
: Sankey SVG height in pixels.

`file`
: Optional path to write a standalone HTML viewer.

`open`
: If `TRUE` and `file` is supplied, open the written viewer in the default browser. If `TRUE` and `file` is `NULL`, a temporary HTML file is written and opened.

Value

A browsable HTML object when `file` is `NULL` and `open` is `FALSE`; otherwise invisibly returns the HTML file path.

Examples

```
footprints <- tibble::tribble(
  ~origin_area, ~origin_item, ~target_item, ~target_area, ~value,
  "Brazil", "Soybeans", "Pigmeat", "China", 10,
  "Brazil", "Soybeans", "Milk", "China", 4,
  "Brazil", "Soybeans", "Soybean Oil", "France", 3
)

if (
  interactive() &&
    requireNamespace("htmltools", quietly = TRUE) &&
    requireNamespace("jsonlite", quietly = TRUE)
) {
  plot_footprint_sankey(footprints)
}
```

plot_input_output *Plot national nitrogen inputs, production, and surplus for a land system.*

Description

Builds a stacked-area plot of Spanish national nitrogen inputs (as negative values), production, residues, and surplus over time for either cropland or semi-natural agroecosystems. For the semi-natural system a nitrogen "Accumulation" term (net soil/biomass N accumulation) is added when the `n_balance_ygpit_all` pin is available.

Usage

```
plot_input_output(
  system = c("Cropland", "semi_natural_agroecosystems"),
  per_ha = FALSE,
  example = FALSE
)
```

Arguments

`system`	Character. One of `"Cropland"` or `"semi_natural_agroecosystems"`.
`per_ha`	Logical. If `TRUE`, express nitrogen flows per hectare of the system's land area (kg N/ha) instead of national totals (Gg N). Requires remote data, so it is ignored in example mode. Default is `FALSE`.
`example`	If `TRUE`, build the plot from a small example dataset without downloading remote data. Default is `FALSE`.

Value

A `ggplot` object.

Examples

```
plot_input_output(example = TRUE)
```

plot_input_output_livestock *Plot national nitrogen inputs, production, and surplus for livestock.*

Description

Builds a stacked-area plot of Spanish national livestock nitrogen feed inputs (as negative values), production, and surplus over time. On real data feed is broken down by origin (local grass, local crops, imports) and production is split into ruminant and monogastric output using the `stock_prod_ygps` pin. In example mode a simpler feed-by-destiny breakdown is used so the plot builds offline.

Usage

```
plot_input_output_livestock(per_ha = FALSE, example = FALSE)
```

Arguments

`per_ha`	Logical. If `TRUE`, express nitrogen flows per hectare of agricultural land (kg N/ha) instead of national totals (Gg N). Requires remote data, so it is ignored in example mode. Default is `FALSE`.
`example`	If `TRUE`, build the plot from a small example dataset without downloading remote data. Default is `FALSE`.

Value

A `ggplot` object.

Examples

```
plot_input_output_livestock(example = TRUE)
```

`plot_input_output_system` *Plot national nitrogen inputs and uses for the full agro-food system.*

Description

Builds a stacked-area plot of Spanish national nitrogen inputs (soil inputs and imports, as negative values) against uses (feed, food, other uses, exports) and surplus over time. A nitrogen "Accumulation" term is added when the `n_balance_ygpit_all` pin is available.

Usage

```
plot_input_output_system(per_ha = FALSE, example = FALSE)
```

Arguments

`per_ha`	Logical. If `TRUE`, express nitrogen flows per hectare of agricultural land (kg N/ha) instead of national totals (Gg N). Requires remote data, so it is ignored in example mode. Default is `FALSE`.
`example`	If `TRUE`, build the plot from a small example dataset without downloading remote data. Default is `FALSE`.

Value

A `ggplot` object.

Examples

```
plot_input_output_system(example = TRUE)
```

`plot_loss_decomp` *Plot cumulative drivers of the change in territorial N losses*

Description

Plots two stacked-area charts of the cumulative, year-on-year contribution to the change in Spain's total territorial nitrogen (N) losses since the start of the reconstruction: one broken down by compartment (cropland, semi-natural, manure, urban), one regrouped by transformation mechanism (scale, specialization, intensification, efficiency), as computed by `decompose_terr_losses()`.

Usage

```
plot_loss_decomp(decomp = NULL)
```

Arguments

`decomp`	A named list from `decompose_terr_losses()`. If `NULL`, computed automatically (slow).

Value

A named list with ggplot objects `by_compartment` and `by_mechanism`.

Examples

```
decomp <- list(
  by_compartment = tibble::tribble(
    ~t0, ~compartment, ~contribution_mgn, ~cumulative_mgn,
    1861, "cropland", 12000, 12000,
    1861, "manure", 4000, 4000,
    1862, "cropland", 8000, 20000,
    1862, "manure", -1000, 3000
  ),
  by_mechanism = tibble::tribble(
    ~t0, ~mechanism, ~contribution_mgn, ~cumulative_mgn,
    1861, "Size", 9000, 9000,
    1861, "Inefficiency", 7000, 7000,
    1862, "Size", 5000, 14000,
    1862, "Inefficiency", 2000, 9000
  )
)
plots <- plot_loss_decomp(decomp)
```

`plot_loss_decomp_periods` *Plot period-based drivers of the change in territorial N losses*

Description

Plots two stacked bar charts from `decompose_terr_losses_periods()`: one bar per reference period comparison, labeled by mean year (1865-1925, 1925-1965, 1965-2015), each compared against the immediately preceding reference period (chained), plus one extra bar for the full analysis window (Total (1865-2015)), broken down by compartment in one chart and by mechanism in the other. Contributions are normalized to Gg N/yr (see `decompose_terr_losses_periods()`), since the chained transitions and the Total span very different numbers of years.

Usage

```
plot_loss_decomp_periods(decomp = NULL)
```

Arguments

`decomp`	A named list from `decompose_terr_losses_periods()`. If `NULL`, computed automatically (slow).

Value

A named list with ggplot objects `by_compartment` and `by_mechanism`.

Examples

```
decomp <- list(
  by_compartment = tibble::tribble(
    ~period, ~compartment, ~contribution_per_yr_mgn,
    "1865-1925", "cropland", 120,
    "1865-1925", "manure", 40,
    "1925-1965", "cropland", 260,
    "1925-1965", "manure", 90
  ),
  by_mechanism = tibble::tribble(
    ~period, ~mechanism, ~contribution_per_yr_mgn,
    "1865-1925", "Size", 90,
    "1865-1925", "Inefficiency", 70,
    "1925-1965", "Size", 150,
    "1925-1965", "Inefficiency", 200
  )
)
plots <- plot_loss_decomp_periods(decomp)
```

`plot_loss_decomp_periods_panel` *Plot period-based drivers of territorial N losses, as one combined panel plot*

Description

Combines the two views from `plot_loss_decomp_periods()` (by compartment and by mechanism) side by side into a single patchwork plot with one shared y-axis label, since both break down contributions to the same total territorial N losses. Each panel keeps its own legend (compartment and mechanism are different fill scales, so the legends aren't collected into one).

Usage

```
plot_loss_decomp_periods_panel(decomp = NULL)
```

Arguments

`decomp`	A named list from `decompose_terr_losses_periods()`. If `NULL`, computed automatically (slow).

Value

A patchwork ggplot object with two panels ("By compartment", "By mechanism").

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)
) {
  decomp <- list(
    by_compartment = tibble::tribble(
      ~period, ~compartment, ~contribution_per_yr_mgn,
      "1865-1925", "cropland", 120,
      "1925-1965", "cropland", 260,
      "1865-1925", "urban", 15,
      "1925-1965", "urban", 35
    ),
    by_mechanism = tibble::tribble(
      ~period, ~mechanism, ~contribution_per_yr_mgn,
      "1865-1925", "Size", 90,
      "1925-1965", "Size", 150,
      "1865-1925", "Intensification", 45,
      "1925-1965", "Intensification", 145
    )
  )
  panel <- plot_loss_decomp_periods_panel(decomp)
}
```

`plot_loss_decomp_rolling` *Plot rolling-mean year-on-year drivers of the change in territorial N losses*

Description

Same data as `plot_loss_decomp_yearly()`, but smooths each year's own additive contribution with a centered rolling mean (`window` years wide, `NA`-padded at the edges) before plotting, to make sustained multi-year trends (e.g. a period of continuously improving efficiency) easier to see than in the raw, noisy year-on-year series.

Usage

```
plot_loss_decomp_rolling(decomp = NULL, window = 10)
```

Arguments

`decomp`	A named list from `decompose_terr_losses()`. If `NULL`, computed automatically (slow).
`window`	Width of the centered rolling-mean window, in years. Default `10`.

Value

A named list with ggplot objects `by_compartment` and `by_mechanism`.

Examples

```
# `window` must not exceed the number of years available per group.
decomp <- list(
  by_compartment = tibble::tribble(
    ~t0, ~compartment, ~contribution_mgn, ~cumulative_mgn,
    1861, "cropland", 12000, 12000,
    1862, "cropland", 8000, 20000,
    1863, "cropland", 9000, 29000
  ),
  by_mechanism = tibble::tribble(
    ~t0, ~mechanism, ~contribution_mgn, ~cumulative_mgn,
    1861, "Size", 9000, 9000,
    1862, "Size", 5000, 14000,
    1863, "Size", 6000, 20000
  )
)
plots <- plot_loss_decomp_rolling(decomp, window = 3)
```

`plot_loss_decomp_rolling_panel` *Plot rolling-mean year-on-year drivers of territorial N losses, as one combined panel plot*

Description

Combines the two views from `plot_loss_decomp_rolling()` (by compartment and by mechanism) side by side into a single patchwork plot, matching `plot_loss_decomp_periods_panel()`'s style: one shared y-axis label, each panel keeping its own legend (compartment and mechanism are different fill scales, so the legends aren't collected into one).

Usage

```
plot_loss_decomp_rolling_panel(decomp = NULL, window = 10)
```

Arguments

`decomp`	A named list from `decompose_terr_losses()`. If `NULL`, computed automatically (slow).
`window`	Width of the centered rolling-mean window, in years. Default `10`.

Value

A patchwork ggplot object with two panels ("By compartment", "By mechanism").

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)
) {
  decomp <- list(
    by_compartment = tibble::tribble(
      ~t0, ~compartment, ~contribution_mgn, ~cumulative_mgn,
      1861, "cropland", 12000, 12000,
      1862, "cropland", 8000, 20000,
      1863, "cropland", 9000, 29000
    ),
    by_mechanism = tibble::tribble(
      ~t0, ~mechanism, ~contribution_mgn, ~cumulative_mgn,
      1861, "Size", 9000, 9000,
      1862, "Size", 5000, 14000,
      1863, "Size", 6000, 20000
    )
  )
  panel <- plot_loss_decomp_rolling_panel(decomp, window = 3)
}
```

plot_loss_decomp_yearly *Plot year-on-year (non-cumulative) drivers of the change in territorial N losses*

Description

Uses the same data as `plot_loss_decomp()`, but plots each year's own additive contribution directly, without accumulating it over time: each bar shows how much a compartment or mechanism contributed to the change in territorial N losses in that one year-on-year transition, not the running total since 1860.

Usage

```
plot_loss_decomp_yearly(decomp = NULL)
```

Arguments

`decomp`	A named list from `decompose_terr_losses()`. If `NULL`, computed automatically (slow).

Value

A named list with ggplot objects `by_compartment` and `by_mechanism`.

Examples

```
decomp <- list(
  by_compartment = tibble::tribble(
    ~t0, ~compartment, ~contribution_mgn, ~cumulative_mgn,
    1861, "cropland", 12000, 12000,
    1862, "cropland", 8000, 20000,
    1861, "urban", 500, 500,
    1862, "urban", 700, 1200
  ),
  by_mechanism = tibble::tribble(
    ~t0, ~mechanism, ~contribution_mgn, ~cumulative_mgn,
    1861, "Size", 9000, 9000,
    1862, "Size", 5000, 14000,
    1861, "Intensification", 3500, 3500,
    1862, "Intensification", 3700, 7200
  )
)
plots <- plot_loss_decomp_yearly(decomp)
```

plot_national_trade_flows_raw

Plot national trade flows: model vs. raw historical FAO series

Description

Plots time series of gross exports and imports, aggregated to Crop vs. Livestock, comparing the provincial GRAFS model's bottom-up national trade against the raw historical Export/Import series for Spain.

Usage

```
plot_national_trade_flows_raw(trade_flows = NULL)
```

Arguments

`trade_flows`	Optional pre-computed output from `compute_trade_flows_raw()`. If `NULL`, calls that function internally.

Value

A ggplot object.

Examples

```
trade_flows <- tibble::tribble(
  ~year, ~item, ~source, ~category, ~flow, ~value_n,
  1930, "Nuts and products", "FAO (raw)", "Crop", "Export", 825,
  1930, "Nuts and products", "WHEP model", "Crop", "Export", 910,
  1930, "Bovine Meat", "FAO (raw)", "Livestock", "Import", 340,
  1930, "Bovine Meat", "WHEP model", "Livestock", "Import", 295
)
p <- plot_national_trade_flows_raw(trade_flows)
```

plot_national_trade_validation

Plot national net trade validation

Description

Plots a time series comparing Spain's national net trade computed bottom-up from the provincial model against official FAOSTAT figures. Values are aggregated over all items per year.

Usage

```
plot_national_trade_validation(validation = NULL)
```

Arguments

`validation`	Optional pre-computed output from `validate_national_trade()`. If `NULL`, calls that function internally.

Value

A ggplot object.

Examples

```
validation <- tibble::tribble(
  ~year, ~item, ~net_prov, ~net_fao, ~diff_net,
  1960, "Barley and products", 12000, 9500, 2500,
  1960, "Bovine Meat", -3000, -2800, -200,
  2000, "Barley and products", 56943, 2175, 54768,
  2000, "Bovine Meat", 1601, 1569, 32
)
p <- plot_national_trade_validation(validation)
```

---

plot_specialization_cov

Plot the specialization-vs-diversification allocation covariance

---

Description

Plots the Olley-Pakes allocation covariance from `decompose_specialization_cov()` as a line chart, one line per dimension (cropland province, cropland destiny, livestock species), with a zero reference line. Positive and rising values indicate genuine specialization (allocation concentrating into high-surplus units); values near zero or falling indicate diversification. This is meant as a supplementary diagnostic reported alongside (not inside) the main additive decomposition, per the decomposition proposal.

Usage

```
plot_specialization_cov(covariance = NULL)
```

Arguments

`covariance`	A named list from `decompose_specialization_cov()`. If `NULL`, computed automatically (slow).

Value

A ggplot object.

Examples

```
covariance <- list(
  cropland_province = tibble::tribble(
    ~year, ~covariance,
    1960, -0.02,
    1980, 0.05,
    2000, 0.11
  ),
  livestock_species = tibble::tribble(
    ~year, ~covariance,
    1960, 0.01,
    1980, 0.08,
    2000, 0.17
  )
)
p <- plot_specialization_cov(covariance)
```

---

plot_typology_indicators_panel

Plot four N indicators as time series per typology

Description

Creates a four-panel figure showing the temporal evolution of external N dependency, Finn Cycling Index, pollution (soil + livestock surplus per ha), and intensification (synthetic + feed imports per ha) as mean lines per typology with interquartile ribbons.

Usage

```
plot_typology_indicators_panel(
  finn_data = NULL,
  n_prov_destiny = NULL,
  area_df = NULL,
  typo_df = NULL
)
```

Arguments

`finn_data`	Pre-computed tibble from `create_finn_indicator()`. If `NULL`, computed automatically (slow).
`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`area_df`	Cropland area per year and province, with columns `year`, `province_name` and `area_ha`. If `NULL`, read from the `npp_ygpit` pin.
`typo_df`	Typology assignment per year and province, with columns `year`, `province_name` and `Typology_base`. If `NULL`, derived from `create_typo_ts_plot()`.

Value

A patchwork ggplot object.

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)
) {
  # Two provinces at two dates is enough to exercise the four panels; the
  # real figure spans 50 provinces and 1860-2023.
  flows <- tibble::tribble(
    ~year, ~province_name, ~box, ~origin, ~destiny, ~mg_n,
    1960, "A", "Cropland", "Synthetic", "Cropland", 900,
    1960, "A", "Cropland", "Outside", "livestock_mono", 300,
    1960, "A", "Cropland", "Cropland", "population_food", 500,
    1960, "B", "Cropland", "Synthetic", "Cropland", 400,
    1960, "B", "Cropland", "Outside", "livestock_mono", 100,
    2000, "A", "Cropland", "Synthetic", "Cropland", 2600,
    2000, "A", "Cropland", "Outside", "livestock_mono", 1800,
    2000, "A", "Cropland", "Cropland", "population_food", 700,
    2000, "B", "Cropland", "Synthetic", "Cropland", 900,
    2000, "B", "Cropland", "Outside", "livestock_mono", 500
  )
  area_df <- tibble::tribble(
    ~year, ~province_name, ~area_ha,
    1960, "A", 10000,
    1960, "B", 8000,
    2000, "A", 9000,
    2000, "B", 7000
  )
  typo_df <- tibble::tribble(
    ~year, ~province_name, ~Typology_base,
    1960, "A", "Specialized cropping systems",
    1960, "B", "Semi-natural agroecosystems",
    2000, "A", "Specialized cropping systems",
    2000, "B", "Semi-natural agroecosystems"
  )
  finn_data <- tibble::tribble(
    ~year, ~province_name, ~finn_index,
    1960, "A", 0.12,
    1960, "B", 0.18,
    2000, "A", 0.07,
    2000, "B", 0.09
  )
  panel <- plot_typology_indicators_panel(
    finn_data = finn_data,
    n_prov_destiny = flows,
    area_df = area_df,
    typo_df = typo_df
  )
}
```

`plot_typology_periods_panel` *Plot four N indicators as period comparisons per typology*

Description

Creates a four-panel figure comparing external N dependency, Finn Cycling Index, pollution (soil + livestock surplus per ha), and intensification (synthetic + feed imports per ha) across four reference periods (1860-1870, 1920-1930, 1960-1970, 2010-2020), analogous to the periods panel from `plot_finn_circularity()`. Each panel facets by typology and adds a "Spain (national)" facet computed from the national GRAFS dataset (`create_n_nat_destiny()`), shown as a single black point per period since there is only one national observation per period.

Usage

```
plot_typology_periods_panel(
  finn_data = NULL,
  n_prov_destiny = NULL,
  n_nat_destiny = NULL,
  panel_data = NULL
)
```

Arguments

`finn_data`	Pre-computed tibble from `create_finn_indicator()`. If `NULL`, computed automatically (slow).
`n_prov_destiny`	Nitrogen flows tibble from `create_n_prov_destiny()`. If `NULL`, loaded automatically.
`n_nat_destiny`	National nitrogen flows tibble from `create_n_nat_destiny()`. If `NULL`, computed automatically (slow).
`panel_data`	Named list overriding the two frames otherwise read from pins: `area_df` (`year`, `province_name`, `area_ha`) and `typo_df` (`year`, `province_name`, `Typology_base`). Missing elements are loaded automatically.

Value

A patchwork ggplot object.

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE)
) {
  # The four reference periods are 1860-1870, 1920-1930, 1960-1970 and
```

```
  # 2010-2020, so an example needs at least one year inside two of them.
  flows <- tibble::tribble(
    ~year, ~province_name, ~box, ~origin, ~destiny, ~mg_n,
    1865, "A", "Cropland", "Synthetic", "Cropland", 900,
    1865, "A", "Cropland", "Outside", "livestock_mono", 300,
    1865, "B", "Cropland", "Synthetic", "Cropland", 400,
    1965, "A", "Cropland", "Synthetic", "Cropland", 2600,
    1965, "A", "Cropland", "Outside", "livestock_mono", 1800,
    1965, "B", "Cropland", "Synthetic", "Cropland", 900
  )
  panel_data <- list(
    area_df = tibble::tribble(
      ~year, ~province_name, ~area_ha,
      1865, "A", 10000,
      1865, "B", 8000,
      1965, "A", 9000,
      1965, "B", 7000
    ),
    typo_df = tibble::tribble(
      ~year, ~province_name, ~Typology_base,
      1865, "A", "Specialized cropping systems",
      1865, "B", "Semi-natural agroecosystems",
      1965, "A", "Specialized cropping systems",
      1965, "B", "Semi-natural agroecosystems"
    )
  )
  finn_data <- tibble::tribble(
    ~year, ~province_name, ~finn_index,
    1865, "A", 0.12,
    1865, "B", 0.18,
    1965, "A", 0.07,
    1965, "B", 0.09
  )
  panel <- plot_typology_periods_panel(
    finn_data = finn_data,
    n_prov_destiny = flows,
    n_nat_destiny = flows,
    panel_data = panel_data
  )
}
```

`polities` *Polities*

Description

Periodized WHEP polity database imported from the `whep-polities` repository.

Usage

```
polities
```

Format

An sf data frame where each row corresponds to one territorial polity over a continuous time interval. Key columns include:

- `polity_code`: Stable WHEP polity identifier, usually `PREFIX-start_year-end_year`.
- `polity_name`: Human-readable polity name.
- `start_year`, `end_year`: Validity interval for the row. `start_year` is inclusive; `end_year` is **exclusive at a succession** and **inclusive at the open end**. Exclusive at a succession, so `F51-1947-1993` Czechoslovakia covers 1947-1992 and 1993 belongs to `CZE-1993-2025` and `SVK-1993-2025`, and 2014 belongs to `"RUS-2014-2025"` rather than to `"RUS-1991-2014"` (filtering `year <= end_year` returns both epochs and double-counts every boundary year). Inclusive at the open end, so 2025 still belongs to `"RUS-2014-2025"`, because no later interval of that polity follows it and a uniformly exclusive read would leave the current year with no polity at all. **Openness is absence of a successor, not a comparison against the last year in the table.** 258 live polities have no successor, 246 of them end in 2025, and no live polity ending in 2025 carries one, so the two readings agree on this vintage; the successor test is the one that keeps agreeing when the horizon moves. The distinction is not decoration: 256 of the 781 rows end in 2025 but only 253 are open, because three are succeeded there (`BLZ-1800-2025`, `CAN-1948-2025` and `IRQ-1921-2025`, all `retired` or `superseded`), and opening those too would count the terminal year twice.
- `iso3_code`, `iso3c`: ISO3 code where one exists. `iso3c` is retained as a compatibility alias.
- `wiki_status`: Upstream review state. `"retired"` and `"superseded"` mark a DEAD row, kept so a code already held in older output stays resolvable, but never a resolution target: polity_area_crosswalk excludes them, so resolve through it rather than through this table.
- `polygon_status`: Polygon status in `whep-polities` (`"assigned"`, `"proxy"`, `"missing"`, or `"excluded"`).
- `polygon_feature_year`, `polygon_feature_date`: The vintage of the source feature the polygon was taken from. `polygon_feature_date` arrived with the #835 upstream re-sync and is populated on 3 rows, where upstream recorded a day rather than only a year.
- `has_geometry`: Logical flag indicating whether the geometry is non-empty.
- `geom`: Multipolygon geometry.

Source

`~/whep-polities/data/final/polities_database.gpkg`.

`polities_cats` *Polity categories and regional classifications*

Description

Reference table for countries and political entities (polities) with identifiers from multiple data sources and assignments to various regional groupings used in the literature and international databases.

Usage

```
polities_cats
```

Format

A tibble where each row corresponds to one polity (country or territory). It contains the following columns:

- `legacy_polity_prefix`: Legacy current polity prefix, usually ISO 3166-1 alpha-3 (e.g., `"AFG"`, `"ALB"`). **Not a polities code** and not a join key to any polity table: it was called `polity_code` until whep#687, where the name promised an identity none of its values holds. Read `reporting_polity_code` for the polity.
- `polity_name`: Current polity, country, or territory name.
- `V1`: Internal row index from the source table.
- `code`: Numeric FAOSTAT country code.
- `polity_area_code`: Numeric WHEP reporting area code used in matrix workflows.
- `reporting_polity_code`: Current periodized WHEP polity code for `code`.
- `reporting_polity_name`: Current WHEP polity name for `code`.
- `reporting_polity_has_geometry`: Logical flag indicating whether the current reporting polity has a polygon.
- `iso3c`: ISO 3166-1 alpha-3 code (character; may duplicate `legacy_polity_prefix` or differ for aggregates).
- `FAOSTAT_name`: Country name as used in FAOSTAT.
- `EU27`: Logical flag; `TRUE` if the polity is a member of the EU27.
- `name`: Country name used in other external databases.
- `eia`: Country name or code used by the US Energy Information Administration (EIA).
- `iea`: Country identifier used by the International Energy Agency (IEA).
- `water_code`: Numeric code used in water statistics datasets.
- `water_area`: Country/area name used in water statistics.
- `baci`: Numeric BACI trade database country code.
- `fish`: Numeric code used in fisheries datasets.
- `region_code`: Numeric regional grouping code.

- `cbs`: Logical flag; `TRUE` if the polity is included in the CBS dataset.
- `fabio_code`: Numeric country code used in the FABIO database.
- `ADB_Region`: Asian Development Bank regional classification.
- `region`: General world region (e.g., `"South Asia"`, `"Eastern Europe"`).
- `uISO3c`: Numeric Unicode / UN M49 country code.
- `Lassaletta`: Country grouping used in Lassaletta et al. nitrogen flow studies.
- `region_krausmann`: Regional grouping from Krausmann et al. biomass flow accounting.
- `region_HANPP`: Regional grouping used in human appropriation of net primary production (HANPP) studies.
- `region_krausmann2`: Alternative Krausmann regional grouping.
- `region_UN_sub`: UN sub-regional classification (M49 sub-region).
- `region_UN`: UN macro-regional classification (M49 region).
- `region_ILO1`: ILO primary regional grouping.
- `region_ILO2`: ILO secondary regional grouping.
- `region_ILO3`: ILO tertiary regional grouping.
- `region_IEA`: IEA regional grouping.
- `region_IPCC`: IPCC regional grouping used in climate assessments.
- `region_labour`: Labour-focused regional grouping.
- `region_labour_agg`: Aggregated labour-focused regional grouping.
- `region_labour_mech`: Labour mechanisation regional grouping, `"mech"` or `"no_mech"`, inherited from regions_full (see there for the Angola cell repaired in whep#855).

Which regional groupings WHEP reads

The grouping columns are not all inputs to this package. Six have a consumer in the tree, measured over `R/`, `data-raw/`, `tests/`, `vignettes/` and `inst/`: `region` (carried into polity_area_crosswalk at build time), `region_krausmann` (the residue recovery rate, and the IPCC excreta regions of `prepare_spatialize_all.R`), `region_HANPP` (the modern-variety adoption share), `region_UN_sub` (the residue feed-use fraction, since whep#405), `ADB_Region` (build_primary_production area keys) and `EU27` (the EU aggregate of the FABIO comparison). `region_code` carries no information `region` does not – the two are a 1:1 relabelling – so it needs no consumer of its own.

The rest – `Lassaletta`, `region_krausmann2`, `region_UN`, `region_ILO1`, `region_ILO2`, `region_ILO3`, `region_IEA`, `region_IPCC`, `region_labour`, `region_labour_agg`, `region_labour_mech` – are published third-party taxonomies shipped for downstream analysis and read by nothing in the package. They are shipped as reference, and carry no promise of being re-validated against their upstream taxonomy on release, so a consumer should check the gap it inherits before keying anything by one (whep#386).

The gap the present-day taxonomies share is dissolved states. Over the 202 `cbs` reporters, `region_ILO1`, `region_ILO2`, `region_ILO3`, `region_IEA` and `region_IPCC` are each `NA` for exactly the four federations WHEP still books commodity balances for – Czechoslovakia (51), Serbia and Montenegro (186), the USSR (228) and the Yugoslav SFR (248) – and

complete everywhere else; `region_UN` labels three of the four and leaves only Czechoslovakia `NA`. `ROW` (999) carries an explicit `"RoW"` value in all of them rather than `NA`. Grouping by one of these without deciding what to do with the federations silently drops the pre-succession record. `region_UN_sub`, which shares the gap and does have a consumer, is pinned against it in `test_region_classifications.R`.

A `region_test` column with two values (`"Europe"`, `"Other"`) and no consumer was dropped in whep#386.

Note

Derived from regions_full rather than vendored separately: the 198-code membership is read from `harmonization/polities_cats.csv` and every column value comes from `regions_full`, so the two tables cannot disagree. The vendored table used to file Bhutan under `RASI` and Comoros under `RAFR` with `cbs` `FALSE` and `fabio_code` `999`, on the grounds that neither had a commodity balance sheet when it was compiled; that override was dropped in whep#395, because the `faostat-cbs-new` pin now carries 175 rows for Bhutan and 237 for Comoros, and because it left both areas' `polity_area_code` and `reporting_polity_code` naming the country individually while three label columns said rest-of-region. Areas WHEP does fold, such as Andorra (`REUR`), carry `polity_area_code` 999 too.

Source

Compiled from FAOSTAT, UN M49, ILO, IEA, and other international statistical sources.

Examples

```
head(polities_cats)
```

`polity_anchor_drift` *Find back-cast rows whose polity label and growth proxy describe different territories*

Description

A pre-1962 WHEP row is a reconstruction with **two territorial references** and they need not agree (whep#748):

- its **level** is the area's reported value at the back-cast anchor, walked backwards by `fill_proxy_growth()`, so it describes the territory that area had in `backcast_anchor`. That is the territory `reporting_polity_code` names by default, because `add_polity_code()` floors the polity lookup at the same anchor;
- its **year-on-year movement** is a ratio of LUH2 land, and under `build_primary_production(land_method = "present_day")` the `luh2-areas` pin is keyed on **present-day ISO3**, so the movement describes the territory that area has today.

Wherever a territory changed after the anchor those are different polygons. Nothing in a built table says so: `polity_coverage_gaps()` answers the neighbouring question "was this polity live in the row's year", which is true of far more rows and says nothing about whether the label's extent is the extent the value was computed on. This reports the disagreement itself, for a table that has already been built, so it can be sized without re-deriving the crosswalk.

No row is an error and none is dropped from any build. `drift_kind` says what kind of disagreement a pair carries:

- `"entity"`: the anchor polity and the reference polity are different entities, e.g. area 181 is labelled `SRH-1953-1964` (Southern Rhodesia) while its LUH2 movement is Zimbabwe's. The label can then overlap a sibling area's own published rows.
- `"interval"`: the same entity in two vintages, e.g. area 238 is labelled `ETH-1952-1993`, which includes Eritrea, while its movement is `ETH-1993-2025`, which does not. The entity name matches and the polygon does not, which is the harder class to notice.
- `"unmapped_reference"`: the area resolves to no polity at `reference_year` at all, so what the movement describes cannot be named.

The comparison is always against the anchor, whatever `options(whep.backcast_polity = )` is set to. That switch decides which of the two territories a published label names, and its `"present_day"` value relabels exactly the pairs reported here whose reference polity is live in `reference_year`, so this is also the list of rows it would move.

A pair whose anchor polity is itself `NA` is **not** reported: that row has no label to disagree with, and it is `polity_mapping_provenance()` and `polity_coverage_gaps()` that account for it.

`data_year_polity_code` is reported alongside, resolved unfloored, because it is the third reference in play: `build_primary_production(land_method = "historical_polity")` measures the `ha` half inside that polity's polygon (see `build_historical_land_areas()`), so under that method the movement describes `data_year_polity_code` rather than `reference_polity_code`.

Usage

```
polity_anchor_drift(
  table,
  code_column = "area_code",
  year_column = "year",
  backcast_anchor = 1961L,
  reference_year = 2023L
)
```

Arguments

`table`	A data frame carrying an area-code column and a year column.
`code_column`	Name of the column holding numeric area codes. The column may hold either a FAOSTAT `area_code` or the `polity_area_code` bucket that published outputs are keyed by; both resolve through the same lookup.
`year_column`	Name of the column holding years. Required: the whole question is year-dependent.

`backcast_anchor`
: First year of reported (non-back-cast) FAOSTAT data; passed to the same resolution `add_polity_code()` documents, and the year the level is anchored at.

`reference_year`
: The year whose territory the growth proxy describes. Defaults to `2023L`, `build_primary_production()`'s default `end_year`, i.e. the last year the published artifact covers, which is the vintage the present-day ISO3 keying of the `luh2-areas` pin resolves to. Measured on the shipped crosswalk the answer is identical for every reference year from 2014 to 2024, so the default is not load-bearing.

Value

A tibble with one row per drifting `(area_code, year)`, ordered by area code and year, carrying `area_code`, `year`, `anchor_polity_code`, `anchor_polity_name`, `data_year_polity_code`, `reference_polity_code`, `reference_polity_name`, `drift_kind` and `n_rows`, the number of rows of `table` that pair carries. Zero rows means every pre-anchor row's label describes the same territory its growth proxy does.

Only the back-cast era is reported

A row at or after `backcast_anchor` carries a value its own year's territory reported, resolved at its own year, so its label and its level agree by construction and it is not a disagreement this function has anything to say about. Such rows are filtered out, which is why a `table` holding no pre-anchor year returns zero rows. A missing year column aborts rather than returning zero rows for the same reason: an empty answer must mean "no drift", never "nothing was supplied to look at".

See Also

`polity_coverage_gaps()` for rows attributed to a polity not live in their year, `polity_mapping_provenance()` for which authority a row's identity rests on, and `polity_bucket_coverage()` for buckets that sum more than one territory.

Examples

```
# Area 238 Ethiopia is the documented case: the 1850 row is labelled
# `ETH-1952-1993`, which includes Eritrea, while its LUH2 growth proxy is
# keyed on present-day `ETH`, which does not.
polity_anchor_drift(
  tibble::tibble(
    area_code = c(238L, 238L, 11L),
    year = c(1850L, 2000L, 1850L),
    value = 1
  )
)
```

`polity_area_crosswalk`

FAOSTAT/FABIO area-to-polity crosswalk

Description

Year-aware bridge from numeric reporting `area_code` values used by FAOSTAT/FABIO-derived WHEP data to periodized WHEP `polity_code` values.

Usage

```
polity_area_crosswalk
```

Format

A tibble with one row per area-code/polity-period mapping. Key columns:

- `area_code`: Numeric FAOSTAT/FABIO reporting area code.
- `area_name`: Reporting area name.
- `area_iso3c`: Reporting-area ISO3-like code where available.
- `legacy_polity_prefix`, `legacy_polity_name`: The ISO3-like stem (`"ARM"`, `"ROCE"`, `"REUR"`) and legacy label this package vendors from `regions_full.csv`, kept for older callers and used at build time only as a *candidate prefix* for polity inference. **Neither is an identity**: not one stem is a polities `polity_code`, so a join to polities on it comes back empty. Read `polity_code` for the polity this row resolves to. Until #711 the pair shipped as `reporting_polity_code`/`reporting_polity_name`, the package's own names for a real periodized polity (see whep_polity_columns), which is exactly the trap #687 removed from regions_full.
- `polity_area_code`: Numeric area code retained for WHEP matrix workflows.
- `polity_code`, `polity_name`: Matched WHEP polity, or `NA` for statistical composites that are not real polities.
- `polity_start_year`, `polity_end_year`: Validity interval for the matched polity, on the same convention as polities: `polity_end_year` is exclusive at a succession, so the period covers `polity_start_year:(polity_end_year - 1)`, and inclusive at the open end, so an interval nothing succeeds covers its own terminal year. `add_polity_code()` resolves a year on that reading, widened further to the inclusive `map_year_end` below where the upstream map declares a reported year past the territorial span.
- `mapping_source`: How the area-to-polity decision was reached. `"upstream_map"` for the published `whep-polities` FAOSTAT area map, which is the authority for the years FAOSTAT reports; `"prefix_outside_map"` for a period of a mapped area lying outside every span the map declares, kept so sources reported under their own historical borders still resolve; `"fabio_row_fold"` where FABIO collapses the area into its Rest-of-World bucket; `"prefix_fallback"` where the map covers the area not at all and the mapping is inferred from the polity-code prefix; `"whep_bucket_aggregate"` for a row that answers for an aggregation **bucket** rather than for a reporting area, described below.

- `map_year_start`, `map_year_end`: Inclusive reporting years the upstream map assigns to this area-polity pair, `NA` unless `mapping_source` is `"upstream_map"`.
- `map_match_route`: Upstream's record of how it decided the row (`"iso-equal"`, `"registry"`, `"manual-route"`, `"manual-replace"`, `"manual-span"`), `NA` unless `mapping_source` is `"upstream_map"`.
- `applies_from_year`: First year the row answers for, when that is later than `polity_start_year`. `NA` on every row that answers for a reporting area, whose lower bound is the polity's own start – a row that exists must resolve, so area 276 answers `SDN-2011-2025` from 2011 even though upstream begins reporting the area in 2012. It is non-`NA` only on a `"whep_bucket_aggregate"` row, where it is the first year the bucket actually sums more than one reporting area.
- `mapping_status`: Whether a polity was found, **not** how much to trust it. `"matched"` when a live polity resolved; `"manual"` when the decision was curated by hand, either by upstream (a `manual-* map_match_route`) or by this package's own area-prefix overrides; `"unmapped"` when no real polity is available, which is FAOSTAT area 351 "China" alone, deliberately left unmapped so it cannot double-count its own components; and `"not_a_reporting_area"` for a territory that carries no FAOSTAT/FABIO area at all, which has `NA` in both `area_code` and `polity_area_code` and so can never be joined to reported data.
- `mapping_note`: Explanation for manual, unmapped and non-reporting rows.

Keying on `(area_code, year)` is keying on the polity

This is a **contract**, asserted by the test suite over the full reporting era, not an accident of the current snapshot: every `(area_code, year)` that resolves at all resolves to exactly **one** `polity_code`, over 17,184 resolving pairs and 306 polities. A join that carries both columns is therefore already polity-correct, whether or not it names a polity, and the territorial identity is recoverable from the numeric code rather than lost by it.

The one enumerated exception is FAOSTAT area 7 (Angola) in 1975, where `ANG-1905-1975` records no successor upstream and so is widened by a year into `AGO-1975-2025`'s first year. Resolution still returns `AGO-1975-2025` there, but by row order rather than by the data. See #683.

`polity_area_code` is a bucket, and does not carry the contract

`polity_area_code` is the key rows are **aggregated on** for the matrix workflows, and several `area_code` values can share one. Where a bucket has one member, or its members agree, `(polity_area_code, year)` recovers the polity too. Over 1961-2025 exactly one bucket does not: **206**, which holds Sudan (former) 206, Sudan 276 and South Sudan 277 and answers with three polities in all 65 reported years. See #414. To say which territory a row belongs to, read this table's own `polity_code`, or resolve through `add_polity_code()`; do not infer it from the bucket. In a WHEP *output* the same answer is `reporting_polity_code`, materialised from this table's `polity_code`. This table carries no column of that name, and `legacy_polity_prefix` is not a substitute for one (#711).

A bucket that sums two territories gets a polity that means both

Every other row here answers for a **reporting area**, because that is what upstream's map is about. Bucket 206 from 2012 is the one place that is not enough: FAOSTAT reports area

206 through 2011 and areas 276/277 from 2012, WHEP sums the two successors back into bucket 206, and the bucket's own code had nothing later than `SUD-1956-2011` to answer with – a polity that ended at the secession, reported `out_of_span` on every post-2011 row (#414).

Upstream now publishes the entity that means the sum, `F206-2011-2025` "Sudan and South Sudan (combined reporting)", `aggregate`, with a constructed polygon, on the same `F<area>` naming as `F237-1954-1975` (Vietnam) and `F249-1918-1990` (Yemen). It has no map row, correctly: as a claim about the reporting area it would be false. `data-raw/table_mappings.R` therefore adds one row per bucket that genuinely sums several territories and has no aggregate label of its own, deriving both the bucket-years and the polity from the two upstream tables rather than naming either by hand (#860).

The row answers from `applies_from_year` (2012), not from the polity's own 2011 start: at 2011 bucket 206 is still area 206 reporting alone, and upstream's map decides that year. So `(206, 2011)` stays `SUD-1956-2011` and only the years the bucket really is a two-territory sum move.

The shape is unusual and deliberate: this aggregate is live **at the same time as its own members**, since areas 276 and 277 keep reporting separately. That is not a defect – `BLX-1850-1999` coexists with `BEL` and `LUX` for 149 years – and it makes bucket 206 the first bucket key carrying three live polities at once.

A bucket resolves here only because it is also an area

There is one key column, `area_code`, and callers hand it **two kinds of code**: a raw FAOSTAT reporting area, and a `polity_area_code` aggregation bucket – which is what `reporting_polity_code` is resolved from on every summed row. Nothing in a row says which of the two it answers for, and that works only because the bucket key space is a **subset** of the area key space: every `polity_area_code` is also an `area_code` here, in all three Rest-of-World modes. A bucket code that were not a reporting area would match no row, so the fold would keep its `area` label through the member fallback and lose `reporting_polity_code` to `NA` without complaint. The subset property is asserted by the test suite rather than assumed.

A row may legitimately answer in **both** spaces, which is why the two are not split. `(238, ETH-1952-1993)` is the case: as a bucket row it dates bucket 238, which sums FAOSTAT area 62 Ethiopia PDR for 1961-1992; as an area row it dates FAOSTAT area 238 itself for a historical source reported under its own year's borders, which is how 149 historical-trade rows resolve at 1961. Marking it for one space would move the other. See #742.
`(206, F206-2011-2025)` is the one row written **for** the bucket, and it answers in the area space too: raw FAOSTAT area 206 for 2012 onwards resolves to it. Nothing is misattributed by that, because FAOSTAT emits no area-206 row after 2011 – the claim is unexercised rather than false – but it is the first row for which the two spaces could be told apart at all, which is the column #742 asks for and this table still does not carry.

Confidence is the pair, not `mapping_status` alone

`"matched"` covers outcomes of very different confidence – a curated hit in upstream's published FAOSTAT map, a prefix-inferred period outside every span that map declares, a prefix guess for an area the map does not cover, and the FABIO Rest-of-World fold. `mapping_source` **is the column that separates them**, and it is non-`NA` on every row, so

read the two together rather than filtering on `mapping_status == "matched"` and assuming a curated decision:

```
dplyr::count(polity_area_crosswalk, mapping_status, mapping_source)
```

Curated rows are `mapping_source == "upstream_map"`; everything a prefix decided is `"prefix_outside_map"` or `"prefix_fallback"`. Prefix inference never overrides the map where the map speaks: a prefix-derived period whose years overlap any span the map declares for that area is dropped at build time, so the two branches cannot disagree about a reported year.

The row space is reporting areas, not polities

This table is **not an index of polities**, and reading a missing `polity_code` as a coverage gap is the mistake #875 records. The builder starts from the reporting areas of `regions_full.csv` and asks which polity each one names; a polity therefore has a row **only if some reporting area names it**. 176 of the 735 live polities have none, because no FAOSTAT or FABIO area was ever reported under their territory – 121 national, 33 colonial, 8 aggregate and the rest smaller units. The 559 that remain are exactly the distinct non-`NA` `polity_code` values here.

Absence is structural rather than accidental, and the test suite asserts it: not one of the absent aggregate polities carries a `legacy_polity_prefix` any reporting area carries, so the row space has no slot to put them in. A polity absent from here is reached by the **label** route instead – `resolve_polity_label()` over polity_label_aliases, which maps the label a historical source writes straight to a polity and needs no area code. That is how five of the eight absent aggregates resolve – `GCT-1919-1956` Gold Coast and British Togoland, `MASG-1946-1963`, `SYL-1944-1953` (all scoped to the `fao1952` yearbook), `PAPNG-1920-1949` (`mitchell`) and `AOI-1936-1941` (any source) – all of them pre-FAOSTAT combined reporting units that never had an area code. Two more (`EGYSUD-1934-1956`, `CODRU-1922-1960`) are upstream composed-union identities held for footnote series that fold a colony into its metropole, and upstream has registered no label for them yet, so they are reachable from neither route by design rather than by omission.

No absence here can NA an `area_code` on a polycell, either: `build_polycell_support()` excludes `polity_type == "aggregate"` by type before any lookup runs, so an aggregate emits no polycell to carry one.

`F206-2011-2025` used to be the one absence that was a real gap, and it is filled: it now has a row, keyed on the bucket rather than on a reporting area, so `polity_bucket_coverage()` classifies bucket 206 `"aggregate"` from 2012 instead of `"predecessor"`. See the bucket section above; #742's `key_role` column, which would mark a row as answering in only one of the two key spaces, is still open and is not what `applies_from_year` does.

Source

Derived from polities, `~/whep-polities/data/final/faostat_area_polity_map.csv` and `inst/extdata/harmonization/regions_full.csv`.

`polity_bucket_coverage` *Report reporting buckets that sum more than one territory*

Description

`polity_area_code` is an aggregation bucket, not an identity: FABIO folds several FAOSTAT reporting areas into one numeric code, and WHEP's builds sum them under it. `reporting_polity_code` is then resolved from that bucket code, so a bucket folding more than one live territory can end up labelled with a polity that covers only part of what the value covers, or that covers the whole of it but in a period that has ended.

This lists every `(polity_area_code, year)` that folds more than one polity and classifies whether the bucket's own label covers the fold:

- `"aggregate"`: the bucket resolves to an aggregate polity (Rest of World, Belgium-Luxembourg, the FAOSTAT combined-reporting entities), whose name and polygon already mean the union of its members. Honest.
- `"predecessor"`: the bucket is labelled with a polity that has **ended**, and whose published `successor` set is exactly the set of polities the bucket folds. The extent is right — that predecessor's territory is the union of its successors — but the period is not, so a consumer filtering polities by span drops the rows.
- `"partial"`: the bucket sums several territories but is labelled with a polity covering only part of them, so the value and its polity describe different extents. This is the worst case, and no bucket is in it today.
- `"unlabelled"`: the bucket code resolves to no polity, so rows carry `NA` and the gap is at least visible rather than wrong.

An area counts as a member only in the years it **reports**: its polity must be in span, and the upstream FAOSTAT map must report the area that year. A year-aware lookup answers every `(area_code, year)` pair regardless, standing in with the nearest period, so asking it about an area that does not report in that year invents a member. FAOSTAT reports area 206 for 1961-2011 and areas 276/277 for 2012-2024, never in the same year, so counting the stand-ins reported bucket 206 as a three-way fold in all 65 years rather than a two-way fold in the 14 it is one (whep#414).

Bucket 206 is the one fold reported today, and it is `"aggregate"` for 2012-2025: it sums FAOSTAT areas 276 Sudan and 277 South Sudan and resolves to `F206-2011-2025` "Sudan and South Sudan (combined reporting)", the aggregate whep#860 wired in. It read `"predecessor"` before that, labelled with `SUD-1956-2011`, a polity that had ended.

So the labelling is settled and only the regionalisation is still open: whether to keep folding the two successors at all is whep#680, and `options(whep.unfold_predecessor_bucket = "all")` is the switch that promotes them. It is off by default because it is not mass-neutral — see `folded_reporting_areas()` for what it withdraws and why. `.aggregate_to_polities()` warns when it builds such a bucket; set `options(whep.warn_polity_folds = FALSE)` to silence that warning.

The polity reported here as the bucket's own is also the `area` label the builds attach to the summed row, and the one the reporting columns resolve. A bucket carries one label

whatever its members resolve to, because `area` is a join key and a bucket under two labels stops summing (whep#563).

Usage

```
polity_bucket_coverage(years = NULL)
```

Arguments

`years`	Integer vector of years to classify. Defaults to the FAOSTAT reporting era, 1961 to 2025. Years before the back-cast anchor resolve to the anchor-year territory, so they classify identically to 1961.

Value

A tibble with one row per folded `(polity_area_code, year)`, with the folded member polities, the polity the bucket itself resolves to, and the `coverage` classification. Zero rows means no bucket folds more than one polity in the requested years.

Examples

```
polity_bucket_coverage(years = 2015L)
```

`polity_containment` *Polity containment edges*

Description

Published member-container containment edges between periodized WHEP polities: which admin/subnational polity sits inside which container polity, and for what years. This is the piece polity_area_crosswalk and polity_label_aliases do not carry – both map a *reporting area* or a *label* to a polity, while this table relates polities to each other. Any WHEP admin unit (a province, a prefecture, ...) is already a `polity_type == "subnational"` row of polities; the containment edge is the missing link between it and the polity it belongs to (whep-polities#649, closing whep#51).

Usage

```
polity_containment
```

Format

A tibble with one row per containment edge. Columns:

- `member_code`: `polity_code` of the contained polity.
- `container_code`: `polity_code` of the containing polity.

- `start_year`, `end_year`: Validity interval of the edge itself, on the same convention as polities: `start_year` is inclusive and `end_year` is exclusive at a succession (a later edge for the same member takes over from `end_year`) and inclusive at the open end (an edge nothing succeeds still covers its own terminal year). The interval belongs to the edge, not to either endpoint polity: `JPN-23-1871-2025` is one member polity spanning 1871-2025, but four separate edges route it through `JPN-1800-1895`, `JPN-1895-1945`, `JPN-1945-1952` and `JPN-1952-2025` as Japan's own periodization changes underneath it.
- `basis`: Free-text description of what the containment relationship represents (a historical territory, a reporting convention, ...).

Details

The map is authored and gated in `whep-polities`; this package embeds a copy rather than deciding containment itself, on the same grounds as polity_label_aliases and polity_area_crosswalk.

Source

`~/whep-polities/data/final/polity_containment.csv`.

`polity_coverage_gaps` *Find rows attributed to a polity not live in the row's year*

Description

`add_polity_code()` reports these rows in `mapping_status`, but WHEP's published outputs do not carry that column: `reporting_polity_code` and `reporting_polity_name` say which polity a row was attributed to, and nothing says the polity did not exist in that row's year. This answers that question for a table that has already been built, so a consumer joining on `reporting_polity_code` can tell a real period hit from the two kinds of substitute without re-deriving the crosswalk.

Neither kind is an error and no row is dropped. It means the polygon, population and period of the returned polity describe a different year than the value does, so `gap_kind` names which kind a row is:

- `"backcast_anchor"`: the row is before `backcast_anchor` and its polity was resolved at the anchor year, which is WHEP's own back-cast convention – pre-1961 series are reconstructions on the anchor year's territory, so a Soviet republic's 1900 land is booked to the republic that reports it today. The polity was matched at the anchor and simply had not begun by the row's own year.
- `"polity_not_started"`: no mapped period covered even the anchored year and the stand-in taken begins after it.
- `"polity_ended"`: the polity had ended, so the value covers a territory that entity no longer describes. This is the harder case: FAOSTAT area 51 "Czechoslovakia" resolves to `F51-1947-1993` whatever year it is asked for, because nothing later is mapped to that area. Bucket 206 used to be the headline instance – areas 276 Sudan and 277 South Sudan fold into it and its label `SUD-1956-2011` had ended at the secession (whep#414)

– and is no longer one: upstream minted `F206-2011-2025`, live over exactly the years the bucket sums both, and whep#860 wired it on.

`gap_kind` is not derivable from the returned columns, which is why it is returned rather than left to the caller. `"backcast_anchor"` is not visible in the years at all – the resolver matched a real period, at the anchor – and the direction of the other two is read at the year the resolver actually matched on, which the back-cast anchor floors at `backcast_anchor`, so a pre-anchor row is classified as the anchor year it was resolved as rather than as the year it carries.

Measured on a real full-range `get_primary_production()`: 2,301 `(area, year)` pairs / 7,247 rows are stand-ins, and the back-cast class adds 9,544 pairs the floor previously hid from this function entirely (whep#763).

The resolution here is the same one the builds use, including the back-cast anchor, so it reports what the table actually got rather than a second reading of the crosswalk. The area column may hold either a FAOSTAT `area_code` or the `polity_area_code` bucket that published outputs are keyed by; both resolve through the same lookup.

Usage

```
polity_coverage_gaps(
  table,
  code_column = "area_code",
  year_column = "year",
  backcast_anchor = 1961L
)
```

Arguments

`table`	A data frame carrying an area-code column, and a year column if the resolution is to be year-aware.
`code_column`	Name of the column holding numeric area codes.
`year_column`	Name of the column holding years. Set to `NULL`, or leave it absent from `table`, to use the current/default mapping, which has no gaps by construction.
`backcast_anchor`	First year of reported (non-back-cast) FAOSTAT data; passed to the same resolution `add_polity_code()` documents. The territory a pre-anchor row is resolved to follows `options(whep.backcast_polity = )`, as the builds' labels do.

Value

A tibble with one row per reported `(area_code, year)`, ordered by area code and year, carrying `area_code`, `year`, `polity_code`, `polity_name`, `polity_start_year`, `polity_end_year`, `gap_kind` (`"backcast_anchor"`, `"polity_not_started"` or `"polity_ended"`) and `n_rows`, the number of rows of `table` that pair carries. Zero rows means every row of `table` landed inside its polity's period, which is the intended state.

See Also

add_polity_code() for the resolution itself, polity_bucket_coverage() for the related question of which buckets sum more than one territory, and whether their label covers the sum, and polity_anchor_drift() for the narrower question of whether a back-cast row's label describes the same territory its growth proxy was measured on.

Examples

```
# FAOSTAT area 51 "Czechoslovakia" is the live case: `F51-1947-1993` ended in
# 1993 and nothing later is mapped to the area, so a post-1993 row is a
# stand-in. Area 238's 1850 row is the back-cast case: `ETH-1952-1993` labels
# it because that is the polity live at the anchor, 102 years later.
polity_coverage_gaps(
  tibble::tibble(
    area_code = c(51L, 51L, 238L),
    year = c(1990L, 2015L, 1850L),
    value = 1
  )
)
```

---

polity_identity_conventions

Which territorial identity WHEP's year-less objects carry

Description

A WHEP polity code is year-scoped: `ESP-1846-1914` and `ESP-1978-2025` are different rows of polities. Several WHEP objects have no year dimension at all – the cell/area grids, the static crop-pattern weights, coefficient tables keyed by a country label – so "attach the polity code" has no single answer for them. This is the register of the answer each one carries, so that the choice is stated rather than left to whichever join a consumer happens to write.

It decides nothing at call time: the table is a declaration, and `tests/testthat/test_territorial_identity.R` checks every row of it against the object it describes, including that no year-less territory-keyed dataset is missing from it.

Usage

```
polity_identity_conventions(kind = NULL)
```

Arguments

`kind`	Optional character vector restricting the result to one or more of `"package_data"`, `"input_pin"` and `"function_output"`. `NULL` (default) returns every row.

Value

A `tibble` with one row per registered object and the columns:

- `object`: The dataset name, pin alias or function it describes.
- `kind`: `"package_data"`, `"input_pin"` or `"function_output"`.
- `territory_key`: The column(s) naming a territory, comma-separated, or `NA` when the object names none.
- `identity`: `"present_day_polity"`, `"polity_period"` or `"identity_free"`.
- `status`: `"carried"` when the object materialises the identity today, `"resolved_by_consumer"` when the consumer attaches it with its own year, `"recommended"` when this is the proposed convention and not yet the implemented one.
- `carrier`: The column holding the polity code, `NA` when there is none.
- `resolver`: The supported call that attaches identity, `NA` when none is needed.
- `rationale`: One sentence saying why.

The three conventions

`"present_day_polity"` The object is a registry of what reports *today*, so the present-day polity is what it means. Its `carrier` column holds a real polities code, the one `add_polity_code(year_column = NULL)` resolves – the crosswalk period running to the open end.

`"polity_period"` One row of the object covers territory that changed hands, so the object needs a validity interval and has to become year-aware. `build_polycell_support()` on the `edu/polycell-spatial-support` branch is the worked precedent for the grid: one row per (cell, polity, validity interval).

`"identity_free"` The object is a coefficient about a *place*, or a label whose referent changes with the year it is applied in, so carrying one polity code would be false precision. Identity is attached at the consumer, with the consumer's year, through the `resolver` call.

Why a year-less object cannot simply be handed a code

Measured on the deployed `spatialize-country-grid` pin (58,795 cells, 178 area codes, no year): **52,420 cells, 89.2%, sit under an `area_code` that polity_area_crosswalk maps to more than one polity over time**, so which polity a cell belongs to is a function of the year the grid does not have. The same holds for country labels resolved through `resolve_polity_label()`: 33 of mueller_synthetic_n's 156 `iso3c` values, 37 of crops_manure_n's 184 `ISO` values and 38 of `gleam_geographic_hierarchy`'s 204 `iso3` values name a *different* polity at 1961 than at 2020.

What counts as a territory here

A supra-national statistical grouping – an IPCC or GLEAM region, a UN sub-region, `region_krausmann` – is **not** a territory and is out of scope by construction: it names a class of places, never a state, so it can never acquire a polity code. Only the column names in `whep:::.territory_key_names()` put an object in this register. That is also why a bare `code` is not one of them: `biomass_coefs$Code` is a crop.

The trap this register exists to make visible

regions_full and polities_cats carry a column of ISO3-like stems (`"AFG"`, `"ROW"`, `"RAFR"`) kept for older callers, of which **not one value is a polities code**. Until whep#687 that column was literally named `polity_code`, so a join from either dataset to polities or polity_area_crosswalk on the one column whose name promised identity came back empty and nothing warned. It is now `legacy_polity_prefix`, which claims nothing. Their real carrier is `reporting_polity_code`, a polities code on all 259 of `regions_full`'s non-`NA` rows and all 198 of `polities_cats`'s.

polity_area_crosswalk vendors the same stems, and shipped them under `reporting_polity_code` — the name that means the opposite — until whep#711 renamed them `legacy_polity_prefix/legacy_polity_n` to match. The register does not reach that table, because it enumerates year-*less* objects and the crosswalk carries a year; the guard for it lives in `tests/testthat/test_territorial_identity.R` alongside #687's.

See Also

`add_polity_code()` for the numeric-code route, `resolve_polity_label()` for the label route, and whep_polity_columns for what a year-*aware* WHEP output carries.

Examples

```
polity_identity_conventions() |>
  dplyr::select(object, kind, identity, status)

polity_identity_conventions(kind = "package_data") |>
  dplyr::select(object, carrier, resolver)
```

`polity_label_aliases` *Source label to polity aliases*

Description

Published map from the country/area **labels** a source writes to periodized WHEP `polity_code` values, consumed by `resolve_polity_label()`. It is the label-keyed counterpart of polity_area_crosswalk, which is keyed by numeric reporting area code.

Usage

```
polity_label_aliases
```

Format

A tibble with one row per alias. Columns:

- `source_label`: The label exactly as the source writes it.
- `source`: Source slug the alias is scoped to, or `NA` when it applies to any source.
- `year_start`, `year_end`: Year range the alias is scoped to. A missing bound is unbounded on that side; both missing means the alias is not year-scoped.

- `polity_code`: The WHEP polity the label resolves to.
- `common_name`: Human-readable name of that polity.
- `confidence`: Curator's confidence in the alias.
- `observed_rows`: Source rows actually observed for the label, `NA` when the label is merely mappable.
- `disposition`: `NA` when the source observed the territory, `"back_cast"` when the years are a reconstruction onto a boundary that did not exist yet. A `"back_cast"` alias may begin before its target polity does, by design: the polity's own span still starts when the territory did. `resolve_polity_label()` drops these rules when asked `back_cast = FALSE`. All `NA` in a snapshot taken before whep-polities introduced the column.
- `indicator`: `NA` when the alias applies to every indicator, otherwise the one indicator it routes (`"area"`, `"production"`, `"yield"`, `"livestock_stock"` or `"landuse"`, the subnational panel's own values). whep-polities #703 added it so that a panel unit whose id names different territories for different indicators can be split; it allows the scope only on the panel's slugs. `resolve_polity_label()` applies a scoped rule only to rows passing that `indicator`. Absent from a snapshot taken before #703, which is read as all `NA`.

Details

The map is authored and gated in `whep-polities`; this package embeds a copy rather than deciding label identity itself, so that a label's meaning has one authority.

Source

`~/whep-polities/data/final/label_alias_map.csv`.

`polity_label_item_corrections`

Source label corrections scoped to one item

Description

Rows a source files under ANOTHER territory's label for one item, published by `whep-polities`. An alias maps a label to a polity with no item dimension, so it cannot say that Mitchell's pre-1910 `"south africa"` sugar cane is Natal's while the other items under that label are the Cape's. Each rule replaces the label before resolution; `resolve_polity_label()` applies them when it is given `item`.

Usage

```
polity_label_item_corrections
```

Format

A tibble with one row per rule. It has zero rows when the shipped snapshot was taken from a `whep-polities` revision that published no rules. Columns:

- `source`: Source slug the rule applies to.
- `source_label`: The label the source files the rows under.
- `item`: The item, exactly as the source writes it.
- `year_start`, `year_end`: Inclusive year range.
- `unit`, `indicator`: Scope of the rule. `NA` (blank upstream) means any; when set, the row's own unit / indicator must equal it. All `NA` in a snapshot taken before whep-polities introduced the columns.
- `correct_label`: The label to resolve instead.
- `polity_code`: Where `correct_label` resolves in the same upstream revision, or `"UNROUTED"` when the rows belong to no polity and `resolve_polity_label()` leaves them unassigned.
- `observed_rows`: Source rows the rule relabels upstream.
- `issue`: The `whep-polities` issue that decided the rule.
- `evidence`: Why the rows belong to the other territory.

Details

A row matches a rule when `source`, `source_label` (normalised as the alias map's labels are) and `item` all match and the row's year lies in `[year_start, year_end]`. Rows without a year are never corrected, and rules do not chain. A rule with `unit` or `indicator` set also needs the row's own unit / indicator to equal it.

Source

`~/whep-polities/data/final/source_label_item_corrections.csv`.

`polity_mapping_provenance` *Report which authority a row's territorial identity rests on*

Description

polity_area_crosswalk is not the upstream FAOSTAT-to-polity map. It is that map plus rows this package manufactures, and its `mapping_source` column says which. Measured on the shipped snapshot, of 648 crosswalk rows:

- `"upstream_map"` (245 rows): a row of `faostat_area_polity_map.csv` in `eduaguilera/whep-polities`. Upstream's statement about the territory.
- `"fabio_row_promoted"` (52): the same file, for the 47 areas the FABIO Rest-of-World fold used to shadow. Equally upstream's statement, and kept separate only because `.unfold_rest_of_world()` chooses between it and the fold row (whep#717). The two together consume the map's 297 rows exactly once.

- `"prefix_outside_map"` (263) and `"prefix_fallback"` (26): WHEP's own ISO3-prefix match, built in `data-raw/table_mappings.R`. No upstream authority. A prefix match can only ever produce an ISO3-family guess, so it cannot express the statements the pre-1961 era actually needs – Turkey before 1913 is the Ottoman Empire, Pakistan before 1947 is British India – whose target stem differs from the source's (whep#740).
- `"fabio_row_fold"` (62): WHEP's own Rest-of-World bucket. Legitimately WHEP's to decide, but the territory still has no upstream row of its own. In the default mode only the 31 members upstream names nowhere resolve through one; see `row_promotion_status()`.
- `"whep_bucket_aggregate"` (1): WHEP's own row for an aggregation bucket rather than for a reporting area, naming the upstream aggregate polity that means the union of the areas the bucket sums. Bucket 206 from 2012 is the only one, on `F206-2011-2025` "Sudan and South Sudan (combined reporting)" (whep#860). The polity is upstream's; the statement that it answers for the bucket is WHEP's, because the bucket is WHEP's.

Counting crosswalk rows overstates the exposure, because most manufactured rows are never picked. This reports the provenance of the **resolution**: for each `(area_code, year)`, the class of the crosswalk row that `add_polity_code()` actually selected, which is what a published value rests on. Measured over the crosswalk's own 1850-2025 grid, 259 of the 263 `"prefix_outside_map"` rows are the resolution of no `(area_code, year)` at all: the back-cast anchor floors every lookup at `backcast_anchor`, so the pre-1961 era resolves through whatever answers the anchor year rather than through the historical periods the prefix rule invented for it.

The same measurement is what whep#717 moved: 10,912 of those `(area_code, year)` resolutions used to rest on `"fabio_row_fold"`, and 5,456 of them – every year of the 31 members upstream names – now rest on the upstream map instead.

Usage

```
polity_mapping_provenance(
  table = NULL,
  code_column = "area_code",
  year_column = "year",
  backcast_anchor = 1961L
)
```

Arguments

`table`	A data frame carrying an area-code column, and a year column if the resolution is to be year-aware. `NULL` (the default) reports over the crosswalk's own `(area_code, year)` grid instead, one row per pair, which is the provenance of the mapping rather than of a build.
`code_column`	Name of the column holding numeric area codes.
`year_column`	Name of the column holding years. Set to `NULL`, or leave it absent from `table`, to use the current/default mapping.
`backcast_anchor`	First year of reported (non-back-cast) FAOSTAT data; passed to the same resolution `add_polity_code()` documents. Set to `-Inf` to resolve strictly

by data year, which is what makes the manufactured pre-1961 periods live. The territory a pre-anchor row is resolved to follows `options(whep.backcast_polity = )`, as the builds' labels do.

Value

A tibble with one row per resolved `(area_code, year)`, ordered by area code and year, carrying `area_code`, `year`, `polity_code`, `mapping_source`, `authority`, `mapping_status` and `n_rows`, the number of rows of `table` that pair carries (always 1 when `table` is `NULL`).

Authority

`authority` collapses `mapping_source` to the question "who said so":

- `"upstream"`: `"upstream_map"` or `"fabio_row_promoted"`.
- `"whep_prefix"`: `"prefix_outside_map"` or `"prefix_fallback"` – a WHEP guess, and the population whep#740 asks to delete rather than replace.
- `"whep_bucket"`: `"fabio_row_fold"` or `"whep_bucket_aggregate"` – WHEP's own documented buckets. Both say the bucket is WHEP's to name; only the second names it with a polity upstream publishes.
- `"unresolved"`: the area resolves to no polity, so nothing was said.

An unrecognised `mapping_source` aborts rather than being folded into one of these, so a new class of manufactured row cannot arrive already classified.

See Also

`add_polity_code()` for the resolution itself, `polity_coverage_gaps()` for the rows whose polity is a nearest-period stand-in, and `polity_bucket_coverage()` for the buckets that sum more than one territory.

Examples

```
# Area 238 Ethiopia is the one reporting area whose published identity rests
# on a manufactured row: its pre-1993 years resolve to `ETH-1952-1993`, a
# period no upstream map row declares.
polity_mapping_provenance(
  tibble::tibble(area_code = 238L, year = c(1900L, 2000L), value = 1)
)

# The headline is one summarise away.
polity_mapping_provenance(
  tibble::tibble(area_code = c(11L, 238L), year = 1990L)
) |>
  dplyr::summarise(n_rows = sum(n_rows), .by = authority)
```

polycell_example_geometries

A minimal polity geometry table for examples and smoke tests

Description

Returns one live polity holding a rectangle that spans six 0.5-degree cells, in the shape `get_polity_geometries()` returns: enough to run `build_polycell_support()` end to end in a fraction of a second, with no pins, no rasters and no environment variables.

Usage

```
polycell_example_geometries()
```

Value

An `sf` table with `polity_code`, `polity_type`, `wiki_status`, `polygon_status`, `start_year`, `end_year`, `area_code` and a `geom` multipolygon in WGS84.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  polycell_example_geometries()
}
```

population_source_reach

Report which areas a present-day-ISO3 population source can reach.

Description

Every population source WHEP reads is keyed on a present-day ISO3 code: the `gdp-population` pin and UN WPP 2024 both are. A WHEP `area_code` names a territory in the years it reported, and for a dissolved territory no present-day ISO3 stands for it, so the source has no row and the area drops out of every per-capita output (see `read_population()` and `build_food_supply()`).

This reports, for each reporting period of each area code, whether the vocabulary in `iso3_codes` reaches it:

- `"direct"` — the period's own ISO3 is in the vocabulary.
- `"successor"` — it is not, but the `successor` relation the polities database publishes leads, transitively, to ISO3 codes that are. Those codes are returned in `iso3_reached`.
- `"unreachable"` — neither. No arrangement of the source's rows supplies this area.

Measured against UN WPP 2024's 237 `Country/Area` ISO3 codes, 14 reporting periods are not `"direct"` and five are `"unreachable"` — `ATF`, `PCN`, `SGS`, `NFK` and `CXR`, every one of them inside the Rest-of-World bucket 999. `ANT-1961-2010`, area 151 Netherlands Antilles, was the last `"unreachable"` outside that bucket (#787); the 2026-08-25 upstream re-sync gave it `CUW-2010-2025; SXM-2010-2025; BES-2010-2025`, so it now reads `"successor"` by lookup rather than by anything hardcoded here.

`"successor"` says the ISO3 codes exist, **not** that summing them is a safe denominator. In general it is not: a successor sum for the Yugoslav SFR falls 17.5% short of the `gdp-population` pin's own figure for the same aggregate, of which Kosovo is 42% and a level disagreement between the pin and WPP on that series is the rest. Use this to see what a source can cover, and read the note at the top of `R/population_reach.R` before turning any of it into a population.

`extent_exceeds_iso3` marks the periods where the codes reported are **narrower than the territory**, because the walk stopped on a polity that reuses one of its own parts' ISO3 — `SRB-2006-2008` "Serbia (including Kosovo)" carrying `SRB`, `SUD-1956-2011` carrying `SDN` (#863). It is orthogonal to `reach`: a `"direct"` period can be flagged, and that is the case worth knowing about, since nothing else about the row hints at it.

Rows on `area_code` 999 describe the **members** of the Rest-of-World fold bucket rather than the bucket itself, since each member is a crosswalk period of its own.

Usage

```
population_source_reach(iso3_codes, crosswalk = NULL)
```

Arguments

`iso3_codes`	Character vector of the present-day ISO3 codes the population source publishes, e.g. `unique(read_wpp_population()$iso3c)`.
`crosswalk`	Optional crosswalk overriding polity_area_crosswalk, for testing.

Value

A tibble with one row per `area_code` and `polity_code` reporting period, sorted by area code then first reported year: `area_code`, `polity_code`, `polity_name`, `map_year_start`, `map_year_end`, `own_iso3` (the period's own ISO3), `reach` (`"direct"` / `"successor"` / `"unreachable"`), `n_iso3`, `iso3_reached` (the ISO3 codes standing in for the area, `NA` when unreachable) and `extent_exceeds_iso3` (`TRUE` where those codes cover less ground than the period's territory).

Examples

```
population_source_reach(c("BEL", "LUX", "CZE", "SVK", "CUW"))
```

`prepare_livestock_emissions` *Prepare production data for livestock emission calculations.*

Description

Bridge between `build_primary_production()` output and the livestock emission functions. Maps species codes to names, converts area codes to ISO3, extracts milk and meat yields, and optionally expands herds into GLEAM cohorts.

Any extra columns present in the input (e.g., `weight`, `diet_quality`, `fat_percent`) are preserved and flow through to the emission functions automatically.

Usage

```
prepare_livestock_emissions(
  data,
  expand_cohorts = FALSE,
  system_shares = NULL,
  method_cohorts = c("milk_animals", "uniform")
)
```

Arguments

`data`	A tibble from `build_primary_production()` with columns `item_cbs_code`, `unit`, `value`, and optionally `year`, `area_code`, `live_anim_code`.
`expand_cohorts`	Logical. If `TRUE`, distributes herds across GLEAM cohorts and production systems via `calculate_cohorts_systems()`. Default `FALSE`.
`system_shares`	Optional dataframe with custom system shares. Passed to `calculate_cohorts_systems()`.
`method_cohorts`	How heads are split across the cohorts of a system: • `"milk_animals"` (default): a dairy-cattle herd (`"Cattle, dairy"`, the FAOSTAT Milk Animals count) goes wholly to the dairy system's `"Adult Female"` cohort; every other herd is split equally across its system's cohorts. • `"uniform"`: every herd, dairy cattle included, is split equally across its system's cohorts, the behaviour before whep#1127. It books five sixths of the milking cows as bulls, heifers and calves, and is kept for sensitivity analysis.

Value

A tibble with columns `species`, `heads`, `iso3` (if `area_code` present), and optionally `milk_yield_kg_day`, `method_milk_yield`, `weight_gain_kg_day`, cohort columns, plus all extra columns from the input. Milk is read for every milked species FAOSTAT reports (cattle, buffalo, sheep, goats, camels), not only for an animal's designated product, and with `expand_cohorts = TRUE` it sits on the milked cohort only (see `calculate_cohorts_systems()`).

Examples

```
tibble::tibble(
  item_cbs_code = 961,
  unit = "heads",
  value = 5000,
  area_code = 79L
) |>
  prepare_livestock_emissions()
```

---

primary_double *Items with double-counting in production statistics*

Description

Identifies production items that appear both as primary crop products and as harvested-area items, requiring special treatment to avoid double-counting in production and biomass accounting.

Usage

```
primary_double
```

Format

A tibble where each row corresponds to one item pair with a double-counting relationship. It contains the following columns:

- `Item_area`: Name of the item as it appears in harvested-area statistics (e.g., `"Seed cotton, unginned"`).
- `item_prod`: Name of the derived production item (e.g., `"Cotton lint, ginned"`, `"Cotton seed"`).
- `item_prod_code`: Numeric FAOSTAT production code of the derived item.
- `Multi_type`: Classification of the double-counting type:
  - `"Primary"`: The area item is the primary crop; product is a direct output.
  - `"Primary_area"`: Area is recorded under a primary aggregate crop name.
  - `"Multi"`: Multiple products share the same harvested area.
  - `"Multi_area"`: Multiple products share a recorded area aggregate.

Source

Derived from FAOSTAT production methodology documentation.

Examples

```
head(primary_double)
```

`propagate_fp_uncertainty` *Propagate input uncertainty through a footprint.*

Description

Monte Carlo propagation of extension uncertainty: perturb the extension vector with multiplicative lognormal noise (so values stay non-negative and the expected factor is one), re-run the footprint for each draw, and summarise the spread of each output cell. A point estimate with no interval is not a trustworthy result; this turns one into a distribution.

Usage

```
propagate_fp_uncertainty(run_fn, extensions, cov = 0.1, options = list())
```

Arguments

`run_fn`	Function taking a perturbed extension vector and returning a footprint tibble with the grouping columns named in `options$by` plus a `value` column. Wrap `compute_footprint()` with the other arguments fixed.
`extensions`	Numeric vector of base extensions per sector.
`cov`	Coefficient of variation of the extensions: one number, or one per sector. Zero means no uncertainty.
`options`	Named list overriding `n` (draws, default 200), `probs` (lower/median/upper quantiles), `by` (grouping columns) and `seed` (for reproducible draws). A `seed` is scoped to this call: the caller's `.Random.seed` is restored on exit, so drawing here does not reshuffle the surrounding session's RNG stream.

Value

A tibble with the `by` columns plus `mean`, `sd`, `cv`, `q_low`, `q_med` and `q_high` per output cell.

Examples

```
run_fn <- function(ext) {
  tibble::tibble(
    target_area = 1L, target_item = 10L, value = sum(ext)
  )
}
propagate_fp_uncertainty(
  run_fn,
  extensions = c(60, 40),
  cov = 0.1,
  options = list(n = 100, seed = 1, by = c("target_area", "target_item"))
)
```

`read_admin_family` *Read the per-family subnational admin-statistics pins*

Description

Read the tier-2 and tier-3 administrative-statistics families that WHEP compiled for the subnational spatialization, one pin per family, and return them as a named list of tibbles.

Every family is read by an explicit alias: there is no discovery step and no wildcard, so a family that is not shipped cannot be silently substituted by another. A requested family that is absent – its alias is not registered in `whep_inputs`, or its pin holds no rows – is named in the returned `not_shipped` element instead of aborting the read, because a family whose publication consent has not cleared is a planned state of this dataset (see the T23 outcome recorded in `inst/extdata/admin_stats_pins_manifest.csv`). Any other failure of `whep_read_file()` is left to propagate: an unreachable board is not an embargo.

Usage

```
read_admin_family(families = NULL, example = FALSE)
```

Arguments

`families`	Character vector of family aliases to read. `NULL`, the default, reads all five. An alias outside that set aborts.
`example`	If `TRUE`, return a small fixture instead of reading any pin. Defaults to `FALSE`.

Value

A named list with one element per requested family that is shipped, each a tibble in the shared admin-statistics reader shape (`source`, `source_native_unit_id`, `source_native_unit_name`, `source_native_item_code`, `source_native_item_name`, `indicator_used`, `quantity`, `year`, `value` or `share`, `value_unit`, `value_flag`, `grain`, `nuts_version`, `source_version`, `recorded_at`), plus a `not_shipped` element holding the aliases that were requested but are not on the board. `not_shipped` is always present, and is `character(0)` when every requested family loaded.

Families

The five aliases, their tier and what each ships, as staged on 2026-09-03 from the harmonized subnational panel:

- `"admin-stats-japan"` (tier 2): 46 prefectures, 32,095 observed area and production rows, 1961-2022.
- `"admin-stats-spain-provinces"` (tier 2): 50 provinces, 323,469 observed-lane area and production rows, 1961-2021. The compilation's Spanish livestock rows are all flagged `admin_level_overlap` and are therefore not shipped, which is why three of its 53 units are absent.

- `"admin-stats-australia"` (tier 2): 8 states and territories, 6,597 observed area, production and head-count rows, 1961-2022.
- `"admin-stats-france-livestock"` (tier 2): 89 departments, 58,740 observed head counts, 1961-2020.
- `"admin-stats-latam"` (tier 3): 142 first-level units of the Infante-Amate, Urrego-Mesa, Badia-Miro and Aguilera panel, 875,514 rows, 1961-2023, shipped as **shares only**. Each row carries `share`, its unit's share of the country total for the same item, indicator and year, and no `value`; this reader aborts if a future version of that pin carries one.

`"admin-stats-jrc"` is not one of these. It is the public tier-1 JRC product with its own reader, and passing it here aborts as an unknown alias.

Examples

```
read_admin_family(example = TRUE)
```

---

read_admin_shares *Read the assembled subnational admin-shares pin*

Description

Read the single `admin-shares` pin – the tier-1, tier-2 and tier-3 administrative statistics assembled onto the one contract `admin_shares_schema()` declares – together with the report of which families are **not** in it and why.

Polity resolution is deliberately not done here. The pin is keyed on source-native identifiers and carries `level_polity_code == NA` on every row; a caller that needs polities passes the rows through `resolve_admin_units()`, which redoes the resolution against the current polities snapshot on every load. That is what lets this artifact age without going stale.

That call needs one rename first. The contract names the identifier column `source_native_id`, where `resolve_admin_units()` takes the readers' `source_native_unit_id`, so the composition is `dplyr::rename(shares, source_native_unit_id = source_native_id)` and then `resolve_admin_units()`. Handing the rows over unrenamed aborts on the missing column.

Usage

```
read_admin_shares(example = FALSE)
```

Arguments

`example`	If `TRUE`, return a small fixture instead of reading the pin. Defaults to `FALSE`.

Value

A named list of three elements:

- `shares`: the pin's rows, in `admin_shares_prototype()`'s shape, or the zero-row prototype when the pin is not shipped.
- `excluded`: a tibble of `source`, `reason` and `detail`, as above.
- `not_shipped`: `"admin-shares"` when the pin's alias is not registered in `whep_inputs` or holds no rows, `character(0)` otherwise.

What the excluded report can and cannot say

`excluded` names every tier-2/3 family of `read_admin_family()` that contributed no row to this pin, with a reason from a closed vocabulary and a `detail` saying why the family is absent – not what permission it ships under, which is the manifest's own job:

- `"no_consent_manifest_row"`: the family is absent from `inst/extdata/admin_stats_pins_manifest.csv`, so its publication consent is not recorded and the assembly refused to ship it.
- `"no_rows_in_pin"`: its consent is recorded, but none of its rows survived onto the contract.
- `"pin_not_read"`: no pin was read at all – the alias is not registered in `whep_inputs`, or the board returned no rows – so nothing about this family was measured. Every family carries this reason together, and `not_shipped` says the same thing.

A read cannot see which rows were dropped or why, so for any other family `detail` points at the assembly's per-source counts. Those counts are a **second, build-time report**, written by `inst/scripts/prepare_admin_shares_pin.R` and not carried into the pin. It shares this vocabulary where the two can mean the same thing (`"no_consent_manifest_row"`, `"no_rows_in_pin"`) and adds one reason a read cannot observe: `"not_available"`, a consented family that was neither registered on the board nor staged locally when the pin was assembled. That is a fact about an assembly this read never saw, so it stays build-time only. What a read can always tell apart is whether it read a pin at all, which is what `"pin_not_read"` says – the report used to claim five families had been read and filtered when the board had not been touched.

Examples

```
read_admin_shares(example = TRUE)
```

---

read_admin_stats_eurostat

Read Eurostat regional crop and livestock statistics

Description

Read one of Eurostat's four regional agricultural dataflows over the SDMX-CSV dissemination API and return its rows in the source's own identifiers: the NUTS code and label exactly as Eurostat served them, the crop or animal code and label exactly as Eurostat served them, and the value converted to WHEP units. Nothing is mapped to a WHEP item code and nothing is resolved to a polity here; those are separate steps, so a change in either leaves this reader untouched.

Tables, verified live on 2026-09-02:

- `"apro_cpshr"` – crop production by NUTS 2 region, 2000-2024.
- `"apro_cpnhr_h"` – the historical companion, 1975-1999.
- `"apro_mt_ls_r"` – animal populations by NUTS 2 region, 1977-2025. It carries dairy cows (`A2300F`) and non-dairy cows (`A2300G`) separately but **no poultry**.
- `"ef_lsk_poultry"` – farm-structure-survey poultry by NUTS 2 region, census years only (2005, 2007, 2010, 2013, 2016, 2020, 2023).

No registration or key is needed. The dataflow identifier is checked against Eurostat's own dataflow registry before any data request, so a renamed table aborts naming the tables this reader knows instead of returning an empty tibble.

Usage

```
read_admin_stats_eurostat(
  table = c("apro_cpshr", "apro_cpnhr_h", "apro_mt_ls_r", "ef_lsk_poultry"),
  filters = list(),
  nuts_level = 2L,
  file = NULL,
  example = FALSE
)
```

Arguments

`table`	Dataflow to read, one of `"apro_cpshr"`, `"apro_cpnhr_h"`, `"apro_mt_ls_r"` or `"ef_lsk_poultry"`.
`filters`	Named list narrowing the request, any of: `geo` (NUTS codes, e.g. `c("FR", "FRF2")`), `items` (crop or animal dimension codes, e.g. `"C1110"` or `"A2000"`) and `years` (integer years; the request spans their range and the result is filtered to them exactly). An absent or empty element means "everything Eurostat serves", which for `"ef_lsk_poultry"` is large.
`nuts_level`	Integer NUTS levels to return as units, a subset of `1:3`. Defaults to `2L`, the level every one of these tables is published at.
`file`	Path to an already-downloaded SDMX-CSV response to parse instead of calling the API. It must have been requested with `label=both`, since the version suffix is read from the labels.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble of source-native rows, one per unit x item x measure x year, with columns `source` (`"Eurostat_<table>"`), `source_native_unit_id`, `source_native_unit_name`, `source_native_item_code`, `source_native_item_name`, `quantity` (`"area"`, `"production"` or `"heads"`), `indicator_used` (the admin-shares indicator vocabulary for area and production rows, `NA` for head counts, which that vocabulary does not cover), `year`, `value` (converted to hectares, tonnes or head), `value_unit`, `value_flag` (Eurostat's observation flag and confidentiality status verbatim, joined by `"|"` when both are present, `NA` when clean), `concept_break`, `grain`, `nuts_level`, `nuts_version`, `source_version` (the response's `LAST UPDATE` stamp verbatim) and `recorded_at`. The NUTS 0 rows are attached as the `"national_check"` attribute, in the same columns with `grain = NA`.

Units and the values this reader converts

Eurostat serves crop areas in thousand hectares (`AR_THS_HA`, `MAR_THS_HA`), crop production in thousand tonnes (`HPRD_THS_T`, `HPRD_HUMD_EU_THS_T`) and animal populations in thousand head (`THS_HD`) or head (`HD`). Every response carries those units in the measure dimension's own label, so the conversion factor is not a constant this package asserts from outside: the reader checks each response's label against the unit it is about to assume and aborts if Eurostat changes it.

Humidity (`HUMD_PC`, `HUMD_EU_PC`) and yield (`YLD_HUMD_EU_T_HA`) rows are dropped and counted. Yield is exactly production divided by area, both of which this reader returns, and its tonnes-per-hectare unit is outside this contract's `value_unit` vocabulary.

What the area figures mean, and the 2025 break

Eurostat's own metadata (ESMS `apro_cp_esms`, section 3.4, metadata last updated 7 October 2025) states that "under the pre-SAIO data collection, up to the 2024 reference year, the area concept was area under cultivation, i.e. the area actually harvested, with non-harvested areas (for example due to natural disasters) excluded", which is why `AR_THS_HA` maps to `indicator_used = "area_harvested"`. From the 2025 reference year the SAIO regulation changes that: "for cereals for the production of grain, dry pulses and protein crops, root crops, industrial crops, and plants harvested green, the areas refer to the sown area", while vegetables stay on harvested area and permanent crops on production area.

Because the new concept differs by crop group, and this reader has no crop vocabulary, it does **not** relabel 2025+ rows. It flags them: every `"apro_cpshr"` row with `year >= 2025` gets `concept_break = TRUE`, and choosing the right `indicator_used` for them is the resolver's job once a crop-group vocabulary exists. As of 2026-09-02 the dataflow held no year beyond 2024, so the flag is prospective.

NUTS versions, and why rows are deduplicated

Eurostat labels every geography whose code has been retired with the last nomenclature in which it was valid – `FR21` arrives as `"Champagne-Ardenne (NUTS 2013)"`. `nuts_version` is read from that suffix, and a code without one is on the current nomenclature, NUTS 2024 (regulation 2023/674, in force 2024-2026; see https://ec.europa.eu/eurostat/web/nuts/history). The suffix is Eurostat's own retirement marker, so `nuts_version` means

"the last nomenclature this code was valid in", which is exactly what a later code-system-scoped resolution step needs.

One response therefore mixes vintages: `apro_cpnhr_h` carries `FR21` for 1989-1999 and `FRF2` – a pure recoding of the same polygon – for 1990-1999. Rows for the same territory in the same year are deduplicated keeping the newest vintage, and the dropped count, the code pairs and any value disagreement are reported. Two rows count as the same territory only when they share a country, a NUTS level and a label once the version suffix is stripped; that comparison finds recodings only, never resolves a code to a polity, and never merges across levels, so `FI2`/`FI20` (both labelled "Aland") and `DE3`/`DE30` (both "Berlin") stay separate. A genuine boundary change between vintages keeps both rows, visible, for the resolution step to settle.

Grain, and the levels this returns

NUTS 2 is `grain = "admin1"` and NUTS 3 is `"admin2"`, the mapping the admin-shares grain rule uses for EU countries. NUTS 1 rows are returned only when asked for (`nuts_level = 1L`) and carry `grain = NA`: in some countries NUTS 1 is a genuine first-order division (Germany's Bundeslaender) and in others it is an aggregate of the units below it, so which of the two it is cannot be decided per row here. Germany is the case that forces the question – it reports crop area at NUTS 2 only up to 2004 and at NUTS 1 from 2005 onward (verified against `apro_cpshr` on 2026-09-02).

NUTS 0 rows are not units at all; they are the national totals the units should sum to, and they are returned separately as the `"national_check"` attribute of the result. Supra-national aggregates (`EU`, `EU27_2020`) and composite codes that would double-count their own members (`EL41_42`, served for 2007-2014 alongside `EL41` and `EL42`) are dropped and counted.

Examples

```
read_admin_stats_eurostat(example = TRUE)
```

`read_admin_stats_nass`

Read United States state or county agricultural statistics from NASS

Description

Reads one USDA NASS Quick Stats bulk dump from disk and returns the requested series as source-native admin-statistics rows: one row per reporting unit, item, indicator and year, with values converted to WHEP units and every non-numeric NASS value code preserved.

The dump is streamed a chunk of lines at a time and filtered chunk by chunk, so the ~8 GB flat file is never materialised. Only the fifteen columns this reader uses are parsed, and a gzipped dump is read without being extracted first.

Identifiers stay exactly as NASS published them: no item is mapped to a WHEP `item_prod_code` and no unit is resolved to a polity. Both happen downstream, from the source-native keys returned here.

Usage

```
read_admin_stats_nass(
  domain = c("crops", "animals"),
  nass_dir = NULL,
  short_desc = NULL,
  reference_period = NULL,
  agg_level = c("STATE", "COUNTY"),
  example = FALSE
)
```

Arguments

domain	Which NASS dump to read, "crops" or "animals".
nass_dir	Directory holding the Quick Stats dumps, as inst/scripts/download/download_nass.R fills it. Defaults to Sys.getenv("WHEP_NASS_DIR").
short_desc	Character vector of NASS SHORT_DESC series to keep. NULL uses the documented default for domain.
reference_period	Character vector of REFERENCE_PERIOD_DESC values to keep. NULL uses "YEAR" for crops and "FIRST OF JAN" for animals.
agg_level	Aggregation level to return, "STATE" (grain "admin1") or "COUNTY" (grain "admin2"). County rows are for within-unit validation, which the state constraint leaves untouched.
example	If TRUE, return a small fixture instead of reading a dump. Defaults to FALSE.

Value

A tibble with one row per unit, item, indicator and year:

- source: always "USDA_NASS".
- source_native_unit_id, source_native_unit_name: the state FIPS code and STATE_NAME verbatim at "STATE" grain; the five-digit state plus county FIPS code and "<state>, <county>" at "COUNTY" grain. FIPS "98" is NASS's own residual unit, "OTHER STATES", an observed value covering the states too few to publish separately.
- source_native_item_code: always NA; the dumps carry no item code.
- source_native_item_name: SHORT_DESC verbatim, the string that keys the series.
- indicator_used: "area_harvested", "area_planted_or_sown", "production" or "yield" for crop rows, NA for head counts.
- quantity: "area", "production", "yield" or "heads".
- year: calendar year.
- value: the published value in WHEP units, NA where NASS published a code instead of a number.
- value_unit: "ha", "tonnes" or "heads".

- `value_flag`: `"residual"` on an OTHER STATES row, the NASS value code verbatim (`"(D)"`, `"(S)"`, `"(Z)"`, `"(NA)"`, `"(X)"`) on a row with no published number, both joined by `"; "` where both apply, and `NA` on a clean row.
- `grain`: `"admin1"` for state rows, `"admin2"` for county rows.
- `nuts_version`: always `NA`; the United States has no NUTS geography.
- `source_version`: the dump's own date stamp, `"YYYYMMDD"`, taken from its file name.
- `recorded_at`: ISO 8601 UTC time the dump was read.

Series selection

NASS keys a series on `SHORT_DESC`, which folds the commodity, its class, the production practice and the statistic into one string (`"CORN, GRAIN - ACRES HARVESTED"`). `short_desc` is therefore the item selector, and its default covers the field crops and livestock species the subnational spatialization needs. The authoritative item-to-series table is built separately; pass `short_desc` to read anything else.

Rows are kept only where `SOURCE_DESC` is `"SURVEY"` (not the quinquennial census), `DOMAIN_DESC` is `"TOTAL"` (not a breakdown by farm size, sales or NAICS class), and `PRODN_PRACTICE_DESC` is `"ALL PRODUCTION PRACTICES"` (not the irrigated or organic split).

Reference periods

Crop area is annual (`FREQ_DESC == "ANNUAL"`) and its final estimate carries `REFERENCE_PERIOD_DESC == "YEAR"`; the in-season forecasts (`"YEAR - AUG FORECAST"` and siblings) are excluded by that default. Livestock inventories are `FREQ_DESC == "POINT IN TIME"` and the default reference period is `"FIRST OF JAN"`.

One species does not follow that default: NASS publishes the annual hog inventory as of **1 December**, so `"HOGS - INVENTORY"` has no `"FIRST OF JAN"` rows at all and returns nothing under the default. That is deliberately not patched here – which hog reference period stands for the annual inventory is a vocabulary decision, not a reading one – but it is never silent: any requested `short_desc` that matches no row warns by name, and `reference_period` selects a different one.

Source

USDA National Agricultural Statistics Service, Quick Stats bulk downloads (https://www.nass.usda.gov/datasets/), files `qs.crops_YYYYMMDD.txt.gz` and `qs.animals_products_YYYYMMDD.txt.gz`. Licence CC0 1.0 Universal. Verified 2026-09-02.

Examples

```
read_admin_stats_nass(example = TRUE)
```

`read_admin_stats_sidra` *Read Brazilian state agricultural statistics from IBGE SIDRA*

Description

Read one IBGE SIDRA table of Brazilian state-level (`admin1`) agricultural statistics through the unauthenticated SIDRA values API and return it in the shared admin-statistics reader shape: one row per source-native unit, item, indicator and year, in WHEP units.

Three tables are served, all annual from 1974:

- `"5457"` – Producao Agricola Municipal (PAM) crop areas and production. Harvested area (`indicator_used == "area_harvested"`) is the indicator the allocation binds on; planted-or-sown area (`"area_planted_or_sown"`) is a flagged proxy SIDRA publishes only from 1988, and production (`"production"`) is the last-resort indicator.
- `"3939"` – Pesquisa da Pecuaria Municipal (PPM) herd inventories, in head. Livestock rows carry `quantity == "heads"` and `indicator_used == NA`, head counts not being one of the area/production/yield indicators `admin_shares_schema()` closes over.
- `"94"` – milked cows, in head, the dairy split PPM table 3939 does not report.

Rows are **source-native**: unit and item identifiers are returned exactly as SIDRA served them and are never mapped to `item_prod_code` or resolved to a polity here. Values are returned in WHEP units with a conversion factor of 1 throughout, SIDRA serving hectares, tonnes and head directly.

Requests are paged by year block so no single query exceeds the API's 50,000-value cap; see `years_per_request`.

Usage

```
read_admin_stats_sidra(
  table = c("5457", "3939", "94"),
  years = NULL,
  items = NULL,
  years_per_request = NULL,
  example = FALSE
)
```

Arguments

`table`	SIDRA table to read, one of `"5457"` (PAM crops), `"3939"` (PPM herds) or `"94"` (milked cows).
`years`	Integer vector of calendar years. `NULL` (the default) requests 1974 – the first year all three tables publish – through the current year; years the table has not published are absent from the result rather than an error.

`items` Optional character vector of classification category codes to restrict the request to (`c782` codes for table `"5457"`, `c79` codes for `"3939"`). `NULL` requests every category and drops the `"0"` ("Total") category, which is the all-crops total rather than an item. Must be `NULL` for table `"94"`, which has no classification.

`years_per_request` Number of years per API request. `NULL` uses the per-table default that keeps a request under the 50,000-value cap.

`example` If `TRUE`, return a small fixture instead of reading from the API. Defaults to `FALSE`.

Value

A tibble with one row per unit, item, indicator and year:

- `source`: `"IBGE_PAM"` for table 5457, `"IBGE_PPM"` for 3939 and 94.
- `source_native_unit_id`, `source_native_unit_name`: the IBGE UF code and name exactly as served.
- `source_native_item_code`, `source_native_item_name`: the classification category exactly as served; the code is `NA` for table 94, whose item is the table itself and whose name is the served variable name.
- `indicator_used`: `"area_harvested"`, `"area_planted_or_sown"` or `"production"` for crop rows, `NA` for head counts.
- `quantity`: `"area"`, `"production"` or `"heads"`.
- `year`: calendar year.
- `value`: the served value in WHEP units, `NA` where the source gave a missing-value code.
- `value_unit`: `"ha"`, `"tonnes"` or `"heads"`.
- `value_flag`: the source's missing-value code verbatim (`"..."`, `".."` or `"X"`), `NA` when the row carries a value. An absolute zero (`"-"`) is a value, not a flag.
- `grain`: always `"admin1"`; SIDRA level N3 is the state.
- `nuts_version`: always `NA`, Brazil having no NUTS geography.
- `source_version`: always `NA`; SIDRA exposes no vintage stamp.
- `recorded_at`: ISO 8601 UTC fetch time of the request the row came from.

Source

IBGE SIDRA API (https://apisidra.ibge.gov.br), tables 5457 (Producao Agricola Municipal), 3939 and 94 (Pesquisa da Pecuaria Municipal). Unauthenticated; verified 2026-09-02.

Examples

```
read_admin_stats_sidra(example = TRUE)
```

`read_critical_n` *Read a Schulte-Uebbing gridded critical-nitrogen layer.*

Description

Reads one 0.5-degree gridded critical-nitrogen layer from the Schulte-Uebbing et al. (2022) archive (doi:10.5281/zenodo.6395016) onto WHEP's grid: the critical nitrogen surplus, the critical nitrogen input, the exceedance of the critical surplus, the three medium-specific critical losses (ammonia emission, groundwater leaching, surface-water load), or the threshold-exceedance map. Values are in kg N per hectare per year (a categorical 1-8 impact code for `threshold_exceedance`). The critical surplus, input and exceedance are selectable by `threshold` (minimum of all media, surface water, groundwater or deposition) and `land_use` (all agricultural land, arable only, or intensively managed grassland); the three critical losses and the threshold-exceedance map ignore `threshold`.

The threshold-exceedance map (the archive's "Threshold exceedance by impact" folder, `threshold_exc_<land_use>.asc`) records which thresholds are exceeded in a cell, not which one binds. The threshold that binds, the one with the lowest critical surplus, is derived by `build_critical_n_binding()`. The layer was formerly requested as `var = "binding_threshold"`; that name still reads it, with a deprecation warning, and the result is stamped `critical_var = "threshold_exceedance"`.

The archive directory comes from `dir`, else the `WHEP_CRITICAL_N_DIR` environment variable, else a local cache that is populated by downloading the archive from Zenodo on first use (see `dir`).

`method = "reproduced"` returns the same layer recomputed from the archive's own 2010 inputs with the source's equations by `calculate_critical_n()`, instead of the deposited raster. It covers the critical surplus, the critical input and the surplus exceedance, matches the deposited layers closely but not exactly (see `calculate_critical_n()`), and is stamped in `method_critical_n` and `critical_source` so it is never read as the deposited surface.

Usage

```
read_critical_n(
  var = c("critical_n_surplus", "critical_n_input", "exceedance", "crit_nh3_emission",
    "crit_leaching_gw", "crit_load_sw", "threshold_exceedance", "binding_threshold"),
  threshold = c("mi", "sw", "gw", "de"),
  land_use = c("all", "ara", "igl"),
  dir = NULL,
  data = NULL,
  example = FALSE,
  verify_source = TRUE,
  method = c("archive", "reproduced")
)
```

Arguments

`var`	Which critical-nitrogen layer to read: one of `"critical_n_surplus"`, `"critical_n_input"`, `"exceedance"`, `"crit_nh3_emission"`, `"crit_leaching_gw"`,

	"crit_load_sw" or "threshold_exceedance". "binding_threshold" is a deprecated alias of "threshold_exceedance".
threshold	Impact threshold selecting the critical value: "mi" (minimum across media, the collapsed boundary), "sw" (surface-water eutrophication), "gw" (groundwater nitrate) or "de" (atmospheric or terrestrial deposition). Ignored by the critical-loss and threshold-exceedance layers.
land_use	Land-use scope: "all" (arable plus intensively managed grassland), "ara" (arable only) or "igl" (intensively managed grassland). Ignored by the critical-loss layers (crit_nh3_emission, crit_leaching_gw, crit_load_sw), which have a single land-use-agnostic file; used by the threshold-exceedance map and the surplus/input/exceedance layers.
dir	Optional path to the archive directory, overriding WHEP_CRITICAL_N_DIR. Defaults to NULL, in which case the archive is resolved as: dir, then WHEP_CRITICAL_N_DIR, then a local cache under rappdirs::user_cache_dir("whep"). When the cache is empty the 18.4 MB CC-BY-4.0 Zenodo archive is downloaded, verified against its published MD5 and unpacked there on first use, so a plain read_critical_n() call works with nothing configured. Unpacking needs a 7-Zip extractor: the archive package (system libarchive) or a 7z binary on PATH. With neither, the call aborts after the download, naming the command to run.
data	Optional pre-read tibble (lon, lat, value) returned directly instead of reading the archive, for tests and injection. Defaults to NULL.
example	If TRUE, return a small fixture instead of reading data. Defaults to FALSE.
verify_source	If TRUE (default), real archive reads verify the selected critical raster and its source-area/IMAGE support rasters against the package's versioned content manifest before parsing. Ignored for data and example injection.
method	Where the layer comes from: "archive" (default) reads the deposited raster; "reproduced" recomputes it with calculate_critical_n() from the archive's 2010 inputs. "reproduced" supports var = "critical_n_surplus", "critical_n_input" and "exceedance" and aborts for the other layers.

Value

A tibble with lon, lat (0.5-degree cell centres), value (kg N per hectare per year; a categorical impact code for threshold_exceedance) and retained layer provenance: critical_var, critical_threshold, critical_land_use, critical_year and critical_source, canonical integer cell_id/row/column keys, deposited source_area_ha, IMAGE-region membership, DOI/version, archive checksum and method_critical_n ("archive" or "reproduced"). NODATA cells are dropped.

Examples

```
read_critical_n(example = TRUE)
```

`read_cru_climate` *Read a CRU TS 4.09 monthly climate variable into a tidy tibble.*

Description

Reads one monthly variable of the CRU TS 4.09 high-resolution gridded climate dataset (1901-2024, global 0.5 degree) from its NetCDF file and returns it in tidy long form, one row per land cell-month. Requested years are sliced at read time so the full grid is never materialised, and ocean fill cells are dropped. Values are returned in the file's native units (not converted): `tmp`/`tmn`/`tmx`/`dtr` in degrees Celsius, `pre` in mm/month, `pet` in mm/day, `vap` in hPa, `cld` in percent, `wet` and `frs` in days.

Usage

```
read_cru_climate(
  var = c("tmp", "pet", "pre", "tmn", "tmx", "vap", "cld", "wet", "dtr", "frs"),
  years = NULL,
  cru_dir = NULL,
  data = NULL,
  example = FALSE
)
```

Arguments

`var`	Variable name, one of `"tmp"`, `"pet"`, `"pre"`, `"tmn"`, `"tmx"`, `"vap"`, `"cld"`, `"wet"`, `"dtr"` or `"frs"`.
`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year present in the file.
`cru_dir`	Path to the CRU TS NetCDF directory. Defaults to `Sys.getenv("WHEP_CRU_DIR")`; aborts when neither is set.
`data`	Optional pre-read tibble (`lon`, `lat`, `year`, `month`, `value`) used in place of reading NetCDF, for testing.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with columns `lon`, `lat`, `year`, `month`, `value` (native units) and `var` (the requested variable name).

Source

CRU TS 4.09 (Climatic Research Unit, University of East Anglia; Harris, Osborn & Jones 2020, Scientific Data, doi:10.1038/s4159702004533).

Examples

```
read_cru_climate(example = TRUE)
```

`read_emep_deposition` *Read EMEP MSC-W atmospheric nitrogen deposition onto WHEP's grid.*

Description

Reads the yearly EMEP MSC-W model results (EMEP01 rv5.6, 2025 reporting round) for one nitrogen species and aggregates the 0.1-degree grid to WHEP's 0.5-degree grid. NHx is `DDEP_RDN_m2Grid + WDEP_RDN` and NOy is `DDEP_OXN_m2Grid + WDEP_OXN`, dry plus wet. The source is a density (mgN/m2 over the whole grid cell), so it is converted to kg N/ha (x 0.01) and aggregated by plain mean: the 0.1-degree grid nests exactly 5x5 inside each 0.5-degree block and the cell area varies by under 0.2 per mille inside a block.

EMEP is the reference `correct_n_deposition()` corrects HaNi's European trend toward. It is a regional product (domain lon -30 to 90, lat 30 to 82) and is not a substitute for HaNi anywhere else.

The files are third-party model output and are read from local disk: `inst/scripts/download/download_emep.R` fetches them from the Norwegian Meteorological Institute THREDDS server into `<dest_dir>/EMEP/`, and `WHEP_EMEP_DIR` points there.

Simpson, D. *et al.* (2012). The EMEP MSC-W chemical transport model – technical description. *Atmospheric Chemistry and Physics* 12(16), 7825-7865. doi:10.5194/acp12-78252012

Usage

```
read_emep_deposition(
  species = c("nhx", "noy"),
  emep_dir = NULL,
  years = NULL,
  example = FALSE
)
```

Arguments

`species`	Which species to read, `"nhx"` or `"noy"`.
`emep_dir`	Path to the directory holding the yearly EMEP files. Defaults to `Sys.getenv("WHEP_EMEP_DIR")`
`years`	Optional integer vector of calendar years to read. `NULL` reads every year with a file in `emep_dir`. A requested year with no file is not read; the years actually returned are the ones in the output.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `year` and `deposition_kgn_ha` (the species' grid-mean deposition rate over the whole 0.5-degree cell).

Examples

```
read_emep_deposition(example = TRUE)
```

`read_fbs_population` *Read FAOSTAT Food Balance Sheet population on WHEP area codes.*

Description

Reads item 2501 "Population", element 511 "Total Population - Both sexes", from the `faostat-fbs-old` and `faostat-fbs-new` pins — the same two pins `build_commodity_balances()` reads for the food itself — and returns population per area and year on the numeric `area_code` the rest of the package uses. FAOSTAT publishes it in thousands; it is converted to persons here.

This is the one population source WHEP has that is **not** keyed on a present-day ISO3 code. It is keyed on the FAOSTAT area code, the same key space the commodity balances are built from, and FAOSTAT keeps a dissolved reporting area alive for the years it reported. So it covers the territories `read_population()` and `read_wpp_population()` structurally cannot: area 186 Serbia and Montenegro for exactly 1992–2005 and area 151 Netherlands Antilles for exactly 1961–2010, the two largest holes `population_source_reach()` reports (#862, #787).

The two pins overlap over 2010–2013 and disagree there, sometimes sharply: FAOSTAT area 272 Serbia in 2010 is 9,647,000 in `faostat-fbs-old` against 7,395,860 in `faostat-fbs-new`, because the old vintage's Serbia includes Kosovo and the new one does not. `faostat-fbs-new` wins an overlapping `(year, area_code)`, which is the order `build_commodity_balances()` gives the same two pins for the numerator.

FAOSTAT's regional and grouping aggregates (`World`, `Africa`, `European Union`, `Least Developed Countries` and 38 others) carry area codes at or above 5000, resolve to no polity, and are dropped, so they cannot leak into a per-country denominator. An area is kept only if it resolves to a **polity**, not merely to a `polity_area_code` bucket: area 351 "China" is the aggregate over areas 41, 96, 128 and 214, is numbered below 5000, and does land on a bucket of its own, so a bucket-only filter admitted it on top of its four members — 1.46 billion persons in 2021 (#939). Every area dropped for having a bucket and no polity is named in a message.

`area_code` is `polity_area_code`, a **bucket, not an identity**, resolved year by year exactly as the commodity balances resolve it. On the real pins no bucket-year receives more than one FAOSTAT area, so no row here sums two territories — but note South Sudan (277) has no population row in either pin, so bucket 206 from 2012 is Sudan alone, where the `gdp-population` pin sums `SDN + SSD` onto it.

Usage

```
read_fbs_population(years = NULL, data = list(), example = FALSE)
```

Arguments

`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the pins cover.
`data`	Optional named list of pre-loaded raw pins to avoid the pin read, `fbs_old` and/or `fbs_new`, each in the pins' own long FAOSTAT layout (`Area Code`, `Item Code`, `Element Code`, `Year`, `Value`). Falls back to `whep_read_file()` for whichever is absent.
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with `year`, `area_code`, `population` (persons) and `source_pop` (`"FAOSTAT FBS old"` or `"FAOSTAT FBS new"`, naming which pin the row came from), one row per area code and year, sorted by year then area code, plus the polity columns below.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status`

= "flag") (or "status") carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
read_fbs_population(example = TRUE)
```

---

read_glaciated_areas *Read the Natural Earth glaciated-areas ice layer*

Description

Reads ne_10m_glaciated_areas, the ice source for build_polycell_support(). A few features are invalid under the spherical s2 engine and GEOS-level sf::st_make_valid() does not repair them, so sf::st_area() aborts with "Loop 0 is not valid". Those features are repaired under the planar engine, and any that remain invalid are measured with terra::expanse(), which does not go through s2, and reported rather than silently dropped.

The layer is a coarse present-day snapshot, so ice area does **not** vary historically. That is acceptable only while ice is a reporting category rather than a driver.

Usage

```
read_glaciated_areas(dir = NULL)
```

Arguments

dir	Directory holding the shapefile. Defaults to Sys.getenv("WHEP_NATURALEARTH_DIR"), under which the layer is expected at ne_10m_glaciated_areas/ne_10m_glaciated_areas.shp.

Value

An sf table of glaciated polygons in WGS84, carrying a s2_repaired logical column. The "unrepaired" attribute is a tibble of the features that stayed s2-invalid, with their terra::expanse() area.

Examples

```
# Requires WHEP_NATURALEARTH_DIR to be set; not run without it.
if (nzchar(Sys.getenv("WHEP_NATURALEARTH_DIR"))) {
  read_glaciated_areas()
}
```

`read_glw_density` *Read GLW3 gridded livestock counts on WHEP's 0.5-degree grid*

Description

Read the Gridded Livestock of the World version 3 rasters (GLW 3, Gilbert et al. 2018) and return them as the `glw_density` table `build_gridded_livestock()` allocates on under `proxy_method = "glw3"`: one row per 0.5-degree cell and WHEP `species_group`, carrying that cell's GLW3 head count.

GLW3 is published at 5 arc-minutes (1/12 degree) in absolute animals per pixel, so the 36 pixels tiling a 0.5-degree cell are summed and nothing is rescaled; see *How five-arcmin pixels become 0.5-degree cells.* The eight GLW species are then mapped onto WHEP's eleven species groups by `crosswalk`; see *The GLW3 crosswalk and what it cannot split.*

The rasters come from `glw_dir`, else the `WHEP_GLW3_DIR` environment variable, which points at the directory `inst/scripts/download/download_glw3.R` fills from the published Harvard Dataverse DOIs. An unset variable aborts naming that script; there is no fallback to another layer.

Usage

```
read_glw_density(
  species = NULL,
  variant = c("DA", "AW"),
  glw_dir = NULL,
  crosswalk = NULL,
  example = FALSE
)
```

Arguments

`species`	Which GLW species to read, as named in `crosswalk`'s `glw_species` column (`"cattle"`, `"buffaloes"`, `"sheep"`, `"goats"`, `"pigs"`, `"chickens"`, `"ducks"`, `"horses"`). `NULL` (default) reads all of them. An unknown name aborts listing the known ones rather than returning a table missing a group.
`variant`	Which GLW3 product to read: `"DA"` (default, the dasymetric product, `5_<Sp>_2010_Da.tif`) or `"AW"` (the areal-weighted product, `6_<Sp>_2010_Aw.tif`), validated with `rlang::arg_match()`. The two are alternatives, never fallbacks: the dasymetric product redistributes census counts with high-resolution covariates, the areal-weighted one spreads them evenly over the reporting unit's suitable land.
`glw_dir`	Directory holding the GLW3 GeoTIFFs, overriding `WHEP_GLW3_DIR`. A `GLW3` subdirectory of it is also searched, since that is where `download_glw3(dest_dir)` puts them. Defaults to `NULL`.

`crosswalk` Optional tibble mapping GLW species onto WHEP species groups, replacing the packaged default. Required columns `glw_species`, `glw_code`, `species_group`; see *The GLW3 crosswalk and what it cannot split*. Defaults to `NULL`.

`example` If `TRUE`, return a small fixture instead of reading any raster. Defaults to `FALSE`.

Value

A tibble with one row per cell and species group:

- `lon`, `lat`: 0.5-degree cell centre coordinates.
- `species_group`: WHEP livestock functional type.
- `density`: GLW3 head count in that cell, the quantity `build_gridded_livestock()` documents as "heads per cell".
- `glw_variant`: the `variant` these rows were read from, `"DA"` or `"AW"`. It travels with the values so that `build_gridded_livestock()` can record the product a run allocated on in `method_livestock_proxy` without being told separately, which is the only way the label cannot drift from the geography. Cells with no positive count are dropped, so the table is sparse.

How five-arcmin pixels become 0.5-degree cells

GLW3 pixel values are absolute animal counts per pixel, not densities (Dataverse file description: "absolute number of animals per pixel; 4320 by 2160 pixels of 0.083333 decimal degrees resolution"). The aggregation is therefore a plain block sum: each 0.5-degree cell takes the sum of the 6 x 6 = 36 five-arcmin pixels it contains, missing pixels skipped, and a cell whose pixels are all missing is dropped. No division by area happens anywhere, because none is needed to reach the engine's per-cell head count.

The block factor is derived from the raster's own resolution rather than hardcoded, and a resolution that does not divide 0.5 degrees a whole number of times aborts: a raster on another grid would otherwise be silently resampled onto shifted cells.

Resolution alone does not place a raster, so the extent is checked too. Every edge must fall on a WHEP cell boundary: a raster of the right resolution offset by, say, 0.1 degrees aggregates onto centres that are not WHEP's, and an extent that is not a whole number of cells leaves a partial edge block whose truncated sum would travel as a full cell's head count. Neither is visible downstream – `build_gridded_livestock()`'s join on `lon`/`lat` would simply drop the species – so both abort here, naming the offending edge and its offset.

The GLW3 crosswalk and what it cannot split

`inst/extdata/glw3_species_group.csv` maps the eight GLW species onto WHEP's `species_group` vocabulary (`inst/extdata/livestock_mapping.csv`). Columns:

- `glw_species`: GLW species name, and the value `species` selects on.
- `glw_code`: the two-letter code in the published file names (`Ct`, `Bf`, `Sh`, `Gt`, `Pg`, `Ch`, `Ho`, `Dk`).

- `species_group`: the WHEP group the layer feeds.
- `note`: why that row exists, for the reader of the file.

Two rules govern a many-to-one or one-to-many row set, and both are applied by this reader:

- **Several GLW species to one group are summed**: `sheep` + `goats` into `sheep_goats`.
- **One GLW species to several groups gives each group the same value**: `cattle` feeds both `cattle_dairy` and `cattle_non_dairy`, `chickens` feeds `chickens_layers` and `chickens_broilers`. GLW3 carries no dairy and no layer/broiler split, so the finer groups inherit one geography and their national totals still differ. This is the interim "coarse layer constrains the sum of the finer groups" rule of whep#1000; task T10 may replace it with a split, and the replacement is a new crosswalk plus a rule here, not a change of contract.

`poultry` takes the `ducks` layer alone. WHEP's `poultry` group is ducks + geese/guinea fowls + turkeys (`inst/extdata/livestock_mapping.csv`, FAOSTAT items 1068, 1072 and 1079); chickens are not in it, they have their own two groups. The crosswalk used to feed `chickens` into `poultry` as well, and since chickens outnumber ducks by an order of magnitude almost everywhere, the summation replaced the duck geography with a chicken-dominated one (whep#1000, task T15a-iii).

This is the duck-only half of the decided rule, and it is interim. The decided rule is a within-group split by national head shares, `density(poultry) = duck * w_duck + chicken * (1 - w_duck)` with `w_duck` the national ducks share of ducks + geese/guinea fowls + turkeys per area and year, falling back to the duck layer alone where a country reports no poultry heads. That split cannot be applied here: it is per country and per year, while this table is neither, and the item-grain national head counts it needs are not among the inputs `run_spatialize()` loads (`livestock_country_data.parquet` is already summed to `species_group`, so ducks, geese and turkeys arrive as one number). Until that input exists, `poultry` carries the geography of the one group member GLW3 publishes, which is the decided rule's own no-heads treatment, rather than the geography of a species that is not in the group at all.

`camels` and `other` have no GLW species and are absent from the result by construction. That is deliberate: `build_gridded_livestock(proxy_method = "glw3")` aborts naming any group its density table does not cover, so those groups have to be run under `proxy_method = "luh2"` rather than being given a wrong geography here.

Source

Gilbert, M., Nicolas, G., Cinardi, G., Vanwambeke, S., Van Boeckel, T. P., Wint, G. R. W. and Robinson, T. P. (2018). Global distribution data for cattle, buffaloes, horses, sheep, goats, pigs, chickens and ducks in 2010. *Scientific Data*, 5, 180227. doi:10.1038/sdata.2018.227. Rasters: Harvard Dataverse `glw_3`, version 3.0, CC0 1.0 Universal.

Examples

```
read_glw_density(example = TRUE)
```

`read_glwd_water` *Read the GLWD inland-water fraction on the 0.5-degree grid*

Description

Returns the Global Lakes and Wetlands Database surface-water fraction of each 0.5-degree cell. `water_frac` is a fraction of the **whole** cell, so the water area of a cell is `water_frac * cell_area_ha`.

This is the inland-water source for `build_polycell_support()`. It is preferred over `ne_10m_lakes`, which carries roughly half of global inland water and omits the Caspian Sea entirely.

The fraction is derived from the GLWD rasters `inst/scripts/download/download_hydrology.R` fetches from the published figshare DOI, through `glwd_water_fraction()` – the same derivation `inst/scripts/prepare_spatialize_all.R` uses to write LPJmL's own `lakes_rivers` input, so the two cannot drift.

Until WHEP settled on GLWD v2 this read LPJmL's `glwd_lakes_and_rivers_30arcmin.clm` and a companion `grid.clm` instead. That pair is derived from GLWD **v1**, no script in this repository produces it, and it gives 2.4759 Mkm2 of inland water over the 67,420-cell CRU land mask against v2's 3.2480 Mkm2. Any figure quoted against the old layer has to be re-measured rather than carried across.

Usage

```
read_glwd_water(dir = NULL)
```

Arguments

`dir`	Directory holding `GLWD/`, as `download_hydrology.R` lays it out. Defaults to `Sys.getenv("WHEP_LPJML_INPUT_DIR")`.

Value

A `tibble` with `lon`, `lat` and `water_frac`, one row per cell the rasters cover.

Source

Lehner, B., Anand, M., Fluet-Chouinard, E. et al. (2025). Mapping the world's inland surface waters: an update to the Global Lakes and Wetlands Database (GLWD v2). *Earth System Science Data* 17, 2277-2329. doi:10.5194/essd1722772025

Examples

```
# Requires WHEP_LPJML_INPUT_DIR to be set; not run without it.
if (nzchar(Sys.getenv("WHEP_LPJML_INPUT_DIR"))) {
  read_glwd_water()
}
```

`read_habitual_cv` *Read FAOSTAT's coefficient of variation of habitual caloric consumption.*

Description

Reads item 21058 from the FAOSTAT Suite of Food Security Indicators bulk download, resolved to WHEP area codes. The file is located in order: the `dir` argument, then `WHEP_FAOSTAT_FS_DIR`, then a cache under `rappdirs::user_cache_dir("whep")`, downloading on first use.

As with the UN WPP reader, the size and MD5 recorded here are WHEP's own: FAOSTAT publishes no checksum for its bulk files, so a mismatch means the upstream release changed, not necessarily that the download failed.

Usage

```
read_habitual_cv(years = NULL, data = NULL, dir = NULL)
```

Arguments

`years`	Optional integer vector of years to keep.
`data`	Optional pre-read table, bypassing the file. Used by the tests.
`dir`	Optional directory holding the bulk zip.

Value

A tibble with `year`, `area_code`, `cv`.

Examples

```
read_habitual_cv(
  data = tibble::tribble(
    ~`Area Code`, ~`Item Code`, ~Year, ~Value,
    2L,           21058L,       2010L, 0.25
  )
)
```

`read_hwsd_topsoil_soc` *Read observed topsoil organic carbon from HWSD onto WHEP's grid.*

Description

Read the observed **0-30 cm** soil organic carbon stock per 0.5-degree cell from the HWSD (Harmonized World Soil Database) map-unit attribute table and raster. Each map unit's stock is the share-weighted mean over its soil components of `t_oc * bulk_density * 30 * (1 - t_gravel / 100)`, and the map units are then averaged over the native HWSD cells inside each 0.5-degree block by the same aggregation `read_soil_ph()` uses.

This exists to be a **benchmark**, not a model input: nothing in the carbon pipeline consumes it. HWSD version 1.2's topsoil is 0-30 cm, exactly the layer `build_carbon_balance()` reports (see its Soil depth section), so it is the only observational anchor available at WHEP's own modelled depth – and it comes out of the archive the carbon balance already reads for clay, so it needs no new download. Note that HWSD**2**, which `inst/scripts/download/download_hwsd.R` fetches, layers its topsoil as D1 = 0-20 cm instead; the two are not interchangeable (whep#851).

Usage

```
read_hwsd_topsoil_soc(
  hwsd_dir = NULL,
  bulk_density = c("measured", "reference"),
  data = list(),
  example = FALSE
)
```

Arguments

`hwsd_dir`	Path to the directory holding `hwsd_data.csv` and `hwsd.bil`. Defaults to `Sys.getenv("WHEP_HWSD_DIR")`.
`bulk_density`	Which HWSD bulk density to use. `"measured"` (default) takes `t_bulk_density`, falling back to `t_ref_bulk_density` where it is absent; `"reference"` takes `t_ref_bulk_density` alone. The default is not cosmetic: `t_ref_bulk_density` is derived from texture and so knows nothing about organic matter, and over the 1,375 map units with `t_oc` above 6% it averages 1.33 against a measured 0.37, which would inflate a peat soil's carbon stock roughly 3.6-fold. Recorded in `method_soc_obs`.
`data`	Optional named list of pre-loaded inputs: `cell_polity` (`lon`, `lat`, at minimum), used both to crop the HWSD raster to the region of interest before reclassification and as the target grid for gap-filling. When absent, cropping and gap-filling are both skipped (documented fallback above).
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `soc_obs_mgc_ha` (0-30 cm soil organic carbon, Mg C per ha) and `method_soc_obs`.

Source

FAO/IIASA/ISRIC/ISSCAS/JRC (2012). *Harmonized World Soil Database version 1.2*. FAO, Rome and IIASA, Laxenburg – topsoil defined as 0-30 cm. Stock equation and

the bulk-density caveat: Hiederer, R. & Koechy, M. (2011). *Global Soil Organic Carbon Estimates and the Harmonized World Soil Database.* EUR 25225 EN, Publications Office of the European Union, 79 pp.

Examples

```
read_hwsd_topsoil_soc(example = TRUE)
```

`read_hyde_population` *Read gridded HYDE population onto WHEP's grid.*

Description

Reads a HYDE baseline-scenario population count (native 5-arcmin ESRI ASCII grid, people per native cell) for one or more calendar years and, by default, aggregates it to WHEP's 0.5-degree grid by summing the 6x6 fine cells inside each 0.5-degree block, since population count is an extensive quantity. Each requested year is read from its own `"{year}AD_pop.zip"` archive, which holds the total (`popc`), urban (`urbc`) and rural (`rurc`) counts on the same grid.

Usage

```
read_hyde_population(
  hyde_dir = NULL,
  years = NULL,
  variable = c("total", "urban", "rural"),
  aggregate = TRUE,
  example = FALSE
)
```

Arguments

`hyde_dir`	Path to the directory holding the HYDE `"{year}AD_pop.zip"` archives. Defaults to `Sys.getenv("WHEP_HYDE_DIR")`.
`years`	Integer vector of calendar years to read (`AD`, so `>= 1`). Required: each year is a real unzip-and-parse of a ~150MB archive, so there is no default range.
`variable`	Which population count to read: `"total"` (default, HYDE `popc`), `"urban"` (`urbc`) or `"rural"` (`rurc`). The output column is named after it.
`aggregate`	If `TRUE` (default), sum the fine cells to WHEP's 0.5-degree grid. If `FALSE`, return the native 5-arcmin cells, keyed by their own centres, for a consumer whose target grid is not 0.5 degrees.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `year` and one count column named after `variable`: `total_pop`, `urban_pop` or `rural_pop` (people in the cell that year). Cells HYDE marks as no-data are absent, never zero.

Examples

```
read_hyde_population(example = TRUE)
read_hyde_population(variable = "urban", example = TRUE)
```

---

read_level_country_grid

Read the cell-to-polity country grid at a containment depth

Description

The spatialization engines allocate a national total into the compartments of a *country grid*: one row per physical 0.5-degree cell and territorial unit, carrying that unit's share of the cell. `level = 0L` is today's grid, one row per cell and reporting `area_code`. `level >= 1L` resolves the polycell support one containment step deeper, so a cell held by a province arrives as a province-keyed row instead of being folded into its country.

Under `grid_vintage = "snapshot_2015"` level 0 is served by the unchanged `.read_polycell_country_grid()` path, so such a run is bit-for-bit what it was. Deeper levels bypass that path entirely – in particular they never reach `.carbon_support_to_area_code()`, whose fold is what deletes a province.

Usage

```
read_level_country_grid(
  level = 0L,
  support = NULL,
  containment = NULL,
  reference_year = NULL,
  grid_vintage = c("year_aware", "snapshot_2015"),
  containers = NULL,
  double_claim = c("co_presence", "measured")
)
```

Arguments

`level`	Containment depth, a non-negative whole number. `0L` (default) is today's cell-to-`area_code` grid.
`support`	Polycell support table in the `build_polycell_support()` grain, overriding `read_polycell_support()`. Not read at `level = 0L` with `grid_vintage = "snapshot_2015"`, which resolves the support itself.
`containment`	Containment edge table in the polity_containment schema, overriding the packaged one. Only read at `level >= 1L`.

`reference_year`
: Optional year to snapshot the grid at, applied with the package's own validity predicate. `NULL` (default) keeps the interval grain, which is what the engines' per-year filter expects. Not read at `level = 0L` with `grid_vintage = "snapshot_2015"`, whose reference year is fixed at `.carbon_support_year()`.

`grid_vintage`
: Which vintage of the cell-to-polity support a level-0 grid is read at, `"year_aware"` (default) or `"snapshot_2015"`. See *Which vintage of the support level 0 is read at*, which states what each one allocates into. Not read at `level >= 1L`, which is year-aware by construction; snapshot a depth with `reference_year`.

`containers`
: Optional integer vector of container reporting `area_code`s the depth read is scoped to – the containers a run grants a depth. `NULL` (default) reads every admitted edge. Only read at `level >= 1L`. See *Which containers a depth is read for*.

`double_claim`
: Which clashes the double-claim gate refuses, one of `"co_presence"` (default) or `"measured"`. See *Which double claims are refused*. Only read at `level >= 1L`.

Value

A `tibble` with `lon`, `lat`, `area_code` (integer, the container's reporting code), `level_polity_code` (character, `NA` at level 0), `level` (integer), `cell_area_frac`, `polycell_id`, `start_year`, `end_year`, `cell_area_ha` and `land_area_ha`. At level 0 with `grid_vintage = "snapshot_2015"` the six columns `.carbon_cell_support()` returns are passed through unchanged; with `"year_aware"` those six arrive alongside `start_year` and `end_year`.

Which vintage of the support level 0 is read at

The support is a table of cell-to-polity claims with validity intervals, so "which polity holds this cell" has an answer per year. Two answers are selectable and neither is ever a fallback for the other:

`"snapshot_2015"` One reference year, `.carbon_support_year()`, folded onto reporting `area_code`s – the carbon path's DA-28 / whep#549 choice that migrating the territorial EXTENT does not migrate the ATTRIBUTION. Every year of a run is allocated into the SAME present-day geography.

`"year_aware"` The support in its own interval grain, keyed and folded per epoch, so `.filter_country_grid_year()` selects the polities valid in each simulation year – the same treatment a granted depth gets. A cell held by a polity that has no reporting code in that year, or by no polity at all, is then absent for that year.

The two are different geographies, not two precisions of one, and the difference is large before about 1990: on the `20260825T102349Z-1a0eb` support the year-aware read covers 41,597 cells and 7,942 Mha of land at 1851 against the snapshot's 66,709 cells and 12,926 Mha. Of that year's LUH2 cropland, 41.6% is taken away from the country the snapshot gives it to and 21.5% is given to a country the snapshot does not – the two directions are reported separately, because a metric that scores only the losses cannot see a share that absorbs one country's land into another. 101.8 Mha of the loss falls in cells no polity claims

in 1851 and is attributed to nobody. At 2015 the two vintages agree exactly, which is the check that fails if the year-aware share is renormalised.

Those figures are the `20260825T102349Z-1a0eb` support's, not the `20260827T190201Z-f82a2` one's: that later pin books every hectare of inland water and ice as land (whep#1010), which moves the cell count and the land, though not the argument. `run_spatialize()` records the resolved value in `run_metadata.yaml` and in a `method_grid_vintage` column on every output it writes.

`"snapshot_2015"` is still the default, against whep#1000 T31(j)'s stated preference, because WHEP's national tables are on a **constant-territory** basis and the support is on a **historical-polity** one. In 1961 `country_areas` reports the Russian Federation, Kazakhstan and Ukraine where the year-aware grid offers only the USSR, so 17.2% of the world's harvested area has no cell to land in and is dropped whole; `validation/spatialize_grid_vintage.R` measures it per year. Reconciling the two bases is a lineage step on the national side, not a grid setting.

Which rows count as a level

A polity is at depth d when the containment edge (polity_containment) places it d steps inside a container that is not itself contained. An edge is admitted only when its container resolves, through polity_area_crosswalk, to a single reporting `area_code` **and** is not of `polity_type "aggregate"`. An aggregate's reporting code is a matrix *bucket*: `999` alone pools 62 territories, so filing a province under one would attribute it to Rest of World rather than to a country. Dropped edges are counted and named, never skipped in silence.

How the share of the cell is measured

`cell_area_frac` is the unit's share of the **physical cell on the land basis** – `land_area_ha / cell_land_ha`, the same basis `.carbon_attach_land_share()` uses at level 0, with the cell's whole measured land as the denominator. Everything the fraction later splits (LUH2 class areas, crop-pattern hectares) is already land-only, so dividing by the whole cell would subtract the water twice. Because the support is read in its interval grain, the denominator is evaluated per interval-start year: the cell's land partition genuinely differs between epochs, and taking one denominator across all of them would count a cell once per epoch.

Two support shapes are recognised, and which one is in hand is **detected, never assumed**:

replacement The support carries the units *instead of* their container (the shape a level-tagged pin delivers). `land_area_ha` is already the unit's absolute land, so the share is taken directly. A container row surviving beside its own members in one cell is refused: both claim the same ground, and choosing one silently is exactly the double count this epic exists to remove.

nested The support carries a within-container share column (`container_area_frac`, `container_frac` or `parent_frac`) alongside the container's own row. The unit's share of the cell is then the composition `f_unit x f_container`, with `f_container` measured from the container's own row on the same land basis.

Which double claims are refused

A support that carries a container's own row beside its units' rows in one cell claims that ground twice: `cell_land_ha` is the sum over the cell, so every unit's share is divided by

a total counting the same ground twice, and nothing downstream can see it because the container's national total still reconciles. The two rules are alternatives, never a fallback.

`"co_presence"` The default, and fail-closed: a container row found beside its own units is refused, full stop. It is the strict rule because this table carries no geometry – whether the two polygons overlap cannot be read off it.

`"measured"` Refuses only a container the cell's own area REFUTES: at least one shared cell-epoch whose whole claimed territory exceeds `cell_area_ha`. A cell cannot hold more territory than it has, so an excess is proof; but the proof is ONE-SIDED, which is why this is the weaker rule and not the default. A cell half of which is ocean can hold a duplicate under its own area and show no excess. Every clash it passes over is warned about by container and cell-epoch count.

On the `20260827T190201Z-f82a2` support the measurement separates the two cleanly: over the ten containers the shipped edge table admits at depth 1, 3,285 of 3,965 clashing cell-epochs over-claim by 529.03 Mha in total, and the whole of that sits in eight containers whose units are genuinely nested – Alaska inside the USA alone over-claims 150 Mha over 1,156 of its 1,372 shared cells. The two that over-claim nothing are Ryukyu inside Japan 1895-1945 (8 cells, 0 Mha) and Singapore inside Malaysia (1 cell, 0 Mha), whose ground in the shared cell is disjoint. The decision is taken per CONTAINER rather than per cell for that reason: a nested container's coastal cells hide their own duplicate, while its interior cells prove the nesting outright.

Which containers a depth is read for

Without `containers`, every edge polity_containment admits at this depth is read, whoever the run is for. On the shipped edge table that is ten containers with support rows – Alaska inside the USA, three 1949 Indonesian units, Manchuria inside three Chinese epochs, Serbia inside three Yugoslav ones, Singapore inside Malaysia – and each of them is carried in the polycell support BESIDE its own units, so `.level_check_no_double_claim()` refuses the read. The refusal is right: continuing would divide every unit's share by a cell land total that counts the same ground twice. It is simply not the Japanese run's business.

`containers` scopes the edge set to the containers asked for, and that is all it does. **Scoping is not suppressing**: the double-claim gate, the aggregate-container refusal and the support-empty abort all still run, on the scoped edges, so a granted container whose own row survives beside its units aborts exactly as before. A container named here that no edge places at this depth aborts with class `whep_level_container_not_admitted` rather than returning a grid that silently lacks it.

See Also

`build_allocation_layer()`, `read_polycell_support()`.

Examples

```
# Offline: a two-cell support for one Japanese prefecture, with the
# containment edge that puts it inside Japan. No pin and no network.
support <- tibble::tribble(
  ~polycell_id, ~cell_id, ~lon, ~lat, ~polity_code,
  "JPN-23-1871-2025@1", 1L, 137.25, 35.25, "JPN-23-1871-2025",
```

```
  "JPN-23-1871-2025@2", 2L, 137.75, 35.25, "JPN-23-1871-2025"
) |>
  dplyr::mutate(
    area_code = 110L,
    start_year = 1952L,
    end_year = 2025L,
    cell_area_ha = 3000,
    land_area_ha = c(1200, 2000)
  )
containment <- tibble::tibble(
  member_code = "JPN-23-1871-2025",
  container_code = "JPN-1952-2025",
  start_year = 1952L,
  end_year = 2025L,
  basis = "prefecture inside JPN-1952-2025 for those years"
)
read_level_country_grid(
  level = 1L,
  support = support,
  containment = containment
)

# The same read scoped to the containers a run grants a depth. Japan's
# reporting code is 110, so the edge above is kept and any other
# container's is left at level 0.
read_level_country_grid(
  level = 1L,
  support = support,
  containment = containment,
  containers = 110L
)

# The same support read at level 0. The default vintage is year-aware,
# which takes this support and keeps its validity intervals; the 2015
# snapshot resolves the support itself and refuses one.
read_level_country_grid(
  level = 0L,
  support = support,
  grid_vintage = "year_aware"
)
```

`read_lpjml_crop_cover` *Read monthly cropland soil cover from an LPJmL run's crop calendar.*

Description

Per-cell, per-month share of a cell's cropped area that is under a growing crop, derived from LPJmL's sowing (`sdate.nc`) and harvest (`hdate.nc`) dates and area-weighted by

`cftfrac.nc`. Supply the result as `data$cropland_cover` to `build_carbon_balance()` to place the cropland soil-cover season where the crops actually are.

Usage

```
read_lpjml_crop_cover(
  run_dir = NULL,
  years = NULL,
  first_year = NULL,
  by = c("cropland", "regime"),
  example = FALSE
)
```

Arguments

`run_dir`	Path to the LPJmL run output directory. `NULL` (default) uses `WHEP_LPJML_RUN_DIR`.
`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the files cover.
`first_year`	Calendar year of the files' first time step. `NULL` (default) reads it from the file's own `time` axis.
`by`	`"cropland"` (default) pools every calendar band into one cover per cell and month. `"regime"` keeps the rainfed and the irrigated bands apart – a band is irrigated when its name starts with `irrigated` – and adds `regime` and `cropped_frac`, the regime's share of the cell, so a consumer can pool the two back by area. Rainfed and irrigated stands of one crop sow and harvest on different dates; the per-regime layer is what the herbaceous rainfed and irrigated crop groups of `build_carbon_balance()` (`crop_groups = list(method = "rotation_groups")`) read, each from its own bands, while plain cropland reads the pool.
`example`	If `TRUE`, return a small fixture instead of reading a run. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `year`, `month` and `cropland_cover` (a fraction in 0-1); with `by = "regime"`, also `regime` (`"rainfed"` or `"irrigated"`) and `cropped_frac`, and up to two rows per cell-month.

Why the temperature proxy is not good enough

Without a calendar, soc_soil_cover_curve is anchored to each cell-year's warmest month as a stand-in for peak canopy. Measured against this run at 2010 over 18,548 cropland cells, the real area-weighted crop mid-season falls in the warmest month in only **5.2%** of them, within one month in 22.6%, and **three or more months away in 51.0%** (median absolute offset three months). Winter cereals, Mediterranean systems and irrigated dry-season crops all grow away from the temperature peak, so the proxy puts modelled full canopy over real fallow and modelled bare soil over the real crop. It is a timing error rather than a level one: the curve's annual mean cover is 0.254 against the calendar's 0.343.

What the calendar does not cover

`sdate`/`hdate` carry 24 bands - the twelve named crops, rainfed and irrigated - while `cftfrac` carries 32. The `others` and managed-grassland bands have no calendar, which is **1.2% of cropped area**; cells are returned with the cover of the area that does have one, and a cell whose cropped area is entirely `others` yields no row rather than a guess.

Band mapping is by NAME, never by index: only 12 of the 24 line up, because `sdate` band 13 is `"irrigated temperate cereals"` where `cftfrac` band 13 is `"rainfed others"`.

Source

Schaphoff, S. et al. (2018). LPJmL4 - a dynamic global vegetation model with managed land - Part 1: Model description. *Geoscientific Model Development*, 11, 1343-1375. doi:10.5194/gmd1113432018.

Examples

```
read_lpjml_crop_cover(example = TRUE)
```

`read_lpjml_grass_productivity` *Read natural-grass productivity from an LPJmL run.*

Description

Sums the natural-grass PFT net primary production bands (the ungrazed natural stand, climate-driven) into a per-cell productivity layer used to distribute grazing livestock by grass production rather than pasture area. The pinned WHEP artifact is used by default; pass `run_dir` to read a local finished LPJmL run, or pass `productivity` / `productivity_path` to use an already-derived custom artifact. Natural-grass NPP is exogenous to the grazing density, so it avoids the livestock to grassland_lsuha to grass-NPP circularity.

Usage

```
read_lpjml_grass_productivity(
  run_dir = NULL,
  years = NULL,
  first_year = NULL,
  example = FALSE,
  productivity = NULL,
  productivity_path = NULL
)
```

Arguments

`run_dir`	Path to the LPJmL run output directory holding `pft_npp.nc`. If unset, the pinned `lpjml-grass-productivity` artifact is used.
`years`	Integer vector of calendar years to read.

`first_year` First calendar year of the run's output time axis.

`example` If `TRUE`, return a small fixture instead of reading a run.

`productivity` Optional already-derived natural-grass productivity tibble/data frame. Takes precedence over pinned data and `run_dir`.

`productivity_path` Optional path to an already-derived natural-grass productivity artifact (`.parquet`, `.csv`, or `.rds`). Takes precedence over pinned data and `run_dir`.

Value

A tibble with `lon`, `lat`, `year` and `grass_npp` (gC/m2/yr).

Examples

```
read_lpjml_grass_productivity(example = TRUE)
```

---

read_lpjml_hydrology *Read an LPJmL hydrology variable into a tidy tibble.*

Description

Reads one monthly LPJmL hydrology output (drainage, evapotranspiration components, precipitation, irrigation, runoff, discharge or soil water content) from a finished run's NetCDF files and returns it in tidy long form. The logical `var` name is mapped to the on-disk file and in-file variable name, so callers need not know the LPJmL naming quirks. The synthetic `"aet"` variable sums the three actual-evapotranspiration components (transpiration, evaporation, interception).

Usage

```
read_lpjml_hydrology(
  var = c("drainage", "transp", "evap", "interc", "aet", "prec", "rain", "irrig",
    "runoff", "runoff_surf", "runoff_lat", "discharge", "swc", "pet", "soiltemp1",
    "soiltemp2", "stand_frac", "cft_nir", "cft_airrig_month", "cft_consump_water_b",
    "cft_consump_water_g"),
  run_dir = NULL,
  years = NULL,
  first_year = NULL,
  monthly = TRUE,
  agg = c("sum", "mean"),
  partial_year = c("abort", "warn", "drop"),
  data = NULL,
  example = FALSE
)
```

Arguments

`var`	Logical variable name, one of `"drainage"`, `"transp"`, `"evap"`, `"interc"`, `"aet"`, `"prec"`, `"rain"`, `"irrig"`, `"runoff"` (total runoff, which already contains the groundwater outflow the `"drainage"` seepage feeds), `"runoff_surf"` and `"runoff_lat"` (surface and lateral soil runoff, which do not), `"discharge"`, `"swc"`, `"pet"` (potential evapotranspiration), `"soiltemp1"` and `"soiltemp2"` (soil temperature by layer), `"stand_frac"` (per-CFT stand area fraction, from `cftfrac.nc`: the weight every other per-CFT cube needs before it can be summed to a cell), `"cft_nir"` (per-CFT net irrigation requirement), `"cft_airrig_month"` (per-CFT APPLIED irrigation by month) or the per-CFT consumptive-water cubes `"cft_consump_water_b"` (blue) and `"cft_consump_water_g"` (green). The per-CFT variables keep their `band` dimension, and carry `band_name` when the file names its bands. `"pet"` and the two soil temperatures exist because the SOC drivers take both from CRU instead, mixing two models' quantities in one expression; see the note at `.hydro_var_map()`. Nothing consumes them yet.
`run_dir`	Path to the LPJmL run output directory. Defaults to `Sys.getenv("WHEP_LPJML_RUN_DIR")`. When neither is set, `"drainage"` is read from the `lpjml-seepage` pin (see *Pinned deep seepage*); every other variable needs a run.
`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year present in the file. A requested year the run does not have aborts, naming the coverage it does have: LPJmL runs ending in different years sit side by side in one folder, so the coverage is a property of `run_dir`, never an assumption of this reader.
`first_year`	First calendar year of the run's monthly time axis. `NULL` (default) reads it from each file's own `time` axis. The last year is not an argument — it is read from the file's own time dimension.
`monthly`	If `TRUE`, return one row per cell-month; if `FALSE`, aggregate the 12 months of each year per cell (flux variables summed, soil water content averaged). Immaterial for the annual per-CFT consumptive-water variables, which LPJmL writes one step per year: they carry no `month` column either way, and aggregating them groups rows that are already one per cell-year-band.
`agg`	Annual aggregation for `monthly = FALSE`, `"sum"` (flux default) or `"mean"` (soil-water default).
`partial_year`	What to do when `monthly = FALSE` and a cell-year does not carry all twelve months, so that aggregating it would silently return a total over eleven. `"abort"` (default) refuses, naming the absent cell-months; `"warn"` returns the short aggregate anyway; `"drop"` removes the incomplete cell-years and reports how many went, so the year is *absent* rather than wrong. Immaterial for the annual per-CFT variables, which carry no `month`. See *Partial years* below.
`data`	Optional pre-read tibble (`lon`, `lat`, `year`, `month`, `value`, plus `layer` for `"swc"` or `band` for `"cft_nir"`) used in place of reading NetCDF, for testing.
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `lon`, `lat`, `year`, `value` (plus `month` for the monthly variables when `monthly = TRUE`, `layer` for `"swc"`, and `band` plus `band_name` for the per-CFT variables). The annual per-CFT consumptive-water variables never carry `month`.

Per-CFT cubes are per-STAND densities

Every per-CFT variable is a density per square metre of ITS OWN CROP'S STAND, not of the gridcell. Summing the bands therefore does NOT give a cell total, and comparing that sum against a whole-cell cube overstates it badly: on the 2026-09-01 run, `"cft_airrig_month"` summed raw across bands is **122.8 times** `"irrig"` (from `mirrig.nc`) for July 2010. Weight each band by its `cftfrac` stand fraction first and the same comparison closes to **0.999**, and to 1.000-1.001 cell by cell on the most irrigated cells.

So: multiply by the stand fraction before aggregating over bands, over cells, or against anything crop-less. The reader returns the model's own units and does not do this for you, because which weighting is wanted depends on the question – a per-hectare-of-crop intensity keeps the per-stand value, a cell or catchment total does not.

Rainfed rice carries applied water

`"cft_airrig_month"` and `"cft_nir"` both book water on the *rainfed* rice band – 36% of the stand-weighted applied total at July 2010. That is LPJmL's paddy management rather than a defect: flooded rice receives water whether or not the stand is classed as irrigated. Filtering to bands whose name starts with `"irrigated"` therefore drops real water.

Partial years

An LPJmL monthly output grows one time step at a time, so a run that was interrupted, or is still writing, leaves a final year with fewer than twelve steps. Aggregating that year sums **eleven** months and says nothing: no value is `NA`, the row count is unchanged (one row per cell-year either way), and every downstream total and identity goes on balancing over the eleven. Reproduced on a real 24-month run truncated to 23 with `ncks -d time,0,22`: the second year's summed deep seepage fell 3.05% over 500 land cells (98,478 to 95,477 mm), with an identical 3,392-row output and no `NA` in any land cell (whep#1073).

The refusal lives here, at the reader, because this is where the absence is created; once the short annual total is downstream it is indistinguishable from a measurement. `years =` was already safe – the coverage check counts only whole years, so requesting a partial one aborts – and `partial_year` closes the `years = NULL` whole-file read that it does not cover.

Pinned deep seepage

`"drainage"` (LPJmL deep seepage, `mseepage.nc`) is the one variable that does not need a local run. Without `run_dir` or `WHEP_LPJML_RUN_DIR` it is read from the `lpjml-seepage` pin: annual seepage (mm per year, the twelve monthly values summed, as `monthly = FALSE` does from a run) per land cell and year, 1850-2023, built from WHEP's reference LPJmL 6.1.1 run by `.hydro_seepage_layer()`, so the pinned and run-derived values are the same numbers. The pin's rows carry the run's `lpjml_version`, `lpjml_git_hash` and `lpjml_run`; the reader returns the same columns as from a run. The pin is annual, so `monthly = TRUE` (and `agg = "mean"`) still needs a run.

Seepage is a whole-cell quantity: LPJmL has no per-crop or per-water-regime seepage, so the irrigated and rainfed land of a cell share one value. Join it on `lon`, `lat` and `year`.

Examples

```
read_lpjml_hydrology(example = TRUE)
```

---

read_lpjml_litterfall

Read litterfall by land-use class from an LPJmL run.

Description

Per-cell, per-year carbon returned to the soil as litter, split by the stand that shed it. Litterfall is what physically enters the soil; net primary production is not, because it also contains the increment that stays in living biomass. On this run natural litterfall is 0.84 times natural NPP at the median cell and 0.68 in aggregate, the shortfall being biomass accumulation, fire, and the conversion pulse.

Usage

```
read_lpjml_litterfall(
  class = c("nv", "agr", "mgrass", "luc", "total"),
  run_dir = NULL,
  years = NULL,
  first_year = NULL,
  example = FALSE
)
```

Arguments

`class`	Which stand's litterfall to read: `"nv"` (standing natural vegetation), `"agr"` (crop stands), `"mgrass"` (managed grassland), `"luc"` (the land-use conversion pulse) or `"total"`.
`run_dir`	Path to the LPJmL run output directory. `NULL` (default) uses `WHEP_LPJML_RUN_DIR`.
`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the file covers.
`first_year`	Calendar year of the file's first time step. `NULL` (default) reads it from the file's own `time` axis, which LPJmL stamps as `"days since YYYY-M-D"`. Pass a value only to override a file that carries no reference date.
`example`	If `TRUE`, return a small fixture instead of reading a run. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `year`, `class` and `litterfall_c_mgc_ha_yr`.

Whole-cell densities

Every class is a density per square metre of GRIDCELL, not of its own stand: each accumulation site in LPJmL multiplies by `stand->frac`. To get the density a stand actually experiences, divide by that stand's area fraction – for `"nv"`, `natural_stand_frac` from `read_lpjml_natural_cover()`. Mixing the two conventions understates a partly-natural cell's input by exactly its natural fraction.

The four classes sum to the total

`nv + luc + agr + mgrass` recovers `total` to float32 rounding, the remainder being set-aside grass. `"luc"` is the pulse released when land is converted, and it is large – 9.9% of all litterfall at 2010 against 1.3% for crop stands. It is booked to no destination: at `landusechange.c` the receiving stand is still natural when the pulse is credited, so apportioning it to the land that gained area is a decision for the caller, not a quantity LPJmL reports.

Source

Schaphoff, S. et al. (2018). LPJmL4 - a dynamic global vegetation model with managed land - Part 1: Model description. *Geoscientific Model Development*, 11, 1343-1375. doi:10.5194/gmd1113432018.

Examples

```
read_lpjml_litterfall(example = TRUE)
```

---

read_lpjml_natural_cover

Read natural-land soil cover from an LPJmL run.

Description

Per-cell, per-year vegetated cover of the natural stand, from LPJmL's foliar projective cover (`fpc.nc`). Band 1 of that file is the natural stand FRACTION – how much of the cell is natural – and bands 2 to 15 are the fourteen natural PFTs' projective cover WITHIN that stand. The cover of natural land is therefore the sum of bands 2 to 15, capped at 1, and never band 1, which is an area share and a different quantity.

Supply the result as `data$natural_cover` to `build_carbon_balance()` to replace soc_soil_cover_curve's constant 0.85 for the natural class.

Usage

```
read_lpjml_natural_cover(
  run_dir = NULL,
  years = NULL,
  first_year = NULL,
  example = FALSE
)
```

Arguments

`run_dir`	Path to the LPJmL run output directory. `NULL` (default) uses `WHEP_LPJML_RUN_DIR`.
`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the file covers.
`first_year`	Calendar year of the file's first time step. `NULL` (default) reads it from the file's own `time` axis.
`example`	If `TRUE`, return a small fixture instead of reading a run. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `year`, `natural_stand_frac` and `natural_cover` (both fractions in 0-1).

Why it matters where it does

The constant is close on the global mean – measured cover runs 0.858 in 1901 to 0.884 in 2023 – and wrong in the distribution: the median natural cell is fully covered at 1.000 while the 5th percentile is bare at roughly zero. The RothC plant-retainment term is `0.6 + 0.4 * (1 - cover)`, so a near-bare cell moves from 0.66 to 1.00, decomposes half again as fast, and loses about a third of its equilibrium carbon. Those arid cells are exactly where the model sits furthest from observation. The effect on the global mean is about 1.8%; per cell it spans 0.66 to 1.10 times.

Managed grassland has no FPC band – `fpc.nc` covers the natural stand only – so it necessarily stays on the curve.

Source

Schaphoff, S. et al. (2018). LPJmL4 - a dynamic global vegetation model with managed land - Part 1: Model description. *Geoscientific Model Development*, 11, 1343-1375. doi:10.5194/gmd1113432018.

Examples

```
read_lpjml_natural_cover(example = TRUE)
```

`read_lpjml_npp` *Read a per-PFT annual LPJmL carbon variable into a tidy tibble.*

Description

Reads one annual per-plant-functional-type (PFT) carbon output of a finished LPJmL run, either net primary production (`"npp"`, from `pft_npp.nc`) or harvested carbon (`"harvestc"`, from `pft_harvestc.nc`), and returns it in tidy long form with the PFT name attached. Values are per-PFT-stand carbon densities in grams of carbon per square metre per year. Requested years are sliced at read time so the full cube is never materialised. The two files index their PFT bands differently, so downstream code should join on `name_pft`, never on the band position.

Usage

```
read_lpjml_npp(
  var = c("npp", "harvestc"),
  years = NULL,
  run_dir = NULL,
  first_year = NULL,
  data = NULL,
  example = FALSE
)
```

Arguments

`var`	Logical variable name, `"npp"` (per-PFT net primary production) or `"harvestc"` (per-PFT harvested carbon).
`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year present in the file.
`run_dir`	Path to the LPJmL run output directory. Defaults to `Sys.getenv("WHEP_LPJML_RUN_DIR")`.
`first_year`	First calendar year of the run's annual time axis. `NULL` (default) reads it from the file's own `time` axis.
`data`	Optional pre-read tibble (`lon`, `lat`, `year`, `npft`, `name_pft`, `value`) used in place of reading NetCDF, for testing.
`example`	If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `lon`, `lat`, `year`, `npft` (band index), `name_pft` (PFT name) and `value` (gC/m2/yr).

Examples

```
read_lpjml_npp(example = TRUE)
```

`read_lpjml_regime_yield` *Read LPJmL irrigated and rainfed crop yields per cell.*

Description

Derives the harvest per unit crop area of the rainfed and the irrigated stand of each crop in each 0.5-degree cell and year from a finished LPJmL run, and maps the LPJmL crop functional types (CFTs) onto WHEP production items. It is the yield ratio that weights the rainfed/irrigated split of a crop row's synthetic nitrogen, production nitrogen and residue removals toward the more productive hectares.

The harvest is `pft_harvestc` (harvested carbon excluding residues), which LPJmL already writes per square metre of the band's own stand; it is not divided by the stand fraction

again. The evidence is in the header of `R/lpjml_regime_yield.R`: weighted by `cftfrac`, the bands reproduce the cell total in `harvestc.nc` exactly on grassland-only cells, while their plain sum overshoots it 1.5 to 26 times.

Usage

```
read_lpjml_regime_yield(
  years = NULL,
  run_dir = NULL,
  data = NULL,
  include_others = FALSE,
  example = FALSE
)
```

Arguments

`years`	Optional integer vector of calendar years. `NULL` reads every year the run's `pft_harvestc.nc` carries. A year the run does not cover aborts, naming the coverage it has.
`run_dir`	Path to a finished LPJmL run output directory holding `pft_harvestc.nc` and `cftfrac.nc`. `NULL` (default) uses `WHEP_LPJML_RUN_DIR`, and with neither set reads the pinned `lpjml-crop-regime-yield` layer, built from one LPJmL run together with the other LPJmL-derived pins. A run that is named but lacks either file aborts rather than falling back to the pin.
`data`	Optional list used in place of reading a run, for testing: `harvestc` (as `read_lpjml_npp()` returns it for `"harvestc"`) and `stand_frac` (as `read_lpjml_hydrology()` returns it for `"stand_frac"`).
`include_others`	If `TRUE`, also return LPJmL's `"others"` catch-all stand, expanded to the items cft_mapping puts on it. Defaults to `FALSE`, the crop-specific stands only.
`example`	If `TRUE`, return a small fixture instead of reading a run. Defaults to `FALSE`.

Value

A tibble with one row per cell, production item and year:

- `lon`, `lat`: cell centre, 0.5-degree grid.
- `year`: calendar year.
- `item_prod_code`, `item_cbs_code`: the WHEP production item and its commodity balance item.
- `lpjml_crop`: the LPJmL CFT the item's yield comes from.
- `yield_rainfed`, `yield_irrigated`: harvested carbon, excluding residues, in grams of carbon per square metre of that regime's own stand per year; `NA` where the regime has no stand in the cell.
- `method_regime_yield`: how the yields were obtained (see *Years before 1901*).

Which crops carry a yield

Only production items mapped in cft_mapping to one of LPJmL's twelve crop-specific CFTs (temperate and tropical cereals, rice, maize, pulses, temperate and tropical roots, sugarcane, and the soybean, groundnut, sunflower and rapeseed oil crops) get rows: 40 items. The other 113 mapped items sit on LPJmL's `"others"` catch-all stand (fruits, vegetables, nuts, fibres, stimulants, oil palm, cotton...), whose yield is that of a composite rather than of the crop, and items absent from cft_mapping (fodder crops among them) have no band at all. By default neither gets a row, so a join against this layer leaves them without a yield, and the caller has to decide what to use instead.

`include_others = TRUE` adds the `"others"` stand as a crop of its own, expanded to the items cft_mapping puts on it. Its yield is the composite stand's, not the item's. The regime yield ratio uses it as the year-to-year anomaly source for crops without a crop-specific CFT, where only the ratio of the two regimes' yields matters; see `build_regime_yield_ratio()`.

The natural key is `item_prod_code`: each production item maps to one CFT, while four `item_cbs_code`s mix CFTs across their production items (2520 "Cereals, Other", temperate and tropical cereals; 2534 "Roots, Other", tropical roots and an unmapped item; 2537 "Sugar beet", temperate roots and an unmapped item; 2605 "Vegetables, Other", where only green maize has a band). `item_cbs_code` is carried for joining, not as a key.

Zero-area stands

A regime whose stand fraction is zero in a cell has no harvest to divide, so its yield is `NA`, not zero. A cell-crop-year gets a row when at least one of the two stands has area. A stand with area and no harvest (a crop that failed or never matured) keeps its real zero.

Years before 1901

The run's climate forcing starts in 1901, and LPJmL recycles the 1901-1930 window for 1750-1900 while land use and CO2 stay historical. Those years are stamped `"lpjml_band_harvest_recycled_clim` in `method_regime_yield`; later years `"lpjml_band_harvest"`.

Rainfed rice

LPJmL waters paddy rice on the rainfed stand too (see `read_lpjml_hydrology()`), so rainfed rice yields are paddy yields and the rice irrigated:rainfed ratio sits closer to one than for an unwatered upland stand.

Source

LPJmL 6.1.1 (Potsdam Institute for Climate Impact Research), fork `lbm364dl/LPJmL` at git hash `e6e6c42b88354f4a0df945e55f2ccf3c4a3cb605` (as stamped in the output metadata), run `global_1750-2023_spinup_300_our_inputs_lpjml611_preindustrial_v2`, outputs `pft_harvestc.nc` and `cftfrac.nc`.

Examples

```
read_lpjml_regime_yield(example = TRUE)
```

`read_lpjml_wind` *Read gridded LPJmL-forcing windspeed onto WHEP's grid.*

Description

Reads the GSWP3-W5E5 monthly windspeed forcing used to drive LPJmL (single consolidated NetCDF, already on WHEP's native 0.5-degree grid) and returns it in tidy long form. The file's `wind` variable carries no `units` attribute, because the monthly aggregation dropped it; the unit is metres per second, taken from the ISIMIP2a GSWP3-W5E5 daily source the file is derived from, which declares `units = "m s-1"` and `standard_name = "wind_speed"`. That is why the output column is named `windspeed_ms`.

The `windspeed_ms` column is named to match the MANNER driver column of the same name, so the output feeds `calculate_nh3()` (`method = "manner"` or `"manner_default"`, whose organic path requires `windspeed_ms`) after a join onto the gridded N-input records; this reader is the package's only source of that driver.

Usage

```
read_lpjml_wind(years = NULL, wind_dir = NULL, example = FALSE)
```

Arguments

`years`	Optional integer vector of calendar years to keep. `NULL` reads every year present in the file (1901-2019 in the currently pinned base; the span is whatever the resolved file holds, not a fixed range).
`wind_dir`	Path to the directory holding a `wind_gswp3-w5e5_<span>_monthly.nc`. The span in that filename is resolved against the directory rather than assumed, so an extended base reads without a code change. Defaults to `Sys.getenv("WHEP_WIND_DIR")`; when that is also unset, the registered `lpjml-wind-isimip-1901-2019` pin is read via `whep_read_file()`.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `year`, `month`, `windspeed_ms`.

See Also

`calculate_nh3()`, which consumes `windspeed_ms`.

Examples

```
read_lpjml_wind(example = TRUE)
```

`read_luh2_landuse` *Read gridded yearly LUH2 land-use-class fractions and areas.*

Description

Read the LUH2 v2h gridded land-use "states" product and aggregate its 12 subgrid states into the four carbon-balance classes (cropland, grassland, natural, urban). Per cell-year-class the `fraction` is the sum of the member states' grid-cell fractions (0..1), LUH2's own share of the **whole** cell. `area_ha` is that class's area inside one polycell: by default the class's share of the cell's LUH2 land, spread over the polycell's own measured land (`area_basis = "polycell_land"`), so the four classes tile the polycell's land exactly and the carbon path uses the same land definition as the nitrogen path. At `resolution = "polity"` the areas are summed to each overlapping polity; a border cell keeps every polity it overlaps.

The cell-to-polity assignment is a **static snapshot**, because LUH2 carries no territorial history: a pre-modern year is the snapshot polity's territory holding that year's land-use composition.

The states grid comes from a `WHEP_LUH2_DIR` tree when there is one, else the reference LUH2-GCB2022 `states.nc` is downloaded on demand from Zenodo (doi:10.5281/zenodo.15556812, CC-BY-4.0), verified against its published MD5 and cached. Whichever is read, the vintage (the NetCDF `source_id`, e.g. `"UofMD-landState-LUH2-GCB2022"`) is recorded on the result with `attach_provenance()`, and a local tree that is not the reference vintage warns: the base v2h release and the annual Global Carbon Budget variants cover different years and do not agree.

Usage

```
read_luh2_landuse(
  resolution = c("grid", "polity"),
  years = NULL,
  states_source = c("auto", "local", "zenodo"),
  area_basis = c("polycell_land", "luh2_fraction"),
  data = NULL,
  example = FALSE
)
```

Arguments

`resolution`	`"grid"` (default, per cell and class) or `"polity"` (aggregated to `area_code` per year and class).
`years`	Optional integer vector of calendar years to keep. `NULL` keeps every year present in the source.
`states_source`	Which states source to read: `"auto"` (default, a `WHEP_LUH2_DIR` tree when present, else the Zenodo download), `"local"` (`WHEP_LUH2_DIR` only, an error without it) or `"zenodo"` (the checksum-verified reference vintage only, ignoring any local tree). Recorded in the provenance record's `input_origin`.

`area_basis` Which land definition the class areas are measured on: `"polycell_land"` (default) spreads each class's share of the cell's LUH2 land over the polycell's measured `land_area_ha`; `"luh2_fraction"` keeps LUH2's own land total (`fraction` times the spherical cell area) and splits it between the polycells of a cell by their share of the cell's land. Both partition the cell identically and differ only in the total spread (~12.78 Gha against ~12.99 Gha globally). The choice is recorded in the `method_land_area` output column.

`data` Named list of pre-loaded inputs bypassing the readers: `states` (raw per-cell-year-state fractions with `lon`, `lat`, `year`, `land_use`, `fraction`) and `country_grid`, the polycell support resolved to one row per cell and `area_code` (`lon`, `lat`, `area_code`, `cell_area_frac` and, for `area_basis = "polycell_land"`, `land_area_ha`). Each falls back to its reader when absent. A support carrying more than one row per cell and `area_code`, or an `NA` one, is refused rather than folded (DA-23).

`example` If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with columns `lon`, `lat`, `area_code`, `year`, `land_use`, `fraction`, `area_ha` and `method_land_area` at `"grid"` resolution; at `"polity"` resolution `lon` and `lat` are dropped and `area_ha` is summed per `(area_code, year, land_use)`. `fraction` stays LUH2's share of the whole cell and is repeated on every polycell of that cell, so under `"polycell_land"` it is a source datum rather than a factor `area_ha` can be recovered from. Both resolutions carry the polity columns below, resolved from the `area_code` the support assigns and the row's `year`; the cell-to-area assignment itself is a static snapshot, which is what LUH2 has, so a pre-modern year is the snapshot cell's area read at that year. When the states grid was read from a NetCDF, a provenance record naming the vintage is attached; read it back with `get_provenance()`.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Source

LUH2 v2h, Hurtt, G. C. et al. (2020). Harmonization of global land use change and management for the period 850-2100 (LUH2) for CMIP6. Geoscientific Model Development 13, 5425-5464. doi:10.5194/gmd1354252020. The reference payload is the Global Carbon Budget vintage of that release: Chini, L. et al. (2021). Land-use harmonization datasets for annual global carbon budgets. Earth System Science Data 13, 4175-4189. doi:10.5194/essd1341752021. Data: LUH2-GCB2022, doi:10.5281/zenodo.15556812 (CC-BY-4.0).

Examples

```
read_luh2_landuse(example = TRUE)
```

---

read_luh2_terrestrial

Read the LUH2 terrestrial-area validation layer

Description

Reads `staticData_quarterdeg.nc` and returns `(1 - icwtr) * carea` summed to the 0.5-degree grid: the terrestrial area LUH2 itself implies. This is the DA-5 validation layer for `build_polycell_support()` and is never a production mask, because `icwtr` includes the ocean as well as ice and inland water, and because LUH2 misses small islands its own 0.25-degree mask calls sea.

Usage

```
read_luh2_terrestrial(vintage = c("GCB2022", "v2h"), dir = NULL)
```

Arguments

vintage	Which LUH2 tree to read. "GCB2022" (default) is the UofMD-landState-LUH2-GCB2022 release under Sys.getenv("WHEP_LUH2_DIR"); "v2h" is the base release under Sys.getenv("WHEP_LUH2_V2H_DIR").
dir	Directory holding staticData_quarterdeg.nc, overriding the vintage's environment variable.

Value

A tibble with lon, lat and terrestrial_ha on the 0.5-degree grid, carrying the vintage in its "luh2_vintage" attribute.

Examples

```
# Requires WHEP_LUH2_DIR to be set; not run without it.
if (nzchar(Sys.getenv("WHEP_LUH2_DIR"))) {
  read_luh2_terrestrial(vintage = "GCB2022")
}
```

---

read_n_deposition *Read a HaNi atmospheric nitrogen deposition species onto WHEP's grid.*

Description

Reads one HaNi NHx or NOy deposition NetCDF (native 5-arcmin grid, total grams N deposited per native cell per year) and aggregates it to WHEP's 0.5-degree grid by summing the 6x6 fine cells inside each 0.5-degree block, since the source quantity is an extensive mass. Returns the summed mass per 0.5-degree cell; converting to a per-hectare rate needs the true cell area and is done downstream by build_n_deposition().

Usage

```
read_n_deposition(
  species = c("nhx", "noy"),
  hani_dir = NULL,
  years = NULL,
  example = FALSE
)
```

Arguments

species	Which HaNi species to read, "nhx" or "noy".
hani_dir	Path to the directory holding ndep_nhx.nc and ndep_noy.nc. Defaults to Sys.getenv("WHEP_HANI_DIR").
years	Optional integer vector of calendar years to keep. NULL reads every year present in the file.
example	If TRUE, return a small fixture instead of reading data. Defaults to FALSE.

Value

A tibble with `lon`, `lat`, `year`, `value_g` (total grams N deposited in the 0.5-degree cell that year) and `method_deposition`, the constant `"hani"`. The provenance tag travels with the field so that `build_n_deposition()` can record where its deposition came from instead of asserting it; see that function's `method_deposition` note.

Measured European bias, growing backwards in time

HaNi is **too flat over Europe**: it largely misses both the European deposition peak around 1990 and the fall that emission controls drove afterwards. Measured against the EMEP MSC-W model (rv5.6, 2025 reporting round) on WHEP's own 0.5-degree grid, over the 3193 cells assigned to an EMEP core country, area-weighted kg N/ha/yr:

year	HaNi	EMEP	HaNi / EMEP
1990	9.81	14.08	0.696
2000	9.65	11.28	0.856
2010	9.04	9.56	0.946
2019	8.00	8.07	0.992

Across 1990-2019 HaNi falls 18.4% where EMEP falls 42.7%, and the shortfall summed over those cells and years is 22.6 Tg N.

Those are whole-cell figures, and part of the level gap in them is a measurement artifact: HaNi's mass is deposition to **land** within the cell while EMEP is a density over the **whole** cell, sea included, so dividing HaNi by whole-cell area reads low in every coastal cell (ratio 0.35 in 1990 over cells under half land). The **trend** does not depend on it. Restricted to the cells whose territory in `build_polycell_support()` is at least 98% of the cell (2605 cells):

year	HaNi	EMEP	HaNi / EMEP
1990	10.6	14.6	0.727
2000	10.4	11.5	0.901
2010	9.71	9.88	0.982
2019	8.61	8.28	1.04

HaNi falls 19% and EMEP 43%. The error is therefore a **function of time**, largest exactly when European deposition was largest, and it compounds backwards into the pre-1990 period where EMEP offers no check at all.

This reader returns HaNi as published. `correct_n_deposition()` applies an opt-in, year-resolved correction of the European trend toward EMEP (`read_emep_deposition()`), stamped `method_deposition = "hani_emep_trend"` on the rows it touches; whether it becomes the default, and how it is carried back before 1990, is the open decision in whep#1121. `validation/n_deposition_emep.R` reproduces both tables and writes the full per-country, per-year ratio series.

The comparison product is the EMEP MSC-W chemical transport model, run in support of the Convention on Long-Range Transboundary Air Pollution: Simpson, D. *et al.* (2012). The EMEP MSC-W chemical transport model – technical description. *Atmospheric Chemistry and Physics* 12(16), 7825-7865. doi:10.5194/acp1278252012

Source

Tian, H., Bian, Z., Shi, H., Qin, X., Pan, N., Lu, C., Pan, S., Tubiello, F. N., Chang, J., Conchedda, G., Liu, J., Mueller, N., Nishina, K., Xu, R., Yang, J., You, L. and Zhang, B. (2022). History of anthropogenic Nitrogen inputs (HaNi) to the terrestrial biosphere: a 5 arcmin resolution annual dataset from 1860 to 2019. *Earth System Science Data* 14(10), 4551-4568. doi:10.5194/essd1445512022

Examples

```
read_n_deposition(example = TRUE)
```

---

read_polycell_support

Read the polycell support table from its registered pin

Description

Resolves the versioned `polycell_support` input (DA-17), preferring a local parquet named by `Sys.getenv("WHEP_POLYCELL_SUPPORT_PATH")` so a development build can be used before it is published.

Usage

```
read_polycell_support(
  path = NULL,
  version = NULL,
  require_layers = TRUE,
  role = c("partition", "overlap", "all")
)
```

Arguments

`path`	Optional path to a local parquet, overriding the environment variable and the pin.
`version`	Pin version, passed to `whep_read_file()`. `NULL` takes the version frozen in whep_inputs.
`require_layers`	Whether to refuse a support built without its inland water and ice layers. `TRUE` (default) aborts with class `whep_polycell_absent_layers`; see *Refusing a zero-filled support*. `FALSE` returns the table anyway, for a caller that needs the territory and not the land/water/ice split and says so.
`role`	Which layer to return. `"partition"` (default) is the rows that partition each cell – every row of a table built with the default `aggregates = "exclude"`, and every row of any table published before whep#803. `"overlap"` is the aggregate layer alone, for a consumer that needs the territory of a reporting bucket whose only polity is an aggregate; it aborts

rather than returning nothing when the table carries no such layer. `"all"` returns both and is only correct where the two are kept apart afterwards – summing across them double-counts every member an aggregate covers.

Value

A `tibble` in the `build_polycell_support()` grain.

Refusing a zero-filled support

`water` and `ice` are optional arguments of `build_polycell_support()` and zero-fill when absent, so a support can be published with every lake, river and glacier inside a polity booked as land. Nothing in the table's own arithmetic can see that: the identity `polity_area_ha == land_area_ha + inland_water_ha + ice_area_ha` holds to `max |residual| = 0 ha` either way, because zero satisfies it. It has happened twice – `20260818T105426Z-a0330` (whep#885) and `20260827T190201Z-f82a2` (whep#1010) – and on the second, 2015 land was 534.9 Mha (+4.1%) too high across 94.3% of the `(cell, area_code)` groups the gridded carbon path reads.

So this reader checks, and **aborts** rather than warning. Both times the defect reached a pin there was a warning to see: whep#885 added `cli_warn()` to the producer's zero-fill branch, and `verify_polycell_support.R` prints one when the layer's environment variable is unset. A warning inside a build that runs for hours is not a gate. Every consumer of this table divides a nutrient or carbon mass by land area, so a 4% error in the denominator is not a caveat to carry forward.

What is checked is whether the layers were **supplied**, never whether the totals reconcile. A table built since whep#1010 carries `layers_supplied`, and the check is an equality on that label. On one published before it, the fallback is that neither `inland_water_ha` nor `ice_area_ha` is identically zero – scale-free, so it holds on a single-country development build as well as on the global pin, and a global zero is not a plausible measurement of either quantity.

A support table may carry a second, **non-partitioning** layer: the aggregate polities of `build_polycell_support()(aggregates = "overlap_layer")`, whose polygons cover their members' and therefore claim ground twice. This returns the partition alone unless asked otherwise, so a consumer that never heard of the layer cannot pick a row of it up by accident.

Examples

```
# Requires WHEP_POLYCELL_SUPPORT_PATH or a published pin; not run without it.
if (nzchar(Sys.getenv("WHEP_POLYCELL_SUPPORT_PATH"))) {
  read_polycell_support()
}
```

`read_population` *Read national population on WHEP area codes.*

Description

Reads the `gdp-population` pin and returns population per country and year on the numeric `area_code` the rest of the package uses. The pin is keyed by ISO3 (in a column confusingly also called `area_code`) and reports population in thousands; both are converted here, so consumers such as `build_food_supply()` and `build_n_percapita()` get the `year`/`area_code`/`population` contract they document.

Regional residual aggregates in the pin (`RAFR`, `RASI`, `REUR`, `RLAM`, `ROCE`) are not countries and carry no numeric code. They are dropped and reported rather than silently discarded, since their omission is what makes the result a countries-only total rather than a world total.

`area_code` here is `polity_area_code`, which is a **bucket, not an identity**: several ISO3 codes can share one code, and this function sums them, so some rows are aggregates of more than one territory. WHEP models the reporting members of bucket 999 in their own right, so with the real pin and the default options the only such row is code 206 "Sudan (former)", carrying Sudan plus South Sudan from 2012 on; it resolves to `SUD-1956-2011`, the pre-secession territory its two members together cover. Under `options(whep.unfold_rest_of_world = "none")` code 999 folds Syria, North Macedonia, Palestine, Eswatini, Equatorial Guinea and French Guiana as well. Every folded row is named in a message.

The pin does not cover every area WHEP models, and that is a bigger gap than the fold. On the real pin 190 of the 256 area codes the crosswalk resolves get a population row; the 66 that do not include Bhutan, Comoros, New Caledonia and the Faroe Islands, all of which the commodity balances do give food to. `build_food_supply()` and `build_n_percapita()` inner-join this table, so those areas are absent from their per-capita output rather than wrong in it. Both warn and name them instead of dropping them silently; `options(whep.warn_missing_population = FALSE)` silences that warning.

`population_source = "pin_wpp_fallback"` fills the country-years the pin does not reach from `read_wpp_population()`, and **only** those: the pin wins wherever both have a value, so turning it on cannot move a denominator that was already published. On the real inputs it adds 44 areas the pin has no row for at all — Réunion, Bhutan, Comoros, Western Sahara, New Caledonia, the French overseas departments and the small island states — and 4,755 country-years inside the pin's own year span. Filled rows are stamped `source_pop = "UN WPP 2024"`.

It is a gap-filler and not a replacement, because the two sources disagree where they overlap: across 12,309 shared country-years they differ by a median 0.64%, a 95th percentile of 4.4% and a maximum of 81%. That is why the pin keeps every value it has under every composition.

`population_source = "pin_wpp_fbs_fallback"` then fills what NEITHER of those reaches from `read_fbs_population()`, the FAOSTAT Food Balance Sheet population. That source is keyed on the FAOSTAT area code rather than a present-day ISO3, so it is the only one that reaches a dissolved reporting area: it closes area 186 Serbia and Montenegro over 1992-2005 (#862) and area 151 Netherlands Antilles over 1961-2010 (#787), the two largest holes in the denominator. It is anti-joined like the WPP fill, so it too cannot move a denominator that was already published.

It is the default (#1133). The sources disagree on the value, not just on the coverage. For area 186 in 2000 FAOSTAT gives 10,801,000; a UN WPP 2024 territorial sum for the same ground (`SRB + MNE + XKX`) gives 10,104,000, 6.5% lower; and the `SRB + MNE` sum a

successor walk can actually reach today gives 8,311,000, 23% lower, because WPP publishes Kosovo separately and it carries no WHEP area code (#863). Under the default `territory_overlap = "federation"` area 186 keeps the FAOSTAT figure and the Serbia and Montenegro rows from UN WPP are dropped in those years, so the federation's denominator covers the same territory, from the same publisher, as the FAOSTAT food divided by it. Measured on the real inputs, the default adds 45 areas to the pin's 190 and changes no value the pin already carries; `"pin"` reproduces the earlier, pin-only table.

`population_source = "pin_fbs_fallback"` is FAOSTAT's own population behind the pin, skipping UN WPP. It reaches the two dissolved areas but only 12 of the 45 areas the pin lacks (202 areas against 235), because the Food Balance Sheets do not report Bhutan, Comoros, the French overseas departments or most small island states. Where both FBS and WPP would fill the same country-year they agree to a median of about 1e-6.

The UN WPP fill stops at 2023, the last year WPP 2024 publishes as an estimate; 2024-2100 are medium-variant projections and are dropped, with a message, rather than returned as population.

Neither fill can overwrite a key the previous source already has, but an anti-join on `(year, area_code)` cannot see two **different** codes naming the same ground in the same year, and all three sources produce such pairs: the pin reports `167` Czechia inside `51` Czechoslovakia for 1850–1992, the UN WPP fill adds the Soviet, Yugoslav and Belgium-Luxembourg successor states in the federation's own years, and the FBS fill adds `186` Serbia and Montenegro beside the `272/273` rows the WPP fill has already added. `territory_overlap` decides which row survives; the overlaps are found from the polities database's `successor` relation, so the answer follows upstream rather than a list kept here (#939).

Neither ISO3-keyed source can reach an area whose territory no longer exists, because both are keyed on a present-day ISO3 code. `population_source_reach()` reports which areas that leaves out and whether the polities database's `successor` relation could stand in for them. Against UN WPP 2024's vocabulary, exactly one reporting area outside the Rest-of-World bucket is unreachable by either route: area 151 Netherlands Antilles, `ANT-1961-2010`, which carries commodity-balance food in every year from 1961 to 2010 and for which upstream publishes no successor at all (#787). Reachable is not the same as safe to sum — see that function and the note at the top of `R/population_reach.R`.

Usage

```
read_population(
  years = NULL,
  data = list(),
  population_source = c("pin_wpp_fbs_fallback", "pin", "pin_wpp_fallback",
    "pin_fbs_fallback"),
  territory_overlap = c("federation", "successors", "none"),
  example = FALSE
)
```

Arguments

`years`	Optional integer vector of calendar years to keep. `NULL` (default) keeps every year the pin covers.

`data` Optional named list of pre-loaded inputs to avoid the pin read: `gdp_population` (the raw pin, with `Year`, `area_code` as ISO3, `pop` in thousands), `wpp_population` (a `read_wpp_population()` output) and `fbs_population` (a `read_fbs_population()` output). Falls back to `whep_read_file()` when absent.

`population_source`
`"pin_wpp_fbs_fallback"` (default), which fills the country-years the `gdp-population` pin does not cover from UN WPP and then what neither reaches from `read_fbs_population()`; `"pin"`, the pin alone; `"pin_wpp_fallback"`, the pin then UN WPP; or `"pin_fbs_fallback"`, which fills from `read_fbs_population()` alone and never reads UN WPP.

`territory_overlap`
Which row survives when two area codes describe overlapping territory in the same year, as a dissolved federation and its successor states do. Overlaps are found from the polities database's `successor` relation, transitively.

- `"federation"` (default) keeps the federation's row and drops the successor states' rows in the years the federation reports. WHEP's own numerator sits on the federation code in exactly those years — the commodity balances carry area 228 for 1961–1991 and Russia only from 1992, area 51 to 1992 and Czechia only from 1993, area 15 to 1999 and Belgium only from 2000 — so this is the row a per-capita divide can actually use, and the sum over areas is the territory counted once.
- `"successors"` keeps the successor states and drops the federation. It is the finer grain, and it does not reproduce the federation's own figure: on `"pin_wpp_fbs_fallback"` the successor sum falls 1.6% short of the USSR at 1961 and runs 2.0% over it at 1991, and falls 14.8% (1961) to 17.5% (1991) short of the Yugoslav SFR — the Kosovo gap of #863. Before 1950 no source has the successors at all, so the federation's population is lost rather than redistributed. It also removes three areas from the table outright, because they exist only in the overlapping years: 51 Czechoslovakia, and — undoing exactly what the FBS fill is for — 186 Serbia and Montenegro (#862) and 151 Netherlands Antilles (#787).
- `"none"` composes the sources exactly as before and **warns**. Any sum over areas then double counts; use it only to reproduce a number published before this argument existed.

The default answers "whose is the USSR's 1961?" the same way `resolve_polity_lineage()` does on the spatial side (#1004, #1114): its default `basis = "historical_polity"` places a 1961 row keyed on area 185 Russia onto `F228-1945-1991`, the polity the grid support carries that year, and its alternative `"constant_territory"` keeps the modern reporter, exactly as `"successors"` does here. The two are complementary, not competing: that function says which polity's **cells** a row belongs on and moves no value, while this one says which of two **rows** for one territory survives. The package therefore has one convention, stated in both places and selectable in both.

`example` If `TRUE`, return a small fixture instead of reading remote data. Defaults to `FALSE`.

Value

A tibble with `year`, `area_code`, `population` (persons), `source_pop` and `method_territory_overlap`, one row per area code and year, sorted by year then area code, plus the polity columns below. `source_pop` carries the pin's own vocabulary (`"Original"`, `"Linear interpolation"`, `"First value carried backwards"`), joined with `" + "` when a bucket sums ISO3 codes of differing provenance, or `"UN WPP 2024"` for a fallback-filled row. A row is one country in the common case, but `area_code` is an aggregation bucket: rows from 2012 on 206 ("Sudan (former)") are sums over several territories rather than a single country, as are rows on 999 ("Rest of World") when the Rest-of-World fold is restored.

Polity columns

Every area-keyed output carries the polity its `area_code` resolves to in that row's year:

- `polity_area_code`: The numeric key rows are AGGREGATED on, for the matrix workflows. It is a bucket, not an identity: use `reporting_polity_code` to say which territory a row belongs to.
- `reporting_polity_code`: The polity itself, e.g. `ESP-1846-1914`. It is year-aware, so the same `area_code` resolves to different polities in different years, which is the point of the crosswalk.
- `reporting_polity_name`: Its name. It can differ from the area's own name where the area folds into an aggregate.
- `reporting_polity_has_geometry`: Whether the polity has a polygon in the WHEP polity database, for callers that need to map or intersect it. `FALSE` is a documented gap upstream, not an error.

Rows whose `area_code` resolves to no polity keep the columns with `NA` rather than being dropped, so a gap is visible instead of silent.

Rows before the back-cast anchor year resolve to the polity live in that anchor year rather than to the polity live in the row's own year, because WHEP's pre-anchor series are back-cast onto the anchor-year territory. See `add_polity_code()` for the reasoning. Where that polity is not live in the row's own year – 41.5% of the pre-1961 `(area, year)` cells – `add_polity_code()` says so as `mapping_status == "backcast_anchor"`, and `polity_coverage_gaps()` reports it as `gap_kind == "backcast_anchor"`. These columns do not say so either way.

A row whose year no mapped period covers is resolved to the NEAREST period of the same area instead, so `reporting_polity_code` can name a polity that did not exist in that row's year – FAOSTAT bucket 206 "Sudan (former)" keeps reporting after `SUD-1956-2011` ends, and its post-2011 rows carry that code. These columns do not say so: `add_polity_code()` reports such a row as `mapping_status == "out_of_span"`, and that column is dropped here so that adding it does not change the schema of every area-keyed output at once. `polity_coverage_gaps()` reports the stand-in rows of a built table, and `options(whep.polity_mapping_status = "flag")` (or `"status"`) carries the signal on the outputs themselves. Both are opt-in; the default is no extra column.

Examples

```
read_population(example = TRUE)
```

`read_soil_hydraulic` *Read gridded soil hydraulic properties from HWSD onto WHEP's grid.*

Description

Reads the HWSD (Harmonized World Soil Database) soil map unit attribute table and raster, resolves each map unit's dominant USDA texture class, looks up that class's volumetric field capacity, wilting point and porosity from soil_hydraulic_by_texture (via the hwsd_texture_usda code crosswalk), and aggregates each property to WHEP's 0.5-degree grid by averaging the native HWSD cells inside each 0.5-degree block. These are the per-cell soil hydraulic drivers the ICBM soil-carbon moisture modifier consumes. Soil texture is a static HWSD property: the result has no `year` column. Cropping to `data$cell_polity` follows the same regional-crop path as `read_soil_ph()`; missing cells are gap-filled from the nearest available neighbour when a target grid is supplied.

Usage

```
read_soil_hydraulic(
  hwsd_dir = NULL,
  data = list(),
  source = c("auto", "pin", "local"),
  version = NULL,
  example = FALSE
)
```

Arguments

`hwsd_dir`	Path to the directory holding `hwsd_data.csv` and `hwsd.bil`. Defaults to `Sys.getenv("WHEP_HWSD_DIR")`. Supplying it derives the grid locally rather than reading the pin.
`data`	Optional named list of pre-loaded inputs: `cell_polity` (`lon`, `lat`, at minimum), used both to crop the HWSD raster and as the gap-filling target grid.
`source`	Where the grid comes from. `"auto"` (default) reads the published pin unless `hwsd_dir` is given; `"pin"` always reads it; `"local"` always derives it from an HWSD archive. The pin is preferred so that every user shares one vintage: HWSD exists in two incompatible versions (v1.2 topsoil 0-30 cm, HWSD2 D1 0-20 cm) and the carbon balance reports 0-30 cm.
`version`	Pin version, passed to `whep_read_file()`. `NULL` takes the version frozen in `whep_inputs`.
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `t_field` (volumetric field capacity), `t_wilt` (volumetric wilting point) and `porosity`, each a fraction.

Caching

Aggregating the HWSD raster to the 0.5-degree grid takes about an hour per pass, and its result depends only on the archive and the target grid, so it is cached under `rappdirs::user_cache_dir("whep")`. The cache key covers the archive's raster and header (size and modification time), the resolution, the requested columns, the target grid's cells and the derived map-unit values, plus an algorithm version that is bumped whenever a change would move the numbers. Set `WHEP_HWSD_CACHE_DIR` to relocate it; the test suite points it at a temporary directory so a fixture-derived grid can never reach a real cache.

Examples

```
read_soil_hydraulic(example = TRUE)
```

`read_soil_ph` *Read gridded soil pH onto WHEP's grid.*

Description

Reads the HWSD (Harmonized World Soil Database) soil map unit attribute table and raster, derives each map unit's pH from its dominant USDA texture class, and aggregates the result to WHEP's 0.5-degree grid by averaging the native HWSD cells inside each 0.5-degree block. Soil pH is a static HWSD property: the result has no `year` column. When `data$cell_polity` is supplied, the native HWSD raster is first cropped to that grid's extent before reclassification (so a regional caller never materialises or reclassifies the full-resolution global raster), and cells present in that target grid but missing from the aggregated HWSD grid are gap-filled from the nearest available neighbour; otherwise cropping and gap-filling are both skipped and the returned grid covers every cell where HWSD itself has data.

Usage

```
read_soil_ph(hwsd_dir = NULL, data = list(), example = FALSE)
```

Arguments

`hwsd_dir`	Path to the directory holding `hwsd_data.csv` and `hwsd.bil`. Defaults to `Sys.getenv("WHEP_HWSD_DIR")`.
`data`	Optional named list of pre-loaded inputs: `cell_polity` (`lon`, `lat`, at minimum), used both to crop the HWSD raster to the region of interest before reclassification and as the target grid for gap-filling. When absent, cropping and gap-filling are both skipped (documented fallback above).
`example`	If `TRUE`, return a small fixture instead of reading data. Defaults to `FALSE`.

Value

A tibble with `lon`, `lat`, `soil_ph`.

Caching

Aggregating the HWSD raster to the 0.5-degree grid takes about an hour per pass, and its result depends only on the archive and the target grid, so it is cached under `rappdirs::user_cache_dir("whep")`. The cache key covers the archive's raster and header (size and modification time), the resolution, the requested columns, the target grid's cells and the derived map-unit values, plus an algorithm version that is bumped whenever a change would move the numbers. Set `WHEP_HWSD_CACHE_DIR` to relocate it; the test suite points it at a temporary directory so a fixture-derived grid can never reach a real cache.

Examples

```
read_soil_ph(example = TRUE)
```

---

read_spam_yields *Read SPAM harvested area, production and yield by crop and technology.*

Description

Reads the SPAM (Spatial Production Allocation Model) global gridded crop tables for the irrigated ("I") and all-rainfed ("R") technologies, per ~5-arcmin pixel and crop, and joins harvested area, production and yield into one row per (pixel, crop, technology). This is the per-crop, per-country "Level" input to the irrigated:rainfed regime yield ratio of `build_regime_yield_ratio()`; it does not itself compute the ratio.

`vintage = "2010"` (SPAM2010 v2.0, doi:10.7910/DVN/PRFF8V) is fetched automatically: the three needed Global_CSV zips (harvested area, production, yield) are downloaded from Harvard Dataverse on first use, verified against their published MD5s, and cached under `rappdirs::user_cache_dir("whep")`; `dir` or the `WHEP_SPAM_DIR` environment variable overrides the cache with a directory of your own (holding the zip files, or their already-extracted `_TI`/`_TR` members).

`vintage = "2020"` (SPAM2020 v2.0, doi:10.7910/DVN/SWPENT) is **never** downloaded: Harvard Dataverse gates every file of that dataset behind a mandatory guestbook requiring an email and institution, and the API's documented way past it needs a logged-in Dataverse account submitting that guestbook response. `dir`/`WHEP_SPAM_DIR` therefore must point at a directory you have already populated by hand – fill the guestbook once in a browser at the dataset's Dataverse page, download the three Global_CSV zips, and pass their directory. The zips are verified against the MD5s Dataverse publishes on the record (not re-checked against a live download, since none is attempted); the column layout is read through the exact same parser as SPAM2010, and aborts clearly, naming what it found, if the expected identifying or crop columns are not there – SPAM2020's layout has not been verified against SPAM2010's, since the guestbook blocks even its own 6 KB ReadMe.

Usage

```
read_spam_yields(vintage = c("2010", "2020"), dir = NULL, example = FALSE)
```

Arguments

`vintage`	Which SPAM release to read: `"2010"` (default, fetched automatically) or `"2020"` (never fetched; needs `dir`/`WHEP_SPAM_DIR`, see Description).
`dir`	Optional path to a directory holding the SPAM Global_CSV zip files (or their extracted `_TI`/`_TR` members), overriding `WHEP_SPAM_DIR`. A `vintage`-named subdirectory is read from it (e.g. `file.path(dir, "2010")`), matching what `inst/scripts/download/download_spam.R` writes.
`example`	If `TRUE`, return a small fixture instead of reading SPAM data. Defaults to `FALSE`.

Value

A tibble with `cell5m` (SPAM's pixel id), `lon`, `lat`, `iso3`, `name_cntr`, `name_adm1`, `name_adm2`, `alloc_key` (SPAM's own admin-coded pixel key), `spam_crop` (SPAM's short crop code, e.g. `"whea"`), `technology` (`"I"` irrigated or `"R"` all-rainfed), `harvested_area_ha`, `production_t`, `yield_kg_ha` (as SPAM publishes it, not converted), `vintage` and `method_spam_source` (`"cache"`: fetched and MD5-verified on demand; `"user_supplied"`: read from `dir`/`WHEP_SPAM_DIR`, MD5-verified when a zip was found there, unverified when only already-extracted CSVs were). A provenance record (DOI, vintage, origin) is attached; read it back with `get_provenance()`.

Source

Yu, Q. et al. (2020). A cultivated planet in 2010 – Part 2: The global gridded agricultural-production maps. Earth System Science Data 12, 3545-3572. doi:10.5194/essd123545-2020. Data: SPAM2010 v2.0, doi:10.7910/DVN/PRFF8V (CC-BY 4.0). SPAM2020 v2.0: doi:10.7910/DVN/SWPENT (CC-BY 4.0; gated behind a Harvard Dataverse guestbook, see Description).

Examples

```
read_spam_yields(example = TRUE)
```

`read_wpp_population` *Read UN World Population Prospects population.*

Description

Reads the UN WPP 2024 medium-variant population by five-year age group and sex, resolved to WHEP area codes. `by = "total"` sums to one row per country and year; `by = "age_sex"` keeps the age and sex detail that `build_protein_requirement()` needs.

The file is resolved in order: the `dir` argument, then the `WHEP_WPP_DIR` environment variable, then a cache under `rappdirs::user_cache_dir("whep")`, downloading it on first use. Population is converted from the thousands WPP publishes to persons.

Only `Country/Area` locations with an ISO3 code are kept, so WPP's regional and income-group aggregates cannot leak into a per-country denominator.

Usage

```
read_wpp_population(
  years = NULL,
  by = c("total", "age_sex"),
  data = NULL,
  dir = NULL
)
```

Arguments

`years`	Optional integer vector of years to keep. `NULL` (default) keeps every year in the file.
`by`	Output grain: `"total"` (default, one row per `year` and `area_code`) or `"age_sex"` (adds `age_start`, `age_span` and `sex`).
`data`	Optional pre-read WPP table, bypassing the file entirely. Used by the tests so the whole path stays offline.
`dir`	Optional directory holding the WPP CSV, `WPP2024_PopulationByAge5GroupSex_Medium.csv.g`

Value

A tibble with `year`, `area_code`, `population` (persons) and, for `by = "age_sex"`, `age_start`, `age_span` and `sex` (`"m"` / `"f"`). ISO3 codes the crosswalk does not resolve are dropped and named, rather than returned on a missing `area_code`.

Examples

```
read_wpp_population(
  by = "age_sex",
  data = tibble::tribble(
    ~ISO3_code, ~LocTypeName,   ~Time, ~AgeGrpStart, ~AgeGrpSpan,
    ~PopMale, ~PopFemale,
    "ESP",      "Country/Area", 2010L, 0L,           5L,
    1170.5,    1103.2
  )
)
```

---

reconcile_admin_allocation

Reconcile a level allocation against the national totals that bind

---

Description

The diagnostic half of `allocate_level_crops()`. The allocation itself decides nothing here: this reads the tables that function returns and says, per container and per unit, how far the administrative statistics sat from the national total they were never allowed to move, what the discarded production series would have implied, and where a unit's target outran the land under it.

Two things it deliberately does not do. It never re-derives a share, so a number in this report is the number the allocator used; and it reads the breach as the **table** `allocate_level_crops()` returns, never by parsing the warning the engine also emits.

Usage

```
reconcile_admin_allocation(
  allocation,
  admin_shares = NULL,
  intensity = list(),
  tolerance_relative = 0.1,
  tolerance_absolute = 1000
)
```

Arguments

`allocation`	The list `allocate_level_crops()` returns. Its `targets`, `coverage` and `breach` elements are read; the others are ignored, so a stored subset of the three is a valid input.
`admin_shares`	The resolved admin shares the allocation was given, BEFORE the allocator drops non-area indicators: the production rows it discards are what the divergence diagnostic reads. `NULL` leaves the indicator, production and bridge columns empty or `NA`, and a table matching no `(container, item)` of the allocation warns rather than degrading to that same empty report in silence.
`intensity`	Named list for the cropping-intensity diagnostic, with `unit_cropland` (`year`, `area_code`, `level_polity_code`, `cropland_ha`, as `unit_cropland_extent()` returns) and `mc_national` (either a single number or a tibble of `year`, `area_code`, `mc_factor`). Any other name is refused rather than ignored.
`tolerance_relative`	Relative discrepancy above which a complete-coverage group is refused; `0.10` by decision T31(d).
`tolerance_absolute`	Absolute discrepancy, in hectares or head, above which the same group is refused; `1000` by decision T31(d). BOTH must be breached.

Value

A list of three tibbles:

- `groups`: one row per `(year, area_code, item_prod_code)` with the binding `indicator`, `national_total`, `admin_sum`, `n_units_reporting`, `n_units_valid`, `coverage`, `coverage_complete`, the allocator's `basis`, `residual_target`, `discrepancy`, `discrepancy_frac`, `beyond_tolerance`, `n_units_production` and `production_divergence_tvd`. `n_units_valid` is `allocate_level_crops()`'s own `n_units`: the units of the allocation layer valid for that year, which is the denominator its coverage decision – and so its choice between rescaling and a residual – actually used. Decision T31(b)'s pattern extension gives every granted unit a row even where the crop has no gridded pattern inside it, so a unit is not quietly dropped from the denominator for lacking one.

- `units`: one row per `(year, area_code, level_polity_code, item_prod_code)` with `area_share`, `production_share`, `area_share_renorm`, `share_divergence`, `implied_yield_ratio`, `production_basis`, the irrigation-floor binding and its per-unit-year count, `sown_ha`, `cropland_ha`, `cropping_intensity`, `mc_factor_national`, `intensity_exceeds_mc`, and the capacity breach at both multi-cropping factors.
- `bridges`: one row per `(area_code, item_prod_code)` that carried any year, with `n_years_observed`, `n_years_carried`, `longest_run`, `longest_interior_run` and the longest interior run's span and treatments.

What binds and what is only measured

The national total binds and the reported area shares set the shape within the country (decisions 2 and 8). So a difference between the national total and the sum of the reported units is a property of the evidence, not of the allocation: it is reported, and past both tolerances it refuses the run, but it never moves a national total.

`discrepancy` is `national_total - admin_sum` and `discrepancy_frac` divides it by the national total, which is `NA` where that total is zero – the absolute hectares stay beside it, because a zero total is exactly the case where the fraction says nothing and the level does. A group whose fraction cannot be evaluated is never refused: refusal asks for BOTH tolerances, and one of them has no value.

`admin_sum` is `NA`, not `0`, where no unit reported an absolute area (`basis` is `"pattern"` or `"share_normalised"`). The sum of nothing is zero and would read as a discrepancy of the whole national total, which would refuse every consented shares-only family in the pin while nothing at all had been measured against it. Hectares can only be compared with hectares.

`admin_sum` is also the point where a failure to harmonise units becomes visible: the readers convert to hectares, tonnes and heads before the pin (Eurostat serves thousands of each), so a source left in its native unit shows up here as a discrepancy three orders of magnitude wide rather than as a quietly wrong shape.

Refusal

A group is refused when `|discrepancy_frac| > tolerance_relative` AND `|discrepancy| > tolerance_absol` and only where coverage is complete – every unit of the layer reported an absolute area, so the two sides are comparable. Under partial coverage the same difference is the residual pseudo-unit's own target and is not a discrepancy at all (decision T31(a)), so it is reported as `beyond_tolerance` and left to the reader rather than being refused or hidden.

Both tolerances are arguments. Their defaults, 10% and 1,000 ha or head, are decision T31(d), and `build_level_crop_targets()` applies the same rule while the targets are built. The rule is restated here so that a report assembled from stored coverage rows is gated too, and so that a run can be re-scored at a different tolerance without being allocated again.

The production-implied divergence

Production never anchors (decision T31(i)): `t0` is the first observed AREA year and a production row is dropped by the allocator. The information is not therefore worthless, so it is reported here.

Per unit, `production_share` is the unit's share of its group's reported production and `area_share_renorm` is the binding area share on the same denominator: both are renormalised over the **common set**, the units carrying both quantities, because a share over one unit set and a share over another are not comparable. `share_divergence` is their difference, `implied_yield_ratio` their ratio – the unit's yield over the common set's mean yield – and `production_divergence_tvd` on the group is half the summed absolute divergence, the fraction of the crop the two shapes place in different units.

A unit with production and no area share has no yield ratio: the quotient is undefined and is left `NA` rather than reported as `Inf`, while its whole production share stands in `share_divergence`, which is where that case is meant to be read. A common set of fewer than two units supports no comparison at all, and every column is `NA`.

`production_share` comes from reported production values where the group has any (`production_basis` is `"value"`), and otherwise from the producers' own declared shares (`"declared"`), which is what a consented shares-only family ships.

Interior bridges

Decision T31(e) admits a LUH2 proxy bridge across an interior gap of any length, with no refusal threshold, so the length is the only thing that makes a 60-year bridge legible rather than merely legal.

A year of a `(container, item)` series is observed when any unit's row that year is `treatment == "observed"`, and carried otherwise. A run of consecutive carried years is a **bridge** when the series has an observed year both before and after it: the leading run that a pre-seam back-cast produces is not a bridge, and reporting the longer of the two would claim the series was interpolated across years it was never observed in at all. `longest_run` keeps the unrestricted maximum beside `longest_interior_run` so the difference is visible.

The implied cropping intensity

Per unit-year, the whole allocated harvested area over the unit's cropland: `sown_ha / cropland_ha`, flagged where it exceeds the national multi-cropping factor, which is the ceiling a level-0 run would have imposed. Both inputs are supplied by the caller through `intensity` and neither is guessed: with no extent the intensity is `NA`, and with no factor the flag is `NA`. `unit_cropland_extent()` builds the extent from the same layer and cropland the allocation ran on, weighting each cell by `cell_area_frac` exactly as the capacity ceiling does.

See Also

`allocate_level_crops()`, `build_level_crop_targets()`, `unit_cropland_extent()`.

Examples

```
allocation <- list(
  coverage = tibble::tibble(
    year = 2000L, area_code = 1L, item_prod_code = 15L,
    n_units = 2L, n_units_reporting = 2L, coverage = 1,
    basis = "admin_sum", national_total_ha = 250, admin_sum = 225
  ),
```

```
  targets = tibble::tibble(
    year = 2000L, area_code = 1L,
    level_polity_code = c("A1", "A2"), item_prod_code = 15L,
    share = c(0.6, 0.4), target_ha = c(150, 100),
    irrigation_clipped_ha = 0, method_crop_alloc = "admin_area_shares"
  ),
  breach = tibble::tibble(
    year = integer(), area_code = integer(),
    level_polity_code = character(), item_prod_code = integer(),
    mc_basis = character(), in_force = logical(), over_ha = numeric()
  )
)
shares <- tibble::tibble(
  year = 2000L, area_code = 1L,
  level_polity_code = c("A1", "A2"), item_prod_code = 15L,
  indicator_used = "area_harvested", value = c(135, 90),
  treatment = "observed"
)
reconcile_admin_allocation(allocation, shares)$groups
```

`record_provenance` *Record provenance for a reproducible result.*

Description

Capture the information needed to regenerate a result: the package version (the code), the R version, the pinned versions of the input datasets used, and a timestamp. Attach the record to an output with `attach_provenance()` so any number can be traced back to the exact inputs and code that produced it.

Unknown aliases are an error rather than a silent omission, so a provenance record never quietly drops an input it could not resolve.

Usage

```
record_provenance(
  aliases = NULL,
  inputs = whep::whep_inputs,
  recorded_at = Sys.time()
)
```

Arguments

`aliases`	Optional character vector of input aliases to record. When `NULL` (default), every registered input is recorded.
`inputs`	Tibble of registered inputs with `alias` and `version` columns. Defaults to whep_inputs.
`recorded_at`	Timestamp for the record. Defaults to the current time; pass a fixed value for reproducible output.

Value

A tibble with one row per recorded input:

- `recorded_at`: When the record was made.
- `whep_version`: Installed package version (the code).
- `r_version`: R version.
- `input_alias`: Input dataset alias.
- `input_version`: Pinned version of that input.

Examples

```
prov <- record_provenance(
  aliases = "bilateral_trade",
  recorded_at = as.POSIXct("2026-01-01", tz = "UTC")
)
prov
```

`redistribute_feed` *Redistribute available feed supply among livestock demand.*

Description

Matches livestock feed demand to available feed items through a hierarchical allocation that follows the remaining-share principle to avoid exceeding availability. The redistribution path adapts to the `fixed_demand` column in the demand table.

Usage

```
redistribute_feed(feed_demand, feed_avail, options = list())
```

Arguments

`feed_demand`	A tibble of feed demand with columns `year`, `territory`, `sub_territory`, `livestock_category`, `item_cbs_code`, `feed_group`, `feed_quality`, `demand_dm_t`, and a logical `fixed_demand`.
`feed_avail`	A tibble of feed availability with columns `year`, `sub_territory`, `item_cbs_code`, `feed_group`, `feed_quality`, `avail_dm_t`, and `feed_scale`.
`options`	A named list of allocation options. See `.redistribute_feed_options()` for the available entries and their defaults. Supply `grass_availability` (a tibble with `year`, `territory` or `area_code`, and `grass_avail_dm_t`) to bound the otherwise-unlimited pasture grass at that supply per polity-year. The grass deficit then cascades: pasture grass is capped at the ceiling, the deficit is redistributed to leftover non-grass availability in the polity (added as `7_grass_deficit_substitute` intake, limited by that leftover), and the residual stays as biologically-feasible underfeeding (`scaling_factor < 1`). Supply `maintenance_share` (a scalar fraction or

a tibble with `livestock_category` and `maintenance_share`) to also diagnose polities pushed below maintenance; the over-stocked demand rows are attached to the result as the `grass_deficit_diagnosis` attribute. Set `distribute_surplus = FALSE` to suppress the surplus-distribution pass that pushes leftover CBS availability onto variable-demand livestock (correct for historical analyses where the CBS feed element is the realised consumption; keep `TRUE`, the default, for unconstrained scenario projections). Supply `feed_exclusions` (a tibble with `livestock_category` and `item_cbs_code`) to bar those items from those categories at every allocation level, including the substitute pool, the surplus pass, the grass-deficit substitute and the `max_intake_share` reroute. A barred item is offered first to the categories allowed to eat it, then the unrestricted items are shared among everyone. Demand that no allowed item can fill stays unmet (`scaling_factor < 1`) rather than being filled with a barred item. `feed_eligibility` names the rule the exclusions come from and is recorded in the `method_feed_eligibility` column (default `"custom"` when `feed_exclusions` is supplied, `"none"` otherwise).

Value

A tibble of realised intake per demand row, with a `method_feed_eligibility` column naming the eligibility rule applied. When `maintenance_share` is supplied alongside `grass_availability`, a `grass_deficit_diagnosis` attribute lists demand rows underfed below maintenance.

Examples

```
# Two variable-demand categories in one territory competing for a single
# feed item. Availability (150) falls short of demand (200), so the
# remaining-share principle underfeeds both by the same factor rather than
# letting either exceed what is there.
feed_demand <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~livestock_category,
  ~item_cbs_code, ~feed_group, ~feed_quality, ~demand_dm_t, ~fixed_demand,
  2000L, "79", "79", "Cattle_milk",
  NA_integer_, NA_character_, "high_quality", 120, FALSE,
  2000L, "79", "79", "Pigs",
  NA_integer_, NA_character_, "high_quality", 80, FALSE
)

feed_avail <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~item_cbs_code, ~feed_group,
  ~feed_quality, ~avail_dm_t, ~feed_scale,
  2000L, "79", "79", 2514L, "cereals", "high_quality", 150, "national"
)

redistribute_feed(feed_demand, feed_avail)
```

`regime_yield_crop_mapping`

Crop sources for the irrigated:rainfed regime yield ratio

Description

Gives every primary crop production item the two sources of its irrigated:rainfed yield ratio: the SPAM2010 crop whose irrigated and all-rainfed yields set the ratio's level (read with `read_spam_yields()`), and the LPJmL crop functional type whose band yields supply its year-to-year anomaly (read with `read_lpjml_regime_yield()`). Items that neither source classifies get a stand-in, so every item has both sources; each `*_basis` column says how close the stand-in is, and `rationale` says why it was chosen.

The items are every row of cft_mapping and every `"Primary crops"` item of items_prod_full except fallow, which together hold every item that carries harvested area in `build_primary_production()` or in the gridded land use.

Usage

```
regime_yield_crop_mapping
```

Format

A tibble with one row per production item. Columns:

- `item_prod_code`: Integer FAOSTAT item code, the join key.
- `item_prod_name`: Human-readable item name, for reading only.
- `spam_crop`: SPAM2010 v2.0 crop code (one of its 42, e.g. `"whea"`), or several joined by an operator whose meaning `spam_basis` fixes (see the section "Reading `spam_crop`").
- `spam_basis`: How the SPAM crop relates to the item: `"direct"` (it is the item), `"direct_aggregate"` (SPAM itself lists the item in that multi-crop aggregate), `"group_proxy"` (SPAM does not list the item; the aggregate of its crop group stands in), `"proxy"` (a different crop stands in, e.g. grain maize for forage maize), `"composite_weighted"` (an area-weighted mean over several SPAM crops' ratios) or `"product_dominance"` (one of two SPAM crops, chosen per country).
- `lpjml_cft`: LPJmL crop functional type, spelled as in cft_mapping's `cft_lpjml`: one of the twelve crop CFTs or `"others"`. Equal to `cft_lpjml` for every item cft_mapping classifies.
- `lpjml_basis`: As `spam_basis`, for the LPJmL CFT. Items on LPJmL's generic `"others"` stand are `"group_proxy"`.
- `rationale`: Why each non-direct source was chosen. The weakest stand-ins, the fodder crops, are marked `WEAK`.

Reading `spam_crop`

The ratio meant here is the irrigated yield over the all-rainfed yield, each yield being production over harvested area. A single code is that crop's ratio. Joined codes mean one of three things:

- `+` with `spam_basis "direct"`: `"pmil+smil"` (millet) and `"acof+rcof"` (coffee). SPAM splits one FAO item in two by national shares, so the codes are **pooled**: harvested area and production are summed over both codes per regime, and the ratio is computed from the sums.
- `+` with `spam_basis "composite_weighted"`: the forage grasses (`"whea+barl+ocer"`, items 638, 639, 645, 651, 996) and forage legumes (`"bean+chic+cowp+pige+lent+opul+rest"`, items 640, 641, 643). Each code's ratio is computed on its own, and the item's ratio is the **area-weighted mean of those ratios**, weighted by each crop's SPAM2010 harvested area (irrigated plus rainfed) in the country, or by its global harvested area where the country grows none of them.
- `|` with `spam_basis "product_dominance"`: Linum (772) and Hemp (776), `"ooil|ofib"`. WHEP books one harvested area for the seed (SPAM `ooil`) and the fibre (SPAM `ofib`), so **one** of the two is used per country: the aggregate of whichever product has the larger FAOSTAT production there.

Source

SPAM crops from Table S3 of the supplement to Yu, Q. et al. (2020). A cultivated planet in 2010 – Part 2: The global gridded agricultural-production maps. Earth System Science Data 12, 3545-3572. doi:10.5194/essd1235452020. LPJmL crop functional types from cft_mapping. Stand-ins for the remaining items chosen by WHEP.

Examples

```
head(regime_yield_crop_mapping)
```

`regional_mms_distribution` *Regional MMS distribution.*

Description

Fraction of manure managed in each manure-management system (MMS) by region and species, in two selectable halves: the GLEAM 2.0 ingest that the manure engines read by default, and the unsourced placeholder it replaced.

Usage

```
regional_mms_distribution
```

Format

A tibble with `source`, `region`, `species`, `mms_type`, `fraction` and `reference`. `source` is either "gleam_2_0" or "placeholder", and is the value the manure engines' MMS-shares option takes (see manure_engine_options). `reference` cites each row. `fraction` sums to one within every `(source, region, species)`. `region` is the IPCC label vocabulary `.add_ipcc_region()` emits, plus a "Global" fallback.

Source

`source == "gleam_2_0"` is ingested from Tables 4.2-4.11 of Supplement S1 to FAO. 2018. *GLEAM Model description, Version 2.0, Revision 5*, July 2018, data reference year 2010 (workbook https://www.fao.org/fileadmin/user_upload/gleam/docs/GLEAM_2.0_Supplement_S1.xlsx, 219715 bytes, md5 `72fd2ea477dfe8b30cd3657b2baa4af1`, re-downloaded from FAO and verified 2026-09-09; FAO issues no DOI). Those ten tables are the only published regional MMS shares in any GLEAM release: version 3.0 dropped the family, and its Supplement S1 carries MMS *definitions* only. The read and the crosswalk are in `data-raw/livestock_coefficients.R`.

The published grain is finer than this table's on three axes at once, so the ingest is a crosswalk and each of its four choices moves numbers (whep#958). Stated here, and at the point of use in the builder:

1. **Species collapse.** GLEAM publishes one table per production system; this table is keyed on `species_gen`. Systems are averaged **unweighted** within a species and region, counting only the systems the source publishes a column for. A herd- or manure-weighted mean would be better and is **not available from the source**: Supplement S1 publishes herd *parameters* (Tab. 2.4-2.21) and no regional animal numbers or system shares anywhere. Tab. 4.4 (feedlot cattle) is not read: GLEAM models the feedlot as a sub-system of the beef herd, so reading it would give a minority system a third of all cattle manure. The equal-weight collapse is coarsest for poultry, where the flat backyard assumption (50 percent pasture, 50 percent daily spread, identical in all ten regions) gets a third of the weight.
2. **Region collapse.** GLEAM's 10 regions are mapped onto the IPCC labels `.add_ipcc_region()` emits. Only one pair collides – the Russian Federation and Eastern Europe both map to `"Eastern Europe"` – and those two published columns are averaged unweighted.
3. **The `Global` row.** GLEAM publishes no global row, and most WHEP output resolves to `"Global"` (whep#678), so the row carrying the most weight is the one with no published value. It is the unweighted mean over the 10 GLEAM regions of the species-collapsed distributions, counting a region only where the source publishes one, and taken over the 10 published regions rather than the 9 IPCC labels so that the Russian Federation and Eastern Europe keep one vote each. **This row is WHEP-derived, not FAO-published**, and its `reference` says so.
4. **MMS vocabulary.** GLEAM names 14 systems; WHEP's manure chain serves six labels and only six, in all four tables it reads by MMS name. Pasture, daily spread, solid storage, liquid slurry, uncovered anaerobic lagoon and poultry manure with litter map by identity. Drylot maps to `"Solid Storage"`, following IPCC's own combined "solid storage and dry lot" category, which ipcc_2019_n2o_ef_direct carries at the same EF3; a separate `"Dry Lot"` label would give a *lower* methane conversion factor (2.5

against 4.0 percent, Temperate). Pit storage maps to `"Poultry Manure"` for chickens (a layer deep pit is IPCC's poultry manure without litter, held at the same EF3 as the with-litter row) and to `"Liquid/Slurry"` otherwise – **assumed, unverified**, as is intensive-windrow composting to `"Solid Storage"`. Burned for fuel and anaerobic digester are **excluded** and the rest renormalised: neither has a servable WHEP label, both being in climate_mcf only at the unjoinable `climate_zone == "All"` and in `manure_loss_fractions.csv` not at all. Excluding them says the excreted nitrogen passes through the systems that remain, which overstates those systems; it removes 18.5 percent of published buffalo shares (dung burnt for fuel), 5.2 percent of swine and 4.1 percent of cattle, and none of the others.

GLEAM 2.0 publishes no manure-management table for horses, camels or mules and asses. Those four rows are the placeholder's, retained so the engine still resolves a split for them, and flagged unsourced in `reference`.

`source == "placeholder"` is the table WHEP shipped before whep#958: a hardcoded `tribble()` annotated "GLEAM 3.0 / FAO statistics (simplified)", round to the nearest 5 percentage points, matching no table of any GLEAM release (whep#921). It is kept selectable so the values published before the ingest stay reproducible and the sensitivity to the ingest stays measurable. It is **not** a defensible alternative estimate.

This object is result-affecting through `.resolve_mms_shares()`: it weights the Tier 2 manure CH4 methane conversion factor (climate_mcf) and the manure direct-N2O emission factor (ipcc_2019_n2o_ef_direct) at both tiers.

Examples

```
regional_mms_distribution
```

`regions_full` *Full polity and region reference table*

Description

Extended reference table covering all polities and aggregate regions, including countries, territories, and statistical composites that appear in international databases but may lack standard ISO codes.

Usage

```
regions_full
```

Format

A tibble where each row corresponds to one polity or aggregate region. It contains the following columns (same definitions as `polities_cats`, minus the five trailing `0...36`–`0...40` artefact columns):

- `legacy_polity_prefix`: Legacy current polity prefix, kept for compatibility with older code that expected ISO3-like values. **Not a polities code**: it was called `polity_code` until whep#687, where the name promised an identity none of its 271 non-`NA` values holds, so a join to polities or polity_area_crosswalk on it returned nothing. Read `reporting_polity_code` for the polity.
- `polity_name`: Current polity, country, territory, or aggregate name.
- `V1`: Internal row index.
- `code`: Numeric FAOSTAT country/region code.
- `polity_area_code`: Numeric WHEP reporting area code used in matrix workflows.
- `reporting_polity_code`: Current periodized WHEP polity code for `code`.
- `reporting_polity_name`: Current WHEP polity name for `code`.
- `reporting_polity_has_geometry`: Logical flag indicating whether the current reporting polity has a polygon.
- `iso3c`: ISO 3166-1 alpha-3 code (`NA` for aggregates).
- `FAOSTAT_name`: Name used in FAOSTAT (may be `"#N/A"` for aggregates).
- `EU27`: Logical EU27 membership flag.
- `name`: Name used in external databases.
- `eia`: EIA country identifier.
- `iea`: IEA country identifier.
- `water_code`: Water statistics numeric code.
- `water_area`: Name used in water statistics.
- `baci`: BACI trade database country code.
- `fish`: Fisheries dataset numeric code.
- `region_code`: Numeric regional code.
- `cbs`: Logical CBS dataset membership flag; `TRUE` if the area has a commodity balance sheet of its own. 202 areas.
- `fabio_code`: FABIO database numeric code, and the value `polity_area_code` is derived from, so it is the fold instruction as well as a fact about FABIO. It is the area's own `code` for a `cbs` reporter and 999 (Rest of World) otherwise, with seven exceptions: 62 -> 238, 276 -> 206 and 277 -> 206 are successor-state folds, and 153, 154, 209 and 212 are `cbs` reporters folded into 999 anyway. Those four are a contradiction inside this table – FABIO's own published region list enumerates all four as regions of their own – left standing because correcting it would move published values. See folded_reporting_areas and issue 556.
- `ADB_Region`: Asian Development Bank region.
- `region`: General world region.
- `uISO3c`: UN M49 numeric code.
- `Lassaletta`: Lassaletta et al. nitrogen study grouping.
- `region_krausmann`: Krausmann regional grouping.
- `region_HANPP`: HANPP study regional grouping.

- `region_krausmann2`: Alternative Krausmann grouping.
- `region_UN_sub`: UN M49 sub-region.
- `region_UN`: UN M49 macro-region.
- `region_ILO1`: ILO primary region.
- `region_ILO2`: ILO secondary region.
- `region_ILO3`: ILO tertiary region.
- `region_IEA`: IEA region.
- `region_IPCC`: IPCC region.
- `region_labour`: Labour-focused region.
- `region_labour_agg`: Aggregated labour region, one of `"SAA"`, `"LACA"`, `"Europe"`, `"AUS"`, `"SE-Asia"`, `"MENA"`, `"FSU"`, `"NAME"` or `"RoW"`.
- `region_labour_mech`: Labour mechanisation region, `"mech"` or `"no_mech"`. Three cells held the row's UN M49 region name instead of a class, in the source spreadsheet too: Angola (code 7) `region_labour_mech` `"Middle Africa"`, and Northern Mariana Islands (163) `"Micronesia"` in both `region_labour_agg` and `region_labour_mech`. They were filled with the class every other member of that region carries: Angola `"no_mech"` (the other eight Middle Africa countries), Northern Mariana Islands `"mech"` and `"AUS"` (the other Micronesia rows, all `"mech"` and all `"AUS"` except the US Minor Outlying Islands, `"NAME"`). The column is not strictly a function of region – `"Pacific"`, `"FSU"` and `"South America - South Cone"` each split across both classes – so this is a region-analogy fill, not a recovered source value (whep#855). Nothing in the package reads either column.

Which regional groupings WHEP reads

The grouping columns are not all inputs to this package. Six have a consumer in the tree, measured over `R/`, `data-raw/`, `tests/`, `vignettes/` and `inst/`: `region` (carried into polity_area_crosswalk at build time), `region_krausmann` (the residue recovery rate, and the IPCC excreta regions of `prepare_spatialize_all.R`), `region_HANPP` (the modern-variety adoption share), `region_UN_sub` (the residue feed-use fraction, since whep#405), `ADB_Region` (build_primary_production area keys) and `EU27` (the EU aggregate of the FABIO comparison). `region_code` carries no information `region` does not – the two are a 1:1 relabelling – so it needs no consumer of its own.

The rest – `Lassaletta`, `region_krausmann2`, `region_UN`, `region_ILO1`, `region_ILO2`, `region_ILO3`, `region_IEA`, `region_IPCC`, `region_labour`, `region_labour_agg`, `region_labour_mech` – are published third-party taxonomies shipped for downstream analysis and read by nothing in the package. They are shipped as reference, and carry no promise of being re-validated against their upstream taxonomy on release, so a consumer should check the gap it inherits before keying anything by one (whep#386).

The gap the present-day taxonomies share is dissolved states. Over the 202 `cbs` reporters, `region_ILO1`, `region_ILO2`, `region_ILO3`, `region_IEA` and `region_IPCC` are each `NA` for exactly the four federations WHEP still books commodity balances for – Czechoslovakia (51), Serbia and Montenegro (186), the USSR (228) and the Yugoslav SFR (248) – and complete everywhere else; `region_UN` labels three of the four and leaves only Czechoslovakia `NA`. `ROW` (999) carries an explicit `"RoW"` value in all of them rather than `NA`. Grouping by one

of these without deciding what to do with the federations silently drops the pre-succession record. `region_UN_sub`, which shares the gap and does have a consumer, is pinned against it in `test_region_classifications.R`.

A `region_test` column with two values (`"Europe"`, `"Other"`) and no consumer was dropped in whep#386.

Source

Compiled from FAOSTAT, UN M49, ILO, IEA, and other international statistical sources.

See Also

polities_cats for the subset restricted to sovereign countries.

Examples

```
head(regions_full)
```

`residue_humification` *Humification fraction by carbon input type.*

Description

Per-input-type fraction of soil carbon input that is stabilised directly into humus (rather than entering the labile, fresh pool). The HSOC turnover model takes a scalar humification fraction; this table supplies the type-specific values so a carbon-input builder can compute a carbon-weighted effective humification fraction per cell-year before running the SOC model. Other models (RothC, ICBM, AMG, Century) use their native carbon partition and ignore this table; AMG instead uses `amg_h_by_input_type`. Values are transcribed from the biomass coefficient table of an earlier regional historical reconstruction, its `Residue_humified_kgC_kgC` column (manure and roots from the corresponding manure and `Root_humified_kgC_kgC` entries).

Usage

```
residue_humification
```

Format

A tibble with columns:

input_type Canonical carbon input type: `"crop_residue"`, `"root"`, `"weed"`, `"woody_residue"`, `"manure"`, `"excreta"`, `"urban"`, `"urban_compost"`, `"compost"`, `"green_manure"`.

humified_fraction Fraction of input carbon stabilised directly into humus (kg humified carbon per kg carbon input).

description Human-readable description of the input type and its provenance in the source coefficient table.

Source

Biomass coefficient set of an earlier regional historical reconstruction (its `Residue_humified_kgC_kgC` and `Root_humified_kgC_kgC` columns), itself compiled from soil organic carbon turnover and litter humification studies including Andren, O. & Katterer, T. (1997). doi:10.1890/10510761(1997)007[1226:ITICBM]2.0.CO;2; Katterer, T., Bolinder, M. A., Andren, O., Kirchmann, H. & Menichetti, L. (2011). Roots contribute more to refractory soil organic matter than above- ground crop residues, as revealed by a long-term field experiment. *Agriculture, Ecosystems & Environment*, 141(1-2), 184-192. doi:10.1016/j.agee.2011.02.029.

Examples

```
residue_humification
```

`resolve_admin_shares` *Resolve one source per container from the admin-shares table*

Description

Choose, for every `(area_code, level, item_prod_code, year)` the admin-shares table covers, the single source whose reported units set the within-country shape, and record why it won. Operates on **observed** rows only: gap filling is a separate step that runs after resolution, so an `interpolated` or `carried` row here is an error, not an input.

The result is a list of five tibbles rather than one table with attributes, because every piece is itself evidence a later task reads, and an attribute does not survive a write to parquet or CSV.

Usage

```
resolve_admin_shares(
  shares,
  overrides = NULL,
  constraint_exclude = NULL,
  not_shipped = character()
)
```

Arguments

`shares`	Admin-shares table, conforming to `admin_shares_schema()` except that `source` joins the key: a multi-source union is exactly what resolution consumes, and the contract's key holds within one source. Every row must have `treatment_year == "observed"` and must carry `value`, `share` or both; a row with neither aborts with class `whep_error_admin_no_measure`, and a non-finite `value` or `share` with class `whep_error_admin_nonfinite`.
`overrides`	Optional tibble of forced choices, with columns `area_code`, `source` and optionally `item_prod_code` (`NA`, or the column absent, means every item of that container). An item-specific row beats a container-wide one. A row that never forces a winner warns.

`constraint_exclude`
: Optional named list of years to withhold, keyed by `area_code` as a character name, such as `list("840" = 1961:1989)`. Those container-years leave the constraint set and are returned in `excluded`, never silently dropped. This is the leave-years-out validation switch.

`not_shipped`
: Character vector of `source` labels whose family is withheld from the pin board (plan T23/T24); the coverage report flags every resolved row whose source is one of them. Empty by default.

Value

A list of five tibbles:

- `shares`: the winning rows, each exactly as supplied plus `resolved_source`, `resolved_tier`, `resolved_grain` and `resolution_rule` (one of `"override"`, `"grain"`, `"tier"`, `"run_length"`, `"source_name"`, `"single_candidate"`).
- `seams`: the seam list described above.
- `coverage`: the coverage report described above.
- `excluded`: the rows `constraint_exclude` withheld, in the input's shape.
- `dropped`: the losing rows, in the input's shape plus `drop_reason` (`"indicator_never_binds"`, `"indicator_precedence"`, `"override"`, `"grain"`, `"tier"`, `"run_length"`, `"source_name"`).

Every input row appears in exactly one of `shares`, `excluded` and `dropped`.

Precedence

Within one candidate group – `(area_code, level, item_prod_code, indicator_used, year)` – each source is one candidate. Candidates are ranked, best first, by:

1. **Override**: a source named for this container in `overrides` wins wherever it is a candidate, and every group it reaches is recorded `resolution_rule = "override"` – including a group where the named source had no rival, so an override's footprint can be read off the coverage report. This is the per-country escape hatch the grain rule requires: an override may keep a coarser source over a finer one.
2. **Grain**, finer first: `"admin1" < "admin2" < "admin3"`, compared through `match()` on that vocabulary, exactly as `admin_shares_schema()` declares it. Grain beats tier, under decision 4 as amended at lock: public first at equal or finer grain, in-house where no public source of equal grain exists. So Spain keeps its 53 NUTS-3 provinces (tier 2, `"admin2"`) over Eurostat's NUTS-2 (tier 1, `"admin1"`). A candidate reporting at several grains at once is ranked on its **coarsest**, and warns.
3. **Tier**, lower first.
4. **Run length**, longer first: the contiguous run of that source around that year, defined below.
5. **Source name**, ascending in the C locale, with a warning naming the tied sources. This is the deterministic last resort, never a scientific rule.

None of the five reads `value`, so a source consented to ship derived shares only – the Latin American panel of the *Shares-only rows* section of `admin_shares_schema()` – competes, wins and is carried on exactly the same terms as a source shipping absolute areas. Its rows arrive in `shares` still carrying `value = NA`; nothing here fills that in, and nothing downstream may.

Before any of that, the indicator in force is chosen per `(area_code, level, item_prod_code, year)` by decision 8's order of acceptance – `"area_harvested"`, `"area_planted_or_sown"`, `"area_main"`, `"area_cultivated"`, then `"production"` only where no area exists. Rows of a losing indicator go to `dropped` with `drop_reason = "indicator_precedence"`. `"yield"` is in the table's vocabulary but never binds an allocation (a share of a ratio is not a share), so a `"yield"` row is dropped as `"indicator_never_binds"`.

Contiguous run

The run length of source `s` at year `y` is the number of years in the maximal block of **consecutive** years containing `y` in which `s` supplies at least one row for the same `(area_code, level, item_prod_code, ir` – the candidate group's columns except `year`. So a source present in 1990:2000 has run length 11 at every one of those years; if it is absent in 1997, its run length is 7 at 1996 and 3 at 1998. The run is measured on the rows that actually reach ranking, after `constraint_exclude` and after indicator precedence: a long run of production years says nothing about the continuity of an area series, and a year withheld for validation is not evidence of coverage.

Seams

One row per `(area_code, level, item_prod_code)` series, seam year and `seam_kind`:

- `"start"`: the series' first resolved year. `value_to` is the source it starts on.
- `"source_switch"`, `"grain_switch"`, `"nuts_version_switch"`, `"indicator_switch"`: the resolved value differs from the previous **resolved** year of that series, which is the previous year present, not necessarily `year - 1`.
- `"coverage_change"`: the set of reporting units differs from the previous resolved year – a unit entering or leaving reporting.

A missing value and a present one count as a change; two missing values do not.

Coverage report

One row per `(area_code, level, item_prod_code, year)`, carrying the resolved source, tier, grain, indicator, NUTS version and rule, the number of reporting units, the units themselves as a `"|"`-joined string of `level_polity_code` sorted in the C locale (a string, not a list-column, so the table writes to parquet and CSV unchanged; unresolved units appear as `"<NA>"` and count as one member), and `coverage_change`, and `not_shipped`, which marks resolved rows whose source family is withheld from the pin board (T23).

Examples

```
# Spain 2010: Eurostat NUTS-2 (tier 1, admin1) against the in-house
# NUTS-3 provinces (tier 2, admin2). Grain beats tier, so tier 2 wins.
rows <- tibble::tibble(
```

```
  area_code = 724L,
  level_polity_code = c("ES-N2-A", "ES-N2-B", "ES-N3-1", "ES-N3-2"),
  level = 1L,
  item_prod_code = 15L,
  indicator_used = "area_harvested",
  year = 2010L,
  value = c(100, 200, 120, 180),
  share = c(1 / 3, 2 / 3, 0.4, 0.6),
  source = rep(c("Eurostat_apro_cpshr", "ES_provinces"), each = 2),
  tier = rep(c(1L, 2L), each = 2),
  grain = rep(c("admin1", "admin2"), each = 2),
  concept_break = FALSE,
  nuts_version = rep(c("2021", "2016"), each = 2),
  source_native_id = NA_character_,
  source_native_name = NA_character_,
  source_id = rep(c("Eurostat_apro_cpshr", "ES_provinces"), each = 2),
  source_version = NA_character_,
  recorded_at = "2026-01-01T00:00:00Z",
  treatment_year = "observed",
  value_flag = NA_character_
)
resolved <- resolve_admin_shares(rows)
resolved$coverage
resolved$dropped
```

`resolve_admin_units` *Resolve source-native admin identifiers to polity codes*

Description

Attach `level_polity_code` to administrative-statistics rows by resolving each row's **source-native identifier** – not its name – through `resolve_polity_label()`, under a source slug naming the code system the identifier belongs to and, for NUTS geographies, its nomenclature version. Rows that resolve to nothing keep `NA` and are counted in the returned diagnostics table rather than being dropped.

This is the load-time step the admin-shares design puts between the readers, which emit source-native keys only, and every consumer that needs a polity. Resolving here rather than at pin-build time is why the `admin-shares` pin does not need a staleness warning; see below.

Usage

```
resolve_admin_units(x, code_system, year_col = "year", aliases = NULL)
```

Arguments

`x`	Administrative-statistics rows, in the shape the readers emit: a data frame carrying `source`, `source_native_unit_id`, the year column named by `year_col`, and `nuts_version` where `code_system` is a NUTS system.

source_native_unit_name is never read. The admin-shares contract (admin_shares_schema()) calls the same identifier source_native_id, so a caller resolving that table renames the column first. An existing level_polity_code is overwritten: this function is the authority on it.

code_system — Code system the identifiers belong to, one of the slugs in the table above. Length 1, or one value per row of x.

year_col — Name of the year column in x. Defaults to "year".

aliases — Optional alias table in the polity_label_aliases schema, used **instead of** the package data. NULL, the default and what production calls use, resolves against the published polity_label_aliases through the ALIAS ROUTE ONLY – never resolve_polity_label() itself, whose name and ISO3 identity routes would let an administrative identifier that happens to collide with a polity name or ISO3 code resolve to the wrong thing (usually the container). An injected table takes the same alias-only route, with the same source and year scoping, so the two paths agree on everything but which table they read.

Value

A list of two tibbles:

- rows: x with alias_source (the slug each row was resolved under, NA where none could be built) and level_polity_code (the resolved polity, NA where nothing matched).
- diagnostics: one row per (source, alias_source) with n_rows, n_unresolved and example_ids – up to five distinct unresolved identifiers, "|"-joined in the C locale, NA when the group resolved fully. Ordered by n_unresolved, descending. A row with no native identifier at all – a derived residual, in admin_shares_schema()'s terms – cannot resolve and is counted here like any other unresolved row, contributing no example id.

Alias source slugs

code_system names the identifier's code system; the slug passed to resolve_polity_label() is built per row from it:

- "usda-nass-fips": the state FIPS code read_admin_stats_nass() emits. Slug as given.
- "eurostat-nuts": the NUTS code read_admin_stats_eurostat() emits. Slug "eurostat-nuts<nuts_vers
e.g. "eurostat-nuts2016".
- "jrc-nuts": the NUTS code of the JRC subnational release (inst/extdata/jrc_subnational_source_mani
coded on NUTS 2016. Slug "jrc-nuts<nuts_version>".
- "ibge-uf": the UF code read_admin_stats_sidra() emits. Slug as given.
- "whep-lab-<family>": the compilation's own admin_unit_id, as read_admin_family() emits it. Slug as given.

The five "whep-lab-" slugs are the five tier-2/3 families read_admin_family() reads, with the "admin-stats-" prefix replaced: "whep-lab-japan", "whep-lab-spain-provinces", "whep-lab-australia", "whep-lab-france-livestock" and "whep-lab-latam". Their

identifiers are the compilation's own (`"JPN-AICHI"`, `"ESP-ES111"`, `"FRA-FR102"`), which is why each family is its own code system rather than a shared one.

Identifiers are normalised exactly as `resolve_polity_label()` normalises a label, on both sides of the comparison, so the injected and published routes agree on what a key is.

Alias rows this needs from whep-polities

The alias rows are a whep-polities deliverable and are **not published yet**: on the 2026-09-03 snapshot, polity_label_aliases holds 1,007 rows scoped to 15 sources (`crops-manure-n`, `fao`, `fao1952`, `faostat`, `federico_tena`, `iia`, `iia-cotton`, `iia-tea`, `juan`, `lassaletta-grassland-share`, `mitchell`, `mueller-synthetic-n`, `sa_colonial`, `trade-sources`, `whep-split-2026-06-29`) and none of the code systems above. Until they land, `aliases = NULL` resolves every administrative unit to `NA` – visibly, and counted – which is the honest state, and the tests inject the rows they need.

One example row per slug, for the fixture countries, in the polity_label_aliases schema (`source_label`, `source`, `year_start`, `year_end`, `polity_code`): `"19"` under `"usda-nass-fips"` for the Iowa state polity; `"FR21"` under `"eurostat-nuts2013"` and `"FRF2"` under `"eurostat-nuts2016"`, both for the one Champagne-Ardenne polity the recoding renames; `"FRF2"` under `"jrc-nuts2016"` for the same one; `"35"` under `"ibge-uf"` for the Sao Paulo state polity; `"ESP-ES111"` under `"whep-lab-spain-provinces"` for the NUTS-3 province `ES111`; and `"JPN-AICHI"` under `"whep-lab-japan"` for `JPN-23-1871-2025`, the one target polity that already exists in the shipped polities.

Why there is no staleness warning

The `admin-shares` pin is keyed on source-native identifiers and carries no resolved polity code, so nothing in it can disagree with a newer polities snapshot: the resolution is redone here on every load. A pin that froze resolved codes would need a `.warn_stale_admin_shares()` guard to say which snapshot it was resolved against; this one has nothing to warn about, which is why the plan chose source-native pinning.

Examples

```
# Two NUTS vintages of one code, resolved under their own slugs. The
# alias rows are injected, and their polity code is illustrative: the
# real rows are the whep-polities deliverable described above.
rows <- tibble::tibble(
  source = "Eurostat_apro_cpnhr_h",
  source_native_unit_id = c("FR21", "FRF2", "FR83"),
  source_native_unit_name = c("Champagne-Ardenne", "idem", "Corse"),
  nuts_version = c("2013", "2016", "2013"),
  year = c(1995L, 1995L, 1995L)
)
aliases <- tibble::tibble(
  source_label = c("FR21", "FRF2"),
  source = c("eurostat-nuts2013", "eurostat-nuts2016"),
  year_start = NA_integer_,
  year_end = NA_integer_,
  polity_code = "FR-CHAMPAGNE-ARDENNE",
  common_name = "Champagne-Ardenne",
```

```
  confidence = "high",
  observed_rows = NA_integer_
)
resolved <- resolve_admin_units(rows, "eurostat-nuts", aliases = aliases)
resolved$rows[, c("alias_source", "level_polity_code")]
resolved$diagnostics
```

`resolve_polity_label` *Resolve a source's country label to a polity*

Description

Maps a country or area **label**, as a source writes it, to a WHEP polity code. This complements `add_polity_code()`, which resolves numeric FAOSTAT/FABIO area codes: before this existed there was no supported path from a label to a polity, so datasets carrying labels went unresolved. mueller_synthetic_n's `iso3c` column holds FAO-style legacy codes (`"BZE"` for Belize, `"ROM"` for Romania, `"ZAR"` for Zaire) and lassaletta_grassland_share's `Country` holds name variants (`"Cape Verde"`, `"Swaziland"`), none of which resolve against polities directly.

Usage

```
resolve_polity_label(
  label,
  source = NULL,
  year = NULL,
  item = NULL,
  country = NULL,
  back_cast = TRUE,
  unit = NULL,
  indicator = NULL
)
```

Arguments

`label`	Character vector of source labels.
`source`	Optional source slug (e.g. `"lassaletta-grassland-share"`). Length 1, or the same length as `label`. On the alias route `NULL` matches unscoped aliases only – 188 of 1,003 – so a `NULL` source narrows that route sharply; the identity routes then get their turn, subject to the guards above.
`year`	Optional integer vector of years. Length 1, or the same length as `label`. On the alias route `NULL` matches aliases with no year scope only, which is the 14 of 1,003 published aliases carrying NEITHER bound. The name and ISO3 routes can still answer without a year, but only for an identifier exactly one polity has ever carried, so supplying a year remains much the stronger question: it is what lets a label resolve to the right *period* rather than to nothing.

`item` Optional item names as the source writes them. Length 1, or the same length as `label`. Only used to apply polity_label_item_corrections, which needs `source` and `year` too.

`country` Optional ISO3 code of the country each label belongs to. Length 1, or the same length as `label`. Restricts the name and ISO3 routes to that country's polities.

`back_cast` Logical. `TRUE` (the default) keeps aliases upstream marks as reconstructions (`disposition == "back_cast"`); `FALSE` drops them.

`unit`, `indicator`
Optional unit and indicator of each row, as the source writes them (e.g. `"tonnes"`, `"ha"`). Length 1, or the same length as `label`. Used to match polity_label_item_corrections rules scoped on them, and `indicator` also to choose between polity_label_aliases rules scoped on it; required for the rows such a rule would otherwise match.

Details

The mapping is polity_label_aliases, a copy of the map published by `whep-polities`. It is deliberately NOT computed here: a label's meaning is a fact about the source, upstream already decides it, and a second lookup in this package would be a second authority for the same question.

Resolution is **source- and year-aware**, and both matter:

- An alias may be scoped to one `source`, because the same label can mean different things in different sources. A scoped alias never applies to another source; an unscoped one applies to any.
- An alias may be scoped to a year range, because a label's referent changes. `"Cape Verde"` in 1970 is the Portuguese colony `CPV-1886-1975`; in 1990 it is `CPV-1975-2025`.

Where several aliases match, the most specific wins: year-scoped over unscoped, then source-scoped, then the narrower year range. That ordering mirrors `matchlib.Matcher.match_alias` upstream, so both sides agree.

Where no alias applies, a second route tries the polity's own `polity_name` and then, for a three-letter label, its `iso3_code`. That mirrors upstream's "alias, then ISO/name family + year containment", and both halves are needed. Without the name half a caller passing the database's own name for a polity got `NA`: `resolve_polity_label("Netherlands")` found nothing while polities carried a polity named exactly that. Without the ISO3 half the map answers only for labels a curator had to decide about, which is 380 of mueller_synthetic_n's 5,043 rows – the 11 legacy codes – against all 5,043 with it, asked at `year = 2000`. Asking without a year resolves only 1,255, because the guard below then refuses every identifier more than one live polity has ever carried. Two guards bound both halves.

- An identifier resolves only when **exactly one** polity carries it in the year asked about, because otherwise row order would decide and `NA` is the honest answer. Sharing an identifier is common in the shipped polities snapshot: of its 726 live rows, 110 normalised names and 133 ISO3 codes are carried by more than one polity. A year separates nearly all of them – no two live polities sharing a normalised name cover a common year – but not the ISO3 index, where 69 pairs still do, 62 of them naming

different territories rather than successive periods of one. `"PAN"` in 1970 is the case that matters: `PAN-1903-1979` and the Canal Zone `CZN-1903-1979` both carry that ISO3 then – a real territorial overlap no re-sync removes – so the answer is `NA`, while `"PAN"` in 2000 resolves to `PAN-1979-2025`.

- An alias covering that year outranks both, whatever its source, and a label naming an area the crosswalk leaves unmapped is refused outright.

Returns `NA` when neither route resolves, which is a real answer rather than a failure. Some labels are aggregates a source keeps reporting after the territory stopped existing – `"FSU"` runs to 2009 though nothing has held that territory since 1991 – and those years are deliberately unmapped rather than routed to a polity that had ended.

Every resolved code is one `get_polity_geometries()` can return a row for, and that is an invariant rather than a happy accident: polity_label_aliases and polities are regenerated together from a single upstream revision, and `data-raw/table_mappings.R` aborts the build if any alias names a polity the shipped table does not carry. A dangling resolution therefore cannot ship.

Three more inputs follow contracts `whep-polities` publishes beside the alias map:

- `item`**: rows a source files under another territory's label for one item.** An alias has no item dimension, so it cannot say that Mitchell's pre-1910 `"south africa"` sugar cane is Natal's while every other `"south africa"` item of those years is the Cape's. polity_label_item_corrections states those cases. When `source`, the label, `item` and a year inside the rule's inclusive range all match, the label is replaced by the rule's `correct_label` BEFORE any route below runs, so the usual alias and year rules then place it. Rules do not chain: each is tested against the label the caller passed. A row without a year is never corrected. A corrected row ignores `country`, which named the territory it was misfiled under, and its new label is not read as an ISO3 code, so only the corrected label decides. A rule whose `polity_code` is `"UNROUTED"` marks rows that belong to no polity (a wrong territory with no right one to land on), and those resolve to `NA`. A rule may also be scoped on `unit` or `indicator`, where only that separates the rows: Mitchell's 1955-1960 `"viet nam"` rice output in tonnes is North plus South Vietnam, its area in hectares South only. A scoped rule applies only when the caller's `unit` / `indicator` equals it; a row it would otherwise match but whose scoped value is missing is an error of class `whep_error_unscoped_label_item_correction`, not a silent miss.
- `country`**: the reporting country, as an ISO3 code.** The name route compares normalised names, and normalisation drops parenthesised qualifiers, so a bare subnational name meets another country's unit: `"Santa Cruz (department of Bolivia)"` normalises to `"santa cruz"`, and so does Argentina's province. Given `country`, the name and ISO3 routes only consider polities whose `iso3_code` (its code prefix where that is missing) is that country. Without it, a name that live polities of two or more countries carry at the same time – counting a subnational polity's parenthesised qualifier as a name, so Mexico's `"Ciudad de México (Distrito Federal)"` shares `"distrito federal"` with Brazil's – is refused with a warning of class `whep_warn_ambiguous_polity_name` rather than guessed. The refusal holds in every year, not only in years two such polities overlap, because a source can report a unit before its own polity begins: the subnational panel reports Argentina's Santa Cruz from 1900, and with no Argentine polity until 1955 the year alone sent it to Bolivia. The alias route is not restricted: an alias names its polity explicitly.

- `back_cast`**: whether to accept reconstructions.** An alias whose `disposition` is `"back_cast"` routes years a source reconstructs onto a boundary that did not exist yet to the modern polity, which may begin after those years by design (`BRA-TO-1988-2025` receives the panel's 1900-1987 Tocantins series). `back_cast = FALSE` drops those aliases, for a caller that wants observation only.
- `indicator`**: aliases split per indicator.** One panel unit id can name two territories: whep-polities #703 found `CHL-LL`'s crops reported for Los Lagos plus Los Ríos in every year, while its landuse and livestock are Los Lagos alone. The alias map's optional `indicator` column scopes a rule to rows carrying that indicator (`NA` means any). Where a label, source and year carry scoped rules, only the rule for the row's `indicator` applies; an indicator the split leaves out resolves to `NA`, never to the name route, which would put the rows back on the post-split polity; and a row that gives no `indicator` is an error of class `whep_error_unscoped_indicator_alias`. Upstream allows the scope only on the subnational panel's slugs (`"juan-subnational"`, `"whep-lab-*"`), so callers resolving that panel must pass its `indicator` column verbatim.

Value

A character vector of polity codes, `NA` where nothing matched. On the identity routes a subnational polity contained in another candidate (per polity_containment) never competes with its container, so an ISO3 shared by a state and its provinces resolves to the state.

See Also

`add_polity_code()` for numeric area codes.

Examples

```
resolve_polity_label("ZAR", source = "mueller-synthetic-n", year = 2000)
resolve_polity_label(
  c("Cape Verde", "Cape Verde"),
  source = "lassaletta-grassland-share",
  year = c(1970L, 1990L)
)
```

`resolve_polity_lineage` *Resolve a national row onto the polity its grid support carries that year*

Description

The national tables and the cell support are keyed on **different polity vintages**. A 1961 harvested-area row is keyed `area_code` 185, whose reporting polity is `RUS-1991-2014`; a year-aware read of the polycell support holds `F228-1945-1991` (the USSR) in 1961 and no Russian polity at all. The row then matches no cell and `build_gridded_landuse()`'s missing-reporter guard drops its whole national total.

This is the **national-side lineage step**: for every `(area_code, year)` it walks the `predecessor` edges of polities from the row's reporting polity back to the polity the supplied `support` actually carries in that year, and records which rule answered. It moves no value and allocates nothing – splitting a predecessor's cells between its successors is the consumer's job, and this function only says **which polity's cells** a row belongs on.

Measured on the live `polycell_support` pin (`20260907T111653Z-e654d`) and the `spatialize-country-areas` pin, over the 1961 crop tables: 27 of 194 reporting areas carry 164.2 Mha of harvested area (17.2% of the world) that a year-aware support cannot place. Under `basis = "historical_polity"` the lineage places 21 of them, 156.5 Mha, leaving a 7.7 Mha (0.81%) residue that is **not** a lineage failure: 0.13 Mha is four Rest-of-region reporting buckets, which are aggregate polities the support excludes by construction under either vintage, and 7.6 Mha is Viet Nam and Yemen. Their reporting polities are the aggregates `F237-1954-1975` and `F249-1918-1990`, which the support carries no cell for and which no polity names as its `predecessor`, so no walk reaches the polities that do hold the ground (North and South Vietnam; the Mutawakkilite Kingdom and Aden) and this function leaves them unresolved. The level-0 grid places them instead from the recorded rows of `polity_cell_support_map.csv`, which key those polities' cells on 237 and 249 and put Yemen on its modern cells in 1962-1989 (whep#1317).

Usage

```
resolve_polity_lineage(
  national,
  support,
  basis = c("historical_polity", "constant_territory"),
  polities = whep::polities
)
```

Arguments

`national`	A tibble of national rows with at least `area_code` and `year`. Any other column is carried through untouched; no row is added, dropped or reordered.
`support`	A tibble of the polity intervals the spatial support actually holds cells for, with `polity_code`, `start_year` and `end_year`. `start_year` is inclusive and `end_year` exclusive, the polities convention. `read_polycell_support()` returns a superset of these columns.
`basis`	Which polity vintage binds a row, `"historical_polity"` (default) or `"constant_territory"`. See *Which vintage binds is a choice, not a fact.*
`polities`	The polity table carrying the `predecessor` edges the walk follows; defaults to polities. Supplied as an argument so a caller can pin a vintage rather than inherit whichever one the package ships.

Value

`national` with three columns added: `lineage_polity_code`, the polity whose cells the row belongs on, `lineage_polity_name`, and `method_polity_lineage`, one of `"anchor"` (the reporting polity is itself carried at that year), `"predecessor"` (a predecessor is),

`"sibling_interval"` (a different interval of the polity the walk landed on is – the support and `polities` disagree about the interval; an interval that succeeded the polity the walk reached is a different country and is never taken), `"aggregate"` (an aggregate reporting union whose `successor` list names a polity on the walk is, e.g. Belgium-Luxembourg for Belgium before 2000), `"constant_territory"`, or `"unresolved"`. An unresolved row keeps `NA` rather than being dropped, so the gap stays visible, and is warned about with condition class `whep_lineage_unresolved`.

Which vintage binds is a choice, not a fact

Two bases are defensible and they differ numerically, so `basis` selects between them and the answer is stamped on every row in `method_polity_lineage`. They are alternatives, never a silent fallback.

`"historical_polity"` The default and the more rigorous. A row is understood as a share of the entity that reported in that year – the FAOSTAT back-series is a modern-territory reconstruction of it – and is placed on that entity's cells. Conserves every national total against a year-aware support; leaves the successor's share of the predecessor's cells for the consumer to decide.

`"constant_territory"` The row keeps its reporting polity, i.e. modern borders wherever a territory changed. Simpler, and what a present-day snapshot support does implicitly, but it keeps the attribution error at every succession: 164.2 Mha of 1961 harvested area has no cell to land on.

The support is an input, and its supply is asserted

`support` is required and is checked for rows covering every year in `national`, because a lineage resolved against polities alone can be confidently wrong. `polities` carries two overlapping Yugoslav intervals, `F248-1920-1991` and `F248-1947-1991`, and the published support emitted only the second: a walk that consulted the edges alone would answer `F248-1920-1991`, satisfy every totals check, and still find no cell. An edgeless `polities` is refused for the same reason – it would return every anchor unchanged and reconcile perfectly.

Examples

```
national <- tibble::tribble(
  ~area_code, ~year, ~harvested_area_ha,
         185, 1961L,           85049155,
         185, 2015L,           45000000
)
support <- tibble::tribble(
  ~polity_code, ~start_year, ~end_year,
  "F228-1945-1991",      1945L,     1991L,
  "RUS-2014-2025",       2014L,     2025L
)
resolve_polity_lineage(national, support)
```

`row_evidence` *Produce row-level evidence for a table.*

Description

Record, as data, what each row of a result rests on: which producer claimed it, at which version of that producer's source, and the documented evidence fields the producer carries per row (a FAOSTAT flag, a `method_*` label, a gap-filling status). This is the row-level counterpart of `record_provenance()`, which records the code and input versions behind a whole dataset (whep#372).

The record is a **separate table**, not an attribute. Attributes are silently dropped by `dplyr` joins, filters and `summarise()`, so attribute-borne evidence cannot survive the composition it is meant to document. A keyed sidecar can: `evidence_for()` re-aligns it onto a table after a join, `combine_row_evidence()` merges the output of several producers without letting one overwrite another, and `evidence_conflicts()` reports where two producers disagree.

Usage

```
row_evidence(
  data,
  source_id,
  key,
  fields,
  source_version = NA_character_,
  recorded_at = Sys.time()
)
```

Arguments

`data`	Table the evidence describes. Not modified. May have zero rows, which yields a zero-row evidence table.
`source_id`	Immutable identifier of the producer, one non-empty string.
`key`	Character vector of column names of `data` that jointly identify a row. Must be unique-valued and free of `NA`, otherwise a piece of evidence would address more than one row or none.
`fields`	The row-level evidence to record, either a character vector of column names of `data`, or a named list whose elements are vectors of length 1 or `nrow(data)`. A list column cannot be recorded.
`source_version`	Version or vintage of the source, one string or `NA`.
`recorded_at`	Timestamp of the record, a length-one `POSIXct`. Defaults to the current time; pass a fixed value for reproducible output.

Value

A tibble of row evidence, as described above.

Evidence table

Format `"whep-row-evidence/1"`. One row per (table row × evidence field), every column `character` so the table round-trips through Parquet, CSV and YAML unchanged:

- `row_key`: The row's identity, i.e. its key values joined by `U+001F` (unit separator). Opaque: build and read it with this family's functions rather than by hand.
- `key_columns`: The key column names, comma-separated, so a consumer can re-derive `row_key` from the data alone.
- `source_id`: The producer's immutable identifier. Callers should use the package's existing dataset labels (`"FAOSTAT_prod"`, `"FAOSTAT_FBS_New"`, `"LUH2"`), because nothing downstream can recover an identity that was renamed between builds.
- `source_version`: Version or vintage of that source, `NA` when the producer has none.
- `recorded_at`: When the record was made, as ISO 8601 UTC.
- `field`, `value`: The evidence field's name and its formatted value for that row. `value` may be `NA`; the field name may not.

Rows are ordered by `row_key`, `field`, `source_id`, `recorded_at` and `value`, in the C locale, so two runs over the same input give byte-identical output apart from `recorded_at`.

The schema is available as data from `row_evidence_schema()` and is asserted with `assert_table_schema()`. It declares no key: combining two runs of the same producer legitimately repeats a claim, so duplication is not an error. Disagreement is, and that is what `evidence_conflicts()` finds.

See Also

`combine_row_evidence()` to merge producers, `evidence_for()` to carry evidence through a join, `evidence_conflicts()` to find disagreement, and `record_provenance()` for the dataset-level record.

Examples

```
cbs <- tibble::tibble(
  area_code = c(724L, 724L, 76L),
  item_cbs_code = c(2511L, 2513L, 2511L),
  year = 2020L,
  value = c(1.5, 2.5, 3.5),
  fao_flag = c("A", "E", "A")
)

# Evidence the table already carries per row, plus one derived field.
evidence <- row_evidence(
  cbs,
  source_id = "FAOSTAT_FBS_New",
  key = c("area_code", "item_cbs_code", "year"),
  fields = list(
    fao_flag = cbs$fao_flag,
    imputed = cbs$fao_flag == "E"
  ),
  source_version = "2024-03-14",
```

```
  recorded_at = as.POSIXct("2026-01-01", tz = "UTC")
)
evidence

# It is a documented schema, so it can be proved rather than trusted.
assert_table_schema(evidence, row_evidence_schema())

# And it round-trips: every column is character.
path <- tempfile(fileext = ".parquet")
write_table_checked(evidence, path)
identical(tibble::as_tibble(nanoparquet::read_parquet(path)), evidence)
unlink(path)
```

`row_evidence_schema` *The declarative schema of a row-evidence table.*

Description

Return the `"whep-row-evidence/1"` contract as a declarative schema, ready for `check_table_schema()` or `assert_table_schema()`. It is plain data, so it can be serialized next to an evidence artifact and used by a consumer that does not load this package.

The schema is closed (`extra_columns = "forbid"`) and ordered (`column_order = "strict"`): an evidence table with an extra column or a permuted column order is not this format, because a consumer rebuilding `row_key` from `key_columns` relies on both.

Usage

```
row_evidence_schema()
```

Value

A schema list, as documented in `check_table_schema()`.

Examples

```
row_evidence_schema()

# The contract is data, so it survives serialization unchanged.
identical(
  yaml::yaml.load(yaml::as.yaml(row_evidence_schema())),
  row_evidence_schema()
)
```

`row_promotion_status` *Report which Rest-of-World members report under their own territory*

Description

Promoting a member of the FABIO Rest-of-World bucket has two halves: the numeric one, which stops its rows being summed into `polity_area_code` 999, and the territorial one, which lets it publish under its own year-scoped `reporting_polity_code` instead of the bucket's `ROW-1850-2025`. Only the first is unconditional. The second needs upstream to name a polity for the area in `faostat_area_polity_map.csv`, and for 30 of the 61 members it does not (whep#717).

Without this, a member still on `ROW-1850-2025` is indistinguishable from the genuine residual: both publish an aggregate polity on the continent `"World"` with no geometry, and nothing says which is which. This names them, and says for each what is missing.

Usage

```
row_promotion_status(crosswalk = NULL)
```

Arguments

`crosswalk`	Crosswalk to inspect. Defaults to the crosswalk this run resolves through, so the answer reflects `options(whep.unfold_rest_of_world)`.

Value

A tibble with one row per Rest-of-World member, ordered by `area_code`:

- `area_code`, `area_name`, `area_iso3c`: The member.
- `cbs`: Whether regions_full flags it as a commodity-balance reporter, which is what `"cbs_reporters"` promotes.
- `status`: One of the four above.
- `n_periods`: How many polity periods it resolves through.
- `polity_codes`: Those periods, comma-separated.

The statuses

- `"own_polity"`: promoted, and publishing under its own polity. 31 members.
- `"polity_unmapped"`: promoted, still on `ROW-1850-2025`, but a live non-aggregate polity carrying the area's ISO3 **does** exist upstream. All that is missing is a row of the FAOSTAT area map naming it for this area, so this is the actionable list to send upstream. 6 members: 22 Aruba, 71 French Southern and Antarctic Territories, 94 Holy See, 218 Tokelau, 243 Wallis and Futuna, 271 South Georgia and the South Sandwich Islands.

- `"no_polity"`: promoted, still on `ROW-1850-2025`, and upstream has no polity for the territory at all. 24 members. Three of them are not territories and must never acquire one – 252 `"Unspecified"`, 254 `"Others (adjustment)"` and the 999 bucket itself, which is excluded here because it is the residual rather than a member of it.
- `"folded"`: this run is re-folding the member, so it is summed into bucket 999 and carries the bucket's polity. Only under `options(whep.unfold_rest_of_world = "none")` or `"cbs_reporters"`.

The ISO3 test behind `"polity_unmapped"` decides **nothing**: it reports that upstream holds a polity this package cannot reach, which is a gap in the map, not a mapping. Resolving an area to a polity by matching ISO3 downstream is the defect whep#711 removed, and #717 argues explicitly against re-minting a territorial identity here.

See Also

`folded_reporting_areas()` for the areas whose rows are summed into another area's code, and `polity_mapping_provenance()` for which authority a resolved row rests on.

Examples

```
status <- row_promotion_status()
table(status$status)
status[status$status == "polity_unmapped", c("area_code", "area_name")]
```

---

run_spatialize *Run the gridded land-use spatialization pipeline*

Description

Wrapper around `build_gridded_landuse()` that resolves a named preset (`"lpjml"` or `"whep"`) into a consistent bundle of input files, engine flags, and output paths. Use this to produce two comparable outputs from the same prepared parquet inputs: an LPJmL/LandInG-faithful run (for cell-by-cell comparison against LPJmL inputs) and the full WHEP run (all historical years, LUH2 type-aware allocation).

Presets can be combined with per-flag `overrides` to produce any intermediate configuration; the resolved configuration is written next to the outputs as `run_metadata.yaml` for traceability.

Usage

```
run_spatialize(
  preset = c("lpjml", "whep"),
  years = NULL,
  components = c("landuse", "livestock"),
  overrides = list(),
  paths = list()
)
```

Arguments

`preset` One of `"lpjml"` or `"whep"`. Selects a default bundle of engine flags and input choices. See *Presets*.

`years` Integer vector of years to spatialize. If `NULL`, the preset default is used: for `"lpjml"` a 10-year benchmark sequence (`seq(1850L, 2020L, by = 10L)`), intersected with the years available in `country_areas`; for `"whep"` all years present in `country_areas`.

`components` Character vector selecting which engines to run. Defaults to `c("landuse", "livestock")`. Pass a subset to run only one (e.g. `"landuse"`). Unknown entries raise an error.

`overrides` Named list of flags that override the preset. Unknown keys raise an error. Recognised entries:

- `use_type_constraint` (logical): enable/disable LUH2 type-aware allocation.
- `aggregate_to_cft` (logical, default `TRUE`): write a CFT-aggregated parquet alongside the crop-level output.
- `max_iterations`: forwarded to the landuse engine.
- `expansion_threshold`: defunct, dropped with a warning; it never changed the allocation (whep#1001). See `build_gridded_landuse()`.
- `pattern_signal_floor`: forwarded to the landuse engine as `config$pattern_signal_floo` the `harvest_fraction` below which a `crop_patterns` cell is float underflow rather than an allocated area. `0` restores the untoleranced behaviour of whep#1070.
- `cft_target`: one of `"whep"` (default for `preset = "whep"`) or `"lpjml"` (default for `preset = "lpjml"`). Selects which column of cft_mapping drives CFT aggregation: `cft_name` (granular 33-class WHEP taxonomy) or `cft_lpjml` (12 LPJmL crop CFTs + single `others` bucket).
- `area_key`: one of `"grid"` (default) or `"polity_area"`, forwarded to both engines. See `build_gridded_landuse()`'s *Which area code the output is keyed on*.
- `country_grid`: which cell-to-polity crosswalk the engines allocate into, `"polycell"` (default), `"centroid"` or `"fraction"`. See *Which cell-to-polity crosswalk*.
- `grid_vintage`: which vintage of the polycell support the level-0 grid is read at, `"year_aware"` (default) or `"snapshot_2015"`. See `read_level_country_grid()`'s *Which vintage of the support level 0 is read at*. The two are different geographies, not two precisions of one: `"snapshot_2015"` allocates every year of a run into the present-day cell-to-country map, `"year_aware"` into the map valid that year. The year-aware read is the default because a cell belongs to the polity that actually held it; the national table is reconciled onto that polity by `.level0_reconcile_vintage()`, which both gridded builders call before the match is judged: see `.grid_vintages()` and `validation/spatialize_grid_vintage.R`. Recorded in `run_metadata.yaml` and per row in `method_grid_vintage`. Not read under `country_grid`

= "centroid" or "fraction", which carry no validity interval at all and are recorded as "static_crosswalk".

- level (integer, default 0L): containment depth the grid is resolved at. 0L is today's cell-to-area_code grid; 1L and deeper key the cells on admin units through read_level_country_grid(), which only the "polycell" crosswalk supports.
- granted_containers (default NULL): the container area_codes the run grants a depth, e.g. 110L for Japan. **Required whenever** level > 0 and refused at level = 0; a depth run must say whose depth it is, because that decides which containment edges read_level_country_grid() reads, which admin-share rows constrain the run, and which cells build_allocation_layer() takes from the deep grid rather than from level 0. See *What a granted depth runs*.
- double_claim: which clashes the depth read's double-claim gate refuses, "co_presence" (default) or "measured", forwarded to read_level_country_grid(). The default is the fail-closed rule.
- output_level (integer, default 0L): grain of the crop output. 0L sums granted-depth rows back onto the container, so (lon, lat, area_code, item_prod_code, stays unique and the schema equals a level-0 run's; a positive value returns unit-grain rows carrying level_polity_code. It may not exceed level.
- constraint_exclude (default NULL): container x year ranges to hold out of the admin constraint, as a list named by area_code, such as list("840" = 1961:1989). That is the key resolve_admin_shares() reads, and it is checked here against the same rule, so a hold-out this run records is one the resolver can honour. Recorded in run_metadata.yaml and consumed by the admin-shares resolver; it does not by itself change a run that has no admin constraint wired.
- livestock_proxy: one of "luh2" (default) or "glw3", forwarded to build_gridded_livestock()'s proxy_method. Under "glw3" the density table is read with read_glw_density(), which needs a WHEP_GLW3_DIR tree and aborts without one; under "luh2" it is not read at all.
- livestock_glw_variant: which GLW3 product a "glw3" run allocates on, "DA" (default, the dasymetric rasters) or "AW" (the areal-weighted ones), forwarded to read_glw_density()'s variant. They are different within-country geographies, so the resolved value is recorded twice: in run_metadata.yaml with the rest of the config, and per row in the output's method_livestock_proxy as "glw3_da" or "glw3_aw". Ignored under livestock_proxy = "luh2", which reads no raster.

paths — Named list of filesystem paths. Recognised entries:

- l_files_dir: path to the L_files root, for local prepared inputs.
- input_dir: directory holding the prepared input parquets. If NULL and l_files_dir is unset, the pinned WHEP spatialization inputs are used.
- out_dir: output directory. If NULL, defaults to <l_files_dir>/whep/spatialize/<preset> when l_files_dir is supplied, otherwise to a session temporary di-

rectory (suffixed with `_custom` when `overrides` is non-empty). Created if missing.

Value

Invisibly, a named list with `preset`, `components`, `cft_target`, resolved `config`, `years`, `out_dir`, `output_paths`, and `admin` – the resolver's coverage report and the constraint summary at a granted depth, `NULL` otherwise.

Presets

`lpjml` LandInG-faithful configuration: no LUH2 type-aware allocation (`use_type_constraint = FALSE`) and a short default year sample suited to comparison against LPJmL inputs.

`whep` Full WHEP configuration: LUH2 type-aware allocation (`use_type_constraint = TRUE`) and the full historical year range present in `country_areas`.

Inputs read from `input_dir`

Landuse (`components` contains `"landuse"`):

- `country_areas.parquet`
- `crop_patterns.parquet`
- `gridded_cropland.parquet`
- `country_grid.parquet`
- `type_cropland.parquet` (required when `use_type_constraint = TRUE`).

Livestock (`components` contains `"livestock"`):

- `livestock_country_data.parquet`
- `gridded_pasture.parquet`
- `gridded_cropland.parquet`, `country_grid.parquet`
- `manure_pattern.parquet` (optional, enables manure-intensity weighting if present).
- `livestock_mapping.csv` from the installed package.
- The GLW3 rasters under `WHEP_GLW3_DIR`, read only when `livestock_proxy = "glw3"` (see `read_glw_density()`).

What a granted depth runs

With `level > 0` the landuse step does not call `build_gridded_landuse()` on a level-0 grid. It runs the subnational chain, in this order, and each step aborts naming what it is missing rather than continuing on the level-0 pattern:

1. `read_level_country_grid()` twice – level 0 for the ungranted countries, `level` scoped to `granted_containers` for the granted ones – and `build_allocation_layer()` to assert the two partition every cell. Both halves are read year-aware, which is what `method_grid_vintage` has always recorded for a depth run.

2. `read_admin_shares()`, scoped to `granted_containers`. A granted container with no admin row aborts with class `whep_run_admin_container_absent`; an unregistered pin with `whep_run_no_admin_shares`.
3. `resolve_admin_units()`, turning each source's native identifier into a polity code under the code system that identifier belongs to. Rows resolving to nothing are dropped and counted; a source resolving to nothing at all aborts with `whep_run_admin_unresolved`, and a source whose code system is undeclared with `whep_run_admin_code_system`.
4. `resolve_admin_shares()`, applying indicator and source precedence and honouring `constraint_exclude`. A constraint sharing no `level_polity_code` with the layer aborts with `whep_run_admin_layer_mismatch`, because a run whose constraint matched nothing is indistinguishable from an unconstrained one in every output it writes.
5. `allocate_level_crops()`, splitting each national total across the container's units and spreading each unit's target over that unit's cells.
6. `reconcile_admin_allocation()` and `seam_gate()`, written beside the parquets as the run's own audit trail.

What actually happened is recorded in `run_metadata.yaml` under `admin_constraint` – the resolved, dropped and held-out row counts, the units constrained, the `method_crop_alloc` tally over the targets, the share basis the gate judged on, and the gate's verdict – and per row in the targets' own `method_crop_alloc`. A level-0 run records `admin_constraint: none`.

Which cell-to-polity crosswalk

The producer builds two crosswalks from the same polygons. `"centroid"` is the deployed `spatialize-country-grid` pin: one `area_code` per 0.5-degree cell, winner-take-all at a border, no share column, so a whole border cell goes to a single polity. `"fraction"` is `cell_polity_fraction.parquet`, which splits each border cell by fractional coverage; the engines already read its `polity_frac` as `cell_area_frac`, so no engine change is involved.

They are alternatives, never a fallback. The fractional parquet used to carry a different area vocabulary from the centroid grid — it keyed Ethiopia `62` and Sudan `206` where today's `regions.csv` uses `238` and `276`, so substituting it dropped both countries entirely (whep#461). Regenerating it closed that gap: the two grids now carry the same 178 area codes, it is published as the `spatialize-cell-polity-fraction` pin so no user has to rebuild it, and `build_cell_polity()` refuses a copy still holding a retired code instead of deleting the countries silently (whep#694). It still cannot rescue a polity smaller than a cell, because its producer restricts it to the cells the centroid grid already has, and it drops 4 of those cells, whose only land is a sliver covering the 0.5-degree cell centre but no 1/12-degree subcell centre. Whichever is selected, `build_gridded_landuse()` and `build_gridded_livestock()` now warn once per call naming every reporting area the chosen grid has no cell for and the national total at stake.

Outputs written to `out_dir`

Every parquet below carries `method_grid_vintage`, the geography the run allocated into: `"year_aware"`, `"snapshot_2015"` or `"static_crosswalk"`.

- `gridded_landuse_crops.parquet` — crop-level output.
- `gridded_landuse.parquet` — CFT-aggregated output (when `aggregate_to_cft = TRUE`).
- `gridded_livestock_emissions.parquet` — gridded livestock stocks and emissions (when livestock component selected).
- `run_metadata.yaml` — resolved preset, components, flags, years, timestamp, package version, and the resolved `method_grid_vintage`.
- `admin_coverage.csv` — which admin source constrained each container x item x year, at what tier, grain and depth. Written only when `level > 0`, with its header and no rows where no coverage is granted. See `admin_coverage_prototype()`.
- Eleven further CSVs, written only when `level > 0` and the landuse component ran: `admin_targets.csv` (one row per unit, item and year with its share, target and `method_crop_alloc`), `admin_group_coverage.csv`, `admin_conservation.csv`, `admin_breach.csv`, `admin_reconciliation.csv`, `admin_reconciliation_units.csv`, `admin_unit_cropland.csv`, `admin_seams.csv` and the three seam-gate tiers `admin_seam_gate_a.csv`, `admin_seam_gate_b.csv`, `admin_seam_gate_c.csv`.

See Also

`build_gridded_landuse()`.

Examples

```
# Dispatch to the engine with a filtered year range (offline
# example; normally called against prepared parquet inputs).
country_areas <- tibble::tribble(
  ~year, ~area_code, ~item_prod_code, ~harvested_area_ha,
  1999L,         1L,             15L,                500,
  2000L,         1L,             15L,               1000
)
crop_patterns <- tibble::tribble(
  ~lon,  ~lat, ~item_prod_code, ~harvest_fraction,
   0.25, 50.25,             15L,               0.6,
   0.75, 50.25,             15L,               0.4
)
gridded_cropland <- tibble::tribble(
  ~lon,  ~lat,  ~year, ~cropland_ha,
   0.25, 50.25, 1999L,          800,
   0.75, 50.25, 1999L,          500,
   0.25, 50.25, 2000L,          800,
   0.75, 50.25, 2000L,          500
)
country_grid <- tibble::tribble(
  ~lon,  ~lat, ~area_code, ~cell_area_frac,
   0.25, 50.25,         1L,              1,
   0.75, 50.25,         1L,              1
)
build_gridded_landuse(
  country_areas, crop_patterns, gridded_cropland, country_grid,
  config = list(years = 2000L)
```

```
)
```

`run_typology_sensitivity` *Run one-at-a-time sensitivity analysis on typology thresholds.*

Description

Varies each threshold by +/- `variation` (one at a time) and reports the share of province-year observations that retain the same typology as the baseline classification.

Usage

```
run_typology_sensitivity(
  n_prov_destiny = NULL,
  variation = 0.2,
  baseline = NULL
)
```

Arguments

`n_prov_destiny`	Nitrogen flows tibble. If `NULL`, loaded automatically.
`variation`	Relative variation applied to each threshold (default 0.2 = 20%).
`baseline`	Pre-computed indicator table from `create_typologies_spain()`, carrying `year`, `province_name` and the indicator columns the thresholds act on. If `NULL`, computed automatically (slow).

Value

A tibble with columns `threshold`, `direction`, and `agreement_pct`.

Examples

```
# `baseline` carries the indicator columns the thresholds act on plus the
# unperturbed `Typology_base` each perturbation is compared against. Two
# provinces show the output shape; the real analysis runs over 50 provinces
# and 1860-2023.
baseline <- tibble::tribble(
  ~year,
  ~province_name,
  ~production_seminatural,
  ~production_crops,
  ~animal_ingestion,
  ~synthetic_share,
  ~crop_productivity,
  ~Livestock_density,
  ~imported_feed_share,
  ~feed_from_seminatural_share,
```

```
  ~local_feed_share,
  ~Manure_share,
  ~Typology_base,
  2000,
  "A",
  1,
  100,
  5,
  0.8,
  40,
  0.1,
  0.1,
  0.1,
  0.1,
  0.1,
  "Specialized cropping systems (intensive)",
  2000,
  "B",
  1,
  10,
  50,
  0.1,
  40,
  0.5,
  0.1,
  0.5,
  0.5,
  0.5,
  "Connected crop-livestock systems (intensive)"
)
sensitivity <- run_typology_sensitivity(baseline = baseline)
```

`seam_gate` *Gate an admin-shares back-cast at its seams*

Description

Judge a back-cast share table at every seam the resolver found, in the three tiers the subnational-spatialization plan's seam section asks for. Nothing is aborted and nothing is repaired: each tier returns its own table with a `pass` column and the numbers behind it, and the caller decides what a failure means. That is deliberate. The gate is evidence for a release decision, not a guard inside a pipeline.

Seam years are read from `seams`, never assumed. This function contains no year: a run whose statistics start in 1850 and one whose statistics start in 1974 are gated by the same code at different years.

Usage

```
seam_gate(
```

```
  shares_backcast,
  seams,
  cells = NULL,
  holdout = NULL,
  tolerances = seam_gate_tolerances()
)
```

Arguments

`shares_backcast`	A back-cast share table, the `shares` element of `backcast_admin_shares()`. Columns: `area_code`, `level`, `item_prod_code`, `level_polity_code`, `year`, `share`, `treatment`, optionally `value`. One row per unit and year.
`seams`	A seam list, the `seams` element of `resolve_admin_shares()`: `area_code`, `level`, `item_prod_code`, `seam_year`, `seam_kind`. Every seam must name a series `shares_backcast` carries, or the gate aborts (`whep_seam_gate_seams_unmatched`): such a seam is gated by nothing and leaves no row in any tier, so it can be reported nowhere else. The resolver emits one `"start"` seam per series, so a matched pair satisfies this by construction and a breach means the two tables come from different runs, or one of them was filtered. A zero-row `shares_backcast` – the tier-C-only call – is exempt.
`cells`	Optional crop-level engine output for tier C: `lon`, `lat`, `year`, `area_code`, `crop_name`, and either `harvested_ha` or `rainfed_ha` plus `irrigated_ha`. `polycell_id`, `cell_id`, `level_polity_code` and `regime` are used when present. `NULL` (default) leaves tier C unevaluated.
`holdout`	What the tier-B hold-out needs, or `NULL` (default), which leaves it unevaluated. A list of `extent`, the per-unit extent table from `aggregate_unit_extent()` that the back-cast was run with, and optionally `args`, a named list of further arguments for `backcast_admin_shares()` – normally `list(settings = <the knobs the c` so the hold-out is run the way the run was. `shares`, `extent` and `seam` are set by the leg and are refused in `args`. With `holdout` supplied, `shares_backcast` must also carry `indicator_used` and `treatment_year`, which the back-cast reads. The re-run's own input errors – an `extent` with two rows for a unit-year, say – surface as errors from it. That is not the gate failing: a gate outcome is always a returned row.
`tolerances`	The numbers to judge with, from `seam_gate_tolerances()`.

Value

A list:

- `tier_a`: one row per series `shares_backcast` carries, with `t0`, `n_units`, `n_observed`, `share_sum`, `max_rel_diff`, `basis`, `seam_start_year`, `matches_seam_start`, `pass` and `reason`. `pass` is `NA` where the tier judged nothing, and `reason` says which of the two ways.
- `tier_b`: one row per gated unit-seam pair, with `basis`, `log_ratio`, `q_reference`, `n_reference`, `beyond_quantile` and `status`, plus the container gate (`n_gated`, `n_beyond`, `frac_beyond`, `threshold`, `gate_status`, `pass`) broadcast onto every row of that container and `basis`.

- `tier_b_holdout`: one row per unit and withheld year, with `anchor_year`, `horizon`, `share_observed`, `share_backcast`, `log_ratio`, `status` and the per-series gate; plus one row, keyed on the series alone, for every start-seam series the leg could not run (`status` beginning `"not_applicable"`). Zero rows when no series has a `"start"` seam.
- `tier_c`: one row per gated `(area_code, seam_year)`, with the three pairs' scanned series counts and flag rates, `neighbour_rate`, `excess`, `n_regime_mismatch`, `pass` and `reason`. Zero rows when `cells` is `NULL`.
- `verdict`: a named logical, `tier_a` / `tier_b` / `tier_b_holdout` / `tier_c` / `overall`. A tier is `TRUE` when every gate it evaluated passed, `FALSE` when any failed and `NA` when it evaluated none – which is what a container of nothing but `"vacuous_by_construction"` rows gives, and what an empty seam list gives in every tier at once. `overall` is `FALSE` if any tier failed; `NA` if any series of `shares_backcast` left tier A, tier B and the hold-out with nothing but `NA` for a `pass`, or if no tier evaluated anything; else `TRUE`. A pass therefore covers every series the gate was handed. Tier C does not count towards that coverage: it is keyed on `(container, seam year)` and carries no item, so it cannot stand in for a series.

Tier A – identity at the anchor

At each series' `t0` – the first year carrying an observed row in `shares_backcast`, which is the anchor the back-cast actually used – the table must still hold the observation:

- every unit's row at `t0` is `treatment == "observed"`;
- the unit shares at `t0` sum to 1 within `identity_rel`;
- where the anchor rows carry `value`, each share equals that value's share of the anchor-year total within `identity_rel` (`basis = "value"`; `"share_only"` where any anchor value is missing, as it is for a residual pseudo-unit, and then only the checks above bite);
- `t0` is the `"start"` seam `seams` names for that series.

What tier A cannot see is the modulation itself: the extent table is not an argument, so `E_u(t0) / E_u(t0) = 1` is verified by `backcast_admin_shares()`'s own tests, not here. The last check is the one that catches a `shares_backcast` and a `seams` built from different runs – and a deliberate `seam =` override, which moves `t0` on purpose, so a temporal hold-out is expected to fail it and says so in `reason`.

Two states are not a pass, and each says so on its own row rather than in a verdict. Both keep their measured numbers, because what is withheld is the verdict and not the evidence:

- **no `"start"` seam names the series**, so the last check above did not run: `pass` is `NA`, `reason = "no_start_seam"`. Gated here means what the check means – a seam year to compare `t0` with – so a series named only by, say, a `source_switch` seam is ungated in tier A and gated in tier B, where that seam actually lands. An empty or filtered seam list would otherwise certify a run by handing tier A nothing to check.
- **no observed row at all**, so there is no anchor to measure: `pass` is `NA`, `reason = "no_observed_anchor"` and every anchor measurement on the row is `NA`. Such a series has a row here rather than none, because a series that never reached the tier is the one state a table of tier-A rows cannot otherwise show.

Both reasons come after the identity ones, so a series whose identity is wrong still fails on the identity: a missing seam withholds a verdict, it does not excuse one.

Tier B – the governed quantity

The seam log-ratio `|log(s_u(t0) / s_u(t0 - 1))|` of every unit at every seam year, judged against the empirical distribution of the **observed** consecutive log-ratios of the same `(container, level, item)`, pooled over its units. Gated pairs are held out of their own reference distribution.

The statistic is the fraction of gated pairs beyond that distribution's `reference_quantile`, aggregated per container across its items, and the gate is `null_rate + binomial_sigma * sqrt(null_rate * (1 - null_rate) / n)`: under the null that a seam pair is an ordinary pair the fraction is binomial, and this is its upper band. A container whose seams are invisible against its own year-to-year variation passes; one whose seams step further than its ordinary years do, does not.

That reasoning holds only where both sides of the pair are observations, which `basis` records per pair and the container gate is grouped by:

- `basis = "observed_both_sides"`: the seam year's row and the year before it are both `treatment == "observed"`. The tier keeps its meaning, and its `pass` enters the verdict. This is the `source_switch`, `grain_switch`, `nuts_version_switch`, `coverage_change` and `indicator_switch` case.
- `basis = "vacuous_by_construction"`: the year before the seam is a row the back-cast produced, or is absent. It is read off the two rows' `treatment`, not off the seam kind, because that is the operative fact. At a `"start"` seam of a table whose anchor matches it – tier A's last check – it always holds: `t0` is the series' first observed year, so `s_u(t0 - 1)` is `s_u(t0) * E_u(t0 - 1) / E_u(t0)` renormalised, one year of smooth extent change. Scored against a reference built from the noisier observed year-to-year moves it cannot fail: two extent proxies with opposite per-unit trends, whose reconstructions differ by a factor of 0.40 to 1.63 ten years down, both return `n_beyond = 0`. Such rows report `gate_status = "vacuous_by_construction"` and `pass = NA`, and do **not** enter the verdict; `frac_beyond` and `threshold` are still filled in, because the numbers are worth reading even though they decide nothing.

Tier B hold-out – what stands in at a start seam

Only when `holdout` is supplied. For every series with a `"start"` seam, the first `holdout_k` observed years are withheld, the series is back-cast again from the next observed year (the `seam =` override of `backcast_admin_shares()`, which is the same lever the verification protocol's temporal hold-out uses), and the reconstruction is scored against what was actually observed in those years: `|log(s_hat_u(t) / s_u(t))|` per unit and withheld year, against the same reference quantile of that series' observed consecutive log-ratios. The two are commensurable under iid year-to-year noise: a reconstruction error is a difference of two years' noise (the anchor's and the withheld year's), like a one-year move, and neither widens with the horizon. On this package's fixture the error distribution comes out slightly narrower than the reference, so the realised null exceedance rate sits under `null_rate` rather than over it; under persistent noise a deep hold-out over-rejects instead. Both departures are the safe direction for a gate, and neither is assumed: the measured fractions are in the tests.

The reference excludes every pair at or below the hold-out anchor as well as every seam year, for the reason tier B holds its gated pairs out of their own reference: a series whose earliest years are unusual would otherwise both widen the yardstick and be measured by it. The consequence is worth stating plainly – where the earliest years genuinely move differently from the rest of the record, this leg fails a proxy that is right about the rest. It fails loudly rather than passing quietly.

The band divides by the number of **units** with a scored pair, not by the number of pairs. A unit's `holdout_k` errors all carry the anchor year's own noise, so they are one cluster and not `holdout_k` independent draws; dividing by pairs made this leg flag a correct proxy on the package's own fixture. `min_gated` therefore reads as a minimum unit count here (`gate_status = "too_few_units"`).

A series with `holdout_k` or fewer observed years is reported `not_applicable`, never passed, and so is every series when `holdout` is not supplied: a start seam then has no evidence at all behind it, which the returned table says on its own row.

What a pass certifies is a `holdout_k`-year extrapolation. Where the published back-cast runs deeper – `n_backcast_years` on every row says how much deeper – it is evidence about the first `holdout_k` years of it and no more.

Every seam kind in `seams` is gated, coverage changes and indicator switches included: the seam list is taken as given rather than filtered, so a kind added upstream is gated without a change here. A pair that cannot be judged is reported with its reason in `status` (`"na_share"`, `"zero_share"`, `"no_previous_row"`, `"no_reference"`) and left out of the statistic rather than counted as a pass.

A unit that *stops* reporting at a seam year has no `s_u(t0)` and so no pair at all: it is not a row of `tier_b`. Its mass does not vanish from the gate, because it moves into its siblings' shares, whose ratios at that year are gated.

Tier C – cell smoke

Only when `cells` is supplied. `check_series_jumps()` on each cell's share of the national total, across three pairs per gated seam year: `(t0-2, t0-1)`, `(t0-1, t0)` and `(t0, t0+1)`. Dividing by the national total is what takes the national series' own splice out of the comparison; that splice is not this feature's to answer for. The gate is that the seam pair's flag rate exceeds the mean of its neighbours' by at most `cell_excess`.

The scan runs one pair at a time, on a table pre-subset to that pair's two years, so the flags returned are exactly that pair's and the memory bound is two years of cells rather than the whole run. Cells under `cell_min_ha` in **both** years are dropped first.

Tier C is keyed on `(container, seam year)`, not on the item: the crop-level engine output carries `crop_name`, not `item_prod_code`, so there is nothing to join a per-item seam to. Pass the crop-level output, never the CFT aggregation, whose rows pool several items.

Where `cells` carries a `regime` column – the within-unit weight regime the allocation records per (unit, item, year) – it must be identical on both sides of every gated pair. A regime flip at a seam fails tier C on its own, whatever the flag rates do, because a cell series that changes weight regime at the seam is discontinuous by construction rather than by measurement.

Nothing in the package writes that column today: the allocation's regime label is `method_crop_alloc`, which lives on the targets table and not on the cell grid, so on the shipped crop-level

output this axis is **not evaluated**. Such a gate reports `regime_checked = FALSE` and `n_regime_mismatch = NA` – `NA` rather than zero, because an axis nobody looked at has no count of mismatches – and the report line says how many of the gates are in that state, so a pass is not read as evidence that no regime flipped.

Examples

```
# Two units back-cast to 1898 from an anchor at 1900, with the seam
# list the resolver would have emitted for that series.
shares <- tibble::tibble(
  area_code = 900L,
  level = 1L,
  item_prod_code = 15L,
  level_polity_code = rep(c("A1", "A2"), each = 3),
  year = rep(1898:1900, times = 2),
  share = c(0.36, 0.38, 0.40, 0.64, 0.62, 0.60),
  treatment = rep(
    c("backcast_t0_geometry", "backcast_t0_geometry", "observed"),
    times = 2
  )
)
seams <- tibble::tibble(
  area_code = 900L,
  level = 1L,
  item_prod_code = 15L,
  seam_year = 1900L,
  seam_kind = "start"
)
seam_gate(shares, seams)$tier_a
```

`seam_gate_tolerances` *Tolerances of the admin-shares seam gate*

Description

The user-settable numbers `seam_gate()` judges with, in one place so a run records what it was gated against. Each is a **proposal** of the subnational-spatialization plan's T29 and may be replaced by its T31 allocation-policy decision; none is a measured quantity and none is derived from data, so each is stated here with the reason it has the value it has rather than left at a call site.

- `identity_rel` (1e-8): tier A's relative tolerance on a unit-share total, the plan's figure. The back-cast leaves the anchor year's observed rows untouched, so the only thing between the two sides is floating-point summation of at most a few hundred shares, which 1e-8 is orders of magnitude above.
- `identity_floor` (1e-12): denominator guard for that relative comparison, so a unit whose observed share rounds to zero does not turn a difference of 1e-18 into a ratio of 1e6. A numerical guard, not a scientific threshold.

- `reference_quantile` (0.95): tier B's reference point, the plan's Q95 of the observed consecutive log-ratio distribution. The null exceedance rate `null_rate` follows from it as `1 - quantile` and is returned rather than set, so the two cannot disagree.
- `binomial_sigma` (2): how many standard errors of slack the tier-B gate allows above `null_rate`, giving the plan's `0.05 + 2 * sqrt(0.05 * 0.95 / n)`. Two standard errors is a ~95% one-sided band under the null that a seam pair is an ordinary pair.
- `min_reference` (20): the smallest reference sample a quantile is taken from. Below it the Q95 is essentially the largest one or two observed ratios, so the series is reported unevaluable instead of being judged against a threshold that is itself noise. Assumed, unverified: no measurement fixes it, and T31 may.
- `cell_ratio_bounds` (`c(0.55, 1.6)`): tier C's plausible band, `check_series_jumps()`'s own default, kept so the cell smoke test speaks the same language as the rest of the `check_*` library.
- `cell_min_ha` (100): tier C's hectare floor. A cell holding under a square kilometre of a crop in both years of a pair is dropped before the scan, its share being numerically tiny and its ratio dominated by rounding. The same 100 ha `backcast_admin_shares()` inherits from `.fix_luh2_crop_collapse()`'s `min_neighbor_mha = 0.001` Mha. `check_series_jumps()`'s `min_value` cannot express it: that applies to the scanned column, which here is a share, not an area.
- `cell_excess` (0.01): the plan's one percentage point. Tier C passes when the seam pair's flag rate exceeds the mean of its two neighbouring pairs' rates by no more than this.
- `holdout_k` (10): how many of a series' earliest observed years the tier-B hold-out withholds. The leg's power grows with the horizon, because a wrong per-unit trend in the extent proxy accumulates linearly with the years back-cast while the yardstick stays a single year's move: on this package's own two-history fixture (`test_admin_shares_gate.R`), two extent tables whose reconstructions differ by a factor of 0.40 to 1.63 at ten years are **not** separated at `holdout_k = 3` (both pass) and are separated decisively at 10 (0.008 against 0.383 of pairs beyond the reference quantile). Each withheld year is also an observation removed from both the anchor and the reference, so `holdout_k` cannot approach the length of the record; ten leaves two decades of reference pairs on a thirty-year series. Assumed, unverified: no measurement fixes it, and it is a choice about how deep an extrapolation the gate certifies, not a property of the data.

Two further numbers are **derived** from those, and returned rather than set, so they cannot disagree with them:

- `null_rate`, the tier-B null exceedance rate `1 - quantile`.
- `min_gated`, the smallest number of gated pairs a container gate is pronounced on. Below it the gate is arithmetically incapable of passing a container with a single exceeding pair: one pair out of `n` is a rate of `1/n`, and `1/n` exceeds the band whenever `1/n > null_rate + binomial_sigma * sqrt(...)`. At the defaults that is every `n` up to 3, so `min_gated` is 4 and a container with three gated pairs is reported `"too_few_pairs"` rather than failed. This is a property of the band, not a preference.

Usage

```
seam_gate_tolerances(
```

```
  identity_rel = 1e-08,
  identity_floor = 1e-12,
  reference_quantile = 0.95,
  binomial_sigma = 2,
  min_reference = 20L,
  cell_ratio_bounds = c(0.55, 1.6),
  cell_min_ha = 100,
  cell_excess = 0.01,
  holdout_k = 10L
)
```

Arguments

`identity_rel`	Relative tolerance for tier A.
`identity_floor`	Denominator guard for tier A.
`reference_quantile`	Reference quantile for tier B, in (0, 1).
`binomial_sigma`	Standard errors of slack in tier B's gate.
`min_reference`	Smallest reference sample tier B will judge against.
`cell_ratio_bounds`	Length-2 plausible band for tier C, passed to `check_series_jumps()`.
`cell_min_ha`	Hectare floor applied to tier C's cells before the scan.
`cell_excess`	Largest flag-rate excess tier C accepts, as a fraction.
`holdout_k`	How many of a series' earliest observed years the tier-B hold-out withholds.

Value

A named list of the arguments plus `null_rate` and `min_gated`.

Examples

```
seam_gate_tolerances()$null_rate
seam_gate_tolerances(reference_quantile = 0.99)$null_rate
```

`sjos_levels` *Safe-and-just nitrogen classification levels and colours.*

Description

The ordered levels of the SJOS-N 2-way classification: the reactive-nitrogen boundary axis (`Within_boundary` versus `Exceedance`) crossed with the nourishment axis (`Under`, `Adequate`, `Over`), with a plotting colour per combined level. Used to classify and colour polities on the safe-and-just nitrogen space. The rows are in the realised `afsetools::load_vectors()` `SJOS_levels` order, which that source reverses: `"Exceedance Over"` first, `"Within_boundary Under"` last. That is the factor-level order Global's SJOS-N figures stack and legend on, so a figure built from these levels reproduces theirs.

Usage

```
sjos_levels
```

Format

A tibble with columns:

level Combined classification level (e.g. `"Exceedance Over"`, `"Within_boundary Under"`).

order Integer plotting/factor order (1 to 6), matching the row order.

colour Plotting colour (an R colour name).

Source

Boundary axis from the agricultural reactive-nitrogen boundary framework of Schulte-Uebbing, L. F., Beusen, A. H. W., Bouwman, A. F. & de Vries, W. (2022). doi:10.1038/s41586022051582 and de Vries, W. et al. (2013). doi:10.1016/j.cosust.2013.07.004; nourishment axis from the nourishment thresholds (see nourishment_thresholds). The level labels, their order and the colours are ported by value from `afsetools::load_vectors()` (`SJOS_levels`, which is wrapped in `rev()` there, and `SJOS_colours`); verify against Zotero before any manuscript use.

Examples

```
sjos_levels
```

`smil_2001_synthetic_n_global`

Smil (2001) global synthetic nitrogen production, 1913-2000

Description

Global synthetic-nitrogen production anchors from Smil (2001) "Enriching the Earth", Tables 5.2 and 5.3, cross-checked with Smil (2002) Ambio 31:126-131. Anchor years span 1913 (first commercial Haber-Bosch plant at BASF Oppau) to 2000. Used by `prepare_nitrogen_inputs()` to backcast country-level synthetic N for the pre-FAOSTAT period (years before 1961): the temporal shape is taken from this global series and downscaled to each country using its 1961-1965 share of global FAOSTAT synthetic N.

Pre-1913 values are treated as zero by the consumer and are not stored here.

Usage

```
smil_2001_synthetic_n_global
```

Format

A tibble with one row per anchor year:

- `year`: Integer anchor year (1913, 1920, 1925, ..., 2000).
- `global_kt_n`: Global synthetic-N production in kt N.

Source

Smil, V. (2001) *Enriching the Earth: Fritz Haber, Carl Bosch, and the Transformation of World Food Production*, MIT Press. Tables 5.2 and 5.3.

Examples

```
head(smil_2001_synthetic_n_global)
```

`soc_crop_group` *Assign each crop to its soil-carbon crop group.*

Description

Map `item_prod_code` and an irrigation flag to the crop group the soil carbon balance marches: `cropland_rainfed_herbaceous`, `cropland_irrigated_herbaceous`, or one group per woody species such as `cropland_rainfed_olive` (the species slug comes from `Name_biomass`, the column the source implementation builds its category from, so it is "Olive" rather than the `names_cats` key "Olives"). Every label keeps the `cropland_` prefix, which is what lets the balance recognise a group as cropland for its soil-cover curve, its C:N lookup and its water term without enumerating groups.

Usage

```
soc_crop_group(
  item_prod_code,
  irrigated = FALSE,
  vocabulary = whep::items_prod_full
)
```

Arguments

`item_prod_code`	Integer vector of FAOSTAT production item codes.
`irrigated`	Logical vector, recycled: is the area irrigated?
`vocabulary`	The crop vocabulary, a tibble with `item_prod_code`, `Herb_Woody` and `Name_biomass`. Defaults to items_prod_full.

Value

A character vector of group labels, one per input element. An item with no `Herb_Woody` classification aborts, naming it: an unclassified crop silently pooled into the wrong group would move carbon between groups with nothing recording it.

Examples

```
soc_crop_group(c(15L, 260L, 260L), irrigated = c(FALSE, FALSE, TRUE))
```

---

soc_rate_modifier_amg

Compute the AMG (AMGv2) annual climate rate modifier.

Description

Annual AMGv2 climate rate-modifying factor: the logistic temperature function f(T) normalized to 1 at the 15 degree reference and the logistic moisture function f(H) of the annual water balance, multiplied and averaged over the supplied series.

Usage

```
soc_rate_modifier_amg(temp_c, water_balance_mm)
```

Arguments

`temp_c`	Numeric mean annual air temperature series (degrees Celsius).
`water_balance_mm`	Numeric annual water balance series (precipitation plus irrigation minus potential evapotranspiration, mm).

Value

The annual mean of the f(T) * f(H) product (a single numeric).

Source

Clivot, H., Mouny, J.-C., Duparque, A., Dinh, J.-L., Denoroy, P., ... Mary, B. (2019). Modeling soil organic carbon evolution in long-term arable experiments with AMG model. doi:10.1016/j.envsoft.2019.04.004; temperature form from Saffih-Hdadi, K. & Mary, B. (2008). doi:10.1016/j.soilbio.2007.08.022.

Examples

```
soc_rate_modifier_amg(
  temp_c = c(8, 15, 22),
  water_balance_mm = c(-100, 0, 200)
)
```

`soc_rate_modifier_century`
Compute the Century DEFAC annual climate rate modifier.

Description

Annual Century DEFAC rate-modifying factor: the original monthly-Century Poisson temperature factor (which is 1 at the 35 degree optimum) and the precipitation-over-evapotranspiration logistic moisture factor, multiplied and averaged over the supplied series.

Both terms are SoilR's, and so is their pairing. Sierra et al. (2012) Table 1 lists two Century temperature functions: `fT.Century1`, attributed to Burke et al. (2003), and `fT.Century2`, attributed to Adair et al. (2008), which is the same expression multiplied by 3.439. This function implements `fT.Century1`, which carries no such prefactor; the moisture term is `fW.Century` verbatim; and `fT.Century1 * fW.Century` is exactly the climate decomposition index Sierra et al. compute in their own Sect. 4.4. The rate constants this modifier scales are SoilR's Century defaults as well (`whep::soc_turnover_params`: 0.094, 0.35, 0.14, 0.0038 and 0.00013 per week for the structural, metabolic, active, slow and passive pools), so rates and modifier come from one implementation rather than two.

What that does NOT establish is the calibration basis of the weekly rates themselves: whether Parton et al. (1987) publishes them as the maxima reached at DEFAC = 1. If they are not, every Century stock is biased high. That question needs the paper and is open as whep#345.

Usage

```
soc_rate_modifier_century(temp_c, precip_mm, pet_mm)
```

Arguments

`temp_c`	Numeric monthly air (or soil-surface) temperature series (degrees Celsius).
`precip_mm`	Numeric monthly precipitation series (mm).
`pet_mm`	Numeric monthly potential evapotranspiration series (mm).

Value

The annual mean of the temperature-by-moisture product (a single numeric).

Source

Parton, W. J., Schimel, D. S., Cole, C. V. & Ojima, D. S. (1987). doi:10.2136/sssaj1987.03615995005100050015x; SoilR implementation: Sierra, C. A., Mueller, M. & Trumbore, S. E. (2012). doi:10.5194/gmd510452012. Temperature function (`fT.Century1`): Burke, I., Kaye, J., Bird, S., Hall, S., McCulley, R. & Sommerville, G. (2003). Evaluating and testing models of terrestrial biogeochemistry: the role of temperature in controlling decomposition. In *Models in Ecosystem Science*, Princeton University Press. The 3.439-scaled variant `fT.Century2`, which this function does NOT implement: Adair, E., Parton, W., Del Grosso, S., Silver, W., Harmon, M., Hall, S., Burke, I. & Hart, S. (2008). doi:10.1111/j.13652486.2008.01674.x.

Examples

```
soc_rate_modifier_century(
  temp_c = c(15, 25, 35),
  precip_mm = c(40, 60, 80),
  pet_mm = c(50, 70, 90)
)
```

---

soc_rate_modifier_icbm

Compute the ICBM annual climate rate modifier.

Description

Annual ICBM re_clim rate-modifying factor: a Ratkowsky-type temperature response and a piecewise volumetric-moisture response, multiplied and rescaled to the Swedish Ultuna reference site, averaged over the supplied series.

Usage

```
soc_rate_modifier_icbm(temp_c, theta, t_field, t_wilt, porosity)
```

Arguments

`temp_c`	Numeric soil (or air-proxy) temperature series (degrees Celsius).
`theta`	Numeric volumetric soil water content series (fraction).
`t_field`	Field-capacity volumetric water content (fraction).
`t_wilt`	Wilting-point volumetric water content (fraction).
`porosity`	Soil porosity (fraction).

Value

The annual mean of the monthly re_clim values (a single numeric).

Source

Katterer, T., Reichstein, M., Andren, O. & Lomander, A. (1998). Temperature dependence of organic matter decomposition. doi:10.1007/s003740050430. Normalization and piecewise moisture form as implemented in the canonical reclim package.

Examples

```
soc_rate_modifier_icbm(
  temp_c = c(5, 15, 25),
  theta = c(0.20, 0.25, 0.30),
  t_field = 0.30,
  t_wilt = 0.10,
  porosity = 0.45
)
```

`soc_rate_modifier_lpjml`
Compute the LPJmL annual decomposition response.

Description

Annual mean of LPJmL's per-time-step decomposition response: a modified Arrhenius temperature function after Lloyd and Taylor (1994), multiplied by a cubic function of the soil's degree of saturation, and **capped at 1**.

Three properties matter when reading the result. The temperature term is normalised so that it is exactly 1 at 10 degrees Celsius, which is what the "at 10 degrees" in LPJmL's rate constants refers to. The moisture term is not normalised at all and peaks at 0.937. And the cap is imposed by LPJmL's source rather than by its published equations: it binds from about 10.7 degrees at optimal moisture, so every warm, well-watered cell returns exactly 1 and the fast pool cannot turn over faster than its nominal rate. The cap is applied per time step, before averaging, because capping the mean is a different function.

Usage

```
soc_rate_modifier_lpjml(temp_soil_c, theta)
```

Arguments

`temp_soil_c`	Numeric soil temperature series (degrees Celsius). LPJmL drives this with per-layer **soil** temperature, not air temperature; the two differ in both damping and lag, so supplying air temperature is a substitution the caller is making, not a detail.
`theta`	Numeric series of the soil's degree of saturation (0-1), as LPJmL's fractional soil water content reports it. Use it raw; it is already a saturation fraction, not a volumetric content needing a porosity divisor.

Value

The annual mean of the capped temperature-by-moisture product (a single numeric).

Source

Schaphoff, S. et al. (2018). LPJmL4 - a dynamic global vegetation model with managed land - Part 1: Model description. *Geoscientific Model Development*, 11, 1343-1375. doi:10.5194/gmd1113432018, Eq. 45 (temperature) and Eq. 96 (moisture); temperature form after Lloyd, J. & Taylor, J. A. (1994). doi:10.2307/2389824. The cap at 1 is in LPJmL's source (`src/soil/littersom.c`) and not in the published equations.

Examples

```
soc_rate_modifier_lpjml(
  temp_soil_c = c(2, 12, 22),
  theta = c(0.3, 0.5, 0.6)
)
```

---

soc_rate_modifier_rothc

Compute the RothC and HSOC annual climate rate modifier.

Description

Annual a*b*c rate-modifying factor of the RothC / HSOC soil-carbon model: the temperature factor a, the topsoil-moisture-deficit factor b, and the plant-cover factor c, averaged over the supplied monthly series. The same function modifies HSOC decomposition.

Usage

```
soc_rate_modifier_rothc(
  temp_c,
  water_minus_pet_mm,
  clay_pct,
  soil_cover,
  soil_depth_m = .soc_topsoil_depth_m()
)
```

Arguments

`temp_c`	Numeric monthly air temperature series (degrees Celsius).
`water_minus_pet_mm`	Numeric monthly water surplus series (precipitation minus potential evapotranspiration, mm), used to accumulate the topsoil moisture deficit.
`clay_pct`	Soil clay content (percent).
`soil_cover`	Vegetated soil-cover fraction (0 bare, 1 fully covered); a scalar or monthly series.
`soil_depth_m`	Topsoil depth over which the moisture deficit is accumulated (metres). Defaults to 0.3, the layer every soil-carbon stock in this package is defined over; RothC's published expression is calibrated on its own 0-23 cm layer and is rescaled to this depth.

Value

The annual mean of the monthly a*b*c product (a single numeric).

Source

Coleman, K. & Jenkinson, D. S. (1996). RothC-26.3: a model for the turnover of carbon in soil. doi:10.1007/9783642610943_17. Moisture deficit and cover terms as implemented in an earlier regional historical SOC reconstruction.

Examples

```
soc_rate_modifier_rothc(
  temp_c = c(1.2, 7.8, 15.0),
  water_minus_pet_mm = c(-10, 5, 20),
  clay_pct = 20,
  soil_cover = 0
)
```

---

soc_soil_cover_curve *Generic land-use soil-cover curve for the RothC/HSOC cover factor.*

Description

Monthly vegetated soil-cover fraction (0 bare, 1 fully covered) by land-use class, driving the plant-cover term of the RothC and HSOC decomposition modifier (`soc_rate_modifier_rothc`, where the cover factor is `0.6 + 0.4 * (1 - soil_cover)`: covered soil decomposes more slowly than bare soil). `build_carbon_balance` attaches these values to each cell-month before reducing the monthly climate drivers to an annual modifier, so cropland (a seasonal crop canopy with a bare fallow period) mineralizes soil carbon differently from grassland and natural land (perennial cover). The curve is keyed on the signed month offset from the cell-year's warmest (peak-canopy) month rather than the calendar month, so the same generic curve aligns to the growing season in either hemisphere from the temperature seasonality alone.

For cropland the fractions follow the FAO-56 crop growth-stage canopy development of a generic 150-day annual field crop (the FAO-56 maize-grain template, 30/40/50/30 days for the initial, development, mid-season and late-season stages): a low establishment cover during the initial stage (about 10 percent ground cover), rising through the development stage to an effective full canopy at mid-season (the peak, offset 0), then declining through late-season senescence, with the remaining fallow/off-season months at a low bare-soil cover. Grassland and natural land carry a sustained high perennial cover in every month (no bare fallow period), following the RothC convention that permanent vegetation is treated as covered year-round.

Usage

```
soc_soil_cover_curve
```

Format

A tibble with columns:

land_use Land-use class: `"cropland"`, `"grassland"` or `"natural"` (matched case-insensitively; a class absent from the table, such as `"urban"`, is treated as bare soil).

months_from_peak Signed month offset from the warmest month of the cell-year (integer `-5` to `6`; `0` is the peak-canopy mid-season month).

soil_cover Vegetated soil-cover fraction for that class and month offset (0 bare, 1 fully covered).

Source

Crop canopy development stages and their durations follow the FAO-56 crop coefficient framework: Allen, R. G., Pereira, L. S., Raes, D. & Smith, M. (1998). *Crop evapotranspiration: Guidelines for computing crop water requirements* (FAO Irrigation and Drainage Paper 56). Food and Agriculture Organization of the United Nations, Rome (Chapter 6 and Table 11: initial stage to about 10 percent ground cover, development to effective full cover, full canopy at mid-season, senescence in late-season; generic 150-day annual-crop template). The binary covered-versus-bare cover convention it feeds is that of the RothC model: Coleman, K. & Jenkinson, D. S. (1996). RothC-26.3: a model for the turnover of carbon in soil. doi:10.1007/9783642610943_17 (the soil cover rate-modifying factor slows decomposition when growing plants are present). The perennial grassland/natural cover and the fallow-period bare-soil value are generic modelling assumptions, not a per-crop observational survey.

Examples

```
soc_soil_cover_curve
```

`soc_turnover_params` *Soil organic carbon turnover parameters by model.*

Description

Per-model structural parameters for the five soil organic carbon turnover models ported in Module B: HSOC (Spain two-pool plus inert organic matter), RothC (five pools), ICBM (two pools), AMG (active plus stable), and Century (five pools). Stored in long form so a builder can assemble each model's pool rate constants, initial pool fractions, inter-pool transfer fractions, and texture/lignin defaults. Climate rate modifiers are model-native and live in the turnover functions, not in this table.

Usage

```
soc_turnover_params
```

Format

A tibble in long form with columns:

model Model namespace: one of `"hsoc"`, `"rothc"`, `"icbm"`, `"amg"`, `"century"`.

component Pool or structural component the parameter belongs to (e.g. `"fresh"`, `"dpm"`, `"young"`, `"active"`, `"str"`, `"defaults"`).

parameter Parameter name (e.g. `"decomposition_rate"`, `"init_active_fraction"`, `"transfer_fraction"`, `"base_rate_weekly"`).

value Numeric parameter value.

unit Unit of the value (e.g. `"per_year"`, `"fraction"`, `"per_week"`).

description Human-readable description of the parameter.

Source

RothC: Coleman, K. & Jenkinson, D. S. (1996). RothC-26.3: A model for the turnover of carbon in soil. In D. S. Powlson et al. (Eds.), *Evaluation of Soil Organic Matter Models* (NATO ASI Series I, Vol. 38, pp. 237-246). Springer. doi:10.1007/978364261094-3_17. ICBM: Andren, O. & Katterer, T. (1997). ICBM: The introductory carbon balance model for exploration of soil carbon balances. *Ecological Applications*, 7(4), 1226-1236. doi:10.1890/10510761(1997)007[1226:ITICBM]2.0.CO;2. AMG: Saffih-Hdadi, K. & Mary, B. (2008). Modeling consequences of straw residues export on soil organic carbon. *Soil Biology and Biochemistry*, 40(3), 594-607. doi:10.1016/j.soilbio.2007.08.022. Century: Parton, W. J., Schimel, D. S., Cole, C. V. & Ojima, D. S. (1987). Analysis of factors controlling soil organic matter levels in Great Plains grasslands. *Soil Science Society of America Journal*, 51(5), 1173-1179. doi:10.2136/sssaj1987.03615995005100050015x; SoilR Century implementation: Sierra, C. A., Mueller, M. & Trumbore, S. E. (2012). Models of soil organic matter decomposition: the SoilR package, version 1.0. *Geoscientific Model Development*, 5, 1045-1060. doi:10.5194/gmd510452012. Inert organic matter initialisation (RothC, HSOC): Falloon, P., Smith, P., Coleman, K. & Marshall, S. (1998). Estimating the size of the inert organic matter pool from total soil organic carbon content for use in the Rothamsted carbon model. *Soil Biology and Biochemistry*, 30(8-9), 1207-1211. doi:10.1016/S00380717(97)002563.

Examples

```
soc_turnover_params
```

`soil_cn_ratios` *Soil carbon-to-nitrogen ratios for organic-matter balances.*

Description

Soil carbon-to-nitrogen ratios used to convert a soil organic carbon stock change into net nitrogen mineralization (when carbon is lost) or net nitrogen sequestration (when carbon accumulates), by cropland class and management system.

Usage

```
soil_cn_ratios
```

Format

A tibble with columns:

cropland_class Land class: `"Cropland"` or `"NonCropland"`.

management Management system: `"Conventional"` or `"Organic"`. **Only the `"Conventional"` rows are selectable by this package.** `build_carbon_balance()` has no management dimension – it is a global gridded balance – so an argument for this column could only be set world-wide, and "run the whole world as organic" is not a run anyone should perform. The `"Organic"` rows therefore ship as *reference values for downstream consumers*, not as inputs this package can be asked to use (issue 809).

cn_ratio Soil organic-matter carbon-to-nitrogen ratio.

cn_mineralization Carbon-to-nitrogen ratio applied when soil organic carbon is mineralized (net carbon loss).

cn_sequestration Carbon-to-nitrogen ratio applied when soil organic carbon is sequestered (net carbon gain).

What the asymmetry represents

The two directional ratios are not two process stoichiometries. They express the FLEXIBILITY of soil C:N: a soil gaining carbon is expected to move to a wider ratio, a soil losing carbon to a narrower one, so the pair brackets the bulk value. Cropland is 8 – 10 – 11 and non-cropland 11 – 15 – 15.

This is why both are documented as applying to the *net* change: the ratio describes where a given soil's C:N is heading, which is a property of the soil, not of an individual crop or field on part of it. Applying the choice at a finer grain than the soil makes the two directions non-cancelling and manufactures nitrogen at a soil whose carbon did not change.

The bounds themselves remain unsourced; see the source section below. What is wanted is a published *marginal* C:N – the ratio of the CHANGE in soil carbon to the change in soil nitrogen – not a bulk soil C:N, which is a different quantity.

Source

Expert parameterisation with a documented rationale and no citation. Traced through the coefficient workbook of the earlier regional historical reconstruction the values come from, whose C:N sheet has no source column and no notes column, as does the packaged `inst/extdata/balances/soil_cn_ratios.csv`:

- The bulk ratios (`cn_ratio` 10 cropland, 15 non-cropland) enter the workbook on 2025-03-14, in a sheet holding only `Cropland_class` and `CN_ratio`. The workbook's earliest tracked version (2021-02-19) has no such sheet at all.
- The asymmetry – management stratification plus `cn_mineralization` and `cn_sequestration` – is added on 2026-03-26, together with an N-limitation on SOC sequestration.

The rationale recorded with the values is: a lower ratio on mineralization for the microbial-biomass and labile pools that are preferentially decomposed, a higher one on sequestration for stable humus formation, and organic cropland sequestering at 13 against 11 conventional for the larger stable-humus fraction of manure- and compost-derived carbon. No citation is attached to any of the values.

One published anchor exists nearby and is worth knowing: the source's methods cite Cleveland & Liptzin (2007), doi:10.1007/s1053300791320, for a soil microbial-biomass C:N of roughly 8-13, and the cropland `cn_mineralization` values (8 and 9) fall inside it. It is cited there for a different parameter, and it is not offered here as the source of these numbers – only as the nearest published range they are consistent with.

This entry previously cited Coleman & Jenkinson's RothC (1996) as the framework the values are "consistent with". That attribution has been removed: RothC is a carbon-only model and sets no carbon-to-nitrogen ratio at all – `calculate_soc_rothc` returns `dpm`, `rpm`, `bio`, `hum`, `iom` and `soc_total`, and no nitrogen anywhere – so no RothC reference can be a source for these numbers. Naming one made an undocumented coefficient set read as a sourced one (whep#346).

The values move published nitrogen: every soil-carbon stock change is divided by one of them to reach mineralized or sequestered nitrogen. Whether to cite Cleveland & Liptzin for the mineralization ratios, and what to cite for the rest, is whep#346 and still open.

Examples

```
soil_cn_ratios
```

`soil_hydraulic_by_texture` *Soil hydraulic properties by USDA texture class.*

Description

Class-average volumetric soil water contents by USDA texture class: total porosity (saturation), field capacity (33 kPa) and permanent wilting point (1500 kPa), each as a volumetric fraction (cubic metre water per cubic metre soil). These are the per-cell soil hydraulic drivers the ICBM soil-carbon moisture modifier consumes (`soc_rate_modifier_icbm()`'s `t_field`, `t_wilt` and `porosity`); `read_soil_hydraulic()` joins this table onto the dominant HWSD texture class of each grid cell to emit those columns, which then feed `get_soc_climate_drivers()`. Every row satisfies `porosity > field_capacity > wilting_point` with all three in `(0, 1)`.

Usage

```
soil_hydraulic_by_texture
```

Format

A tibble with columns:

usda_texture_class USDA texture class (snake_case): `"sand"`, `"loamy_sand"`, `"sandy_loam"`, `"silt_loam"`, `"silt"`, `"loam"`, `"sandy_clay_loam"`, `"silty_clay_loam"`, `"clay_loam"`, `"sandy_clay"`, `"silty_clay"` or `"clay"`.

porosity Total porosity / saturated volumetric water content (fraction).

field_capacity Volumetric water content at field capacity, 33 kPa (fraction).

wilting_point Volumetric water content at the permanent wilting point, 1500 kPa (fraction).

Source

Class-average hydraulic properties compiled by texture class in "Average hydraulic properties of ARS soil texture classes" (Schaake, J., draft February 2000; 2128 soil samples), the values used as the default soil-texture lookup in the VIC land-surface model (https://vic.readthedocs.io/en/master/Documentation/soiltext/). The underlying regression relating the water-retention parameters to texture is Cosby, B. J., Hornberger, G. M., Clapp, R. B. & Ginn, T. R. (1984). A statistical exploration of the relationships of soil moisture characteristics to the physical properties of soils. *Water Resources Research*, 20(6), 682-690. doi:10.1029/WR020i006p00682.

Examples

```
soil_hydraulic_by_texture
```

`som_ranges` *Soil organic matter content bins.*

Description

Half-open bins that map a soil organic matter share (fraction) to a soil organic matter class. A share `s` is assigned to the class whose interval satisfies `s_min < s <= s_max`; the top (`High`) bin has no finite ceiling in the source and is treated as open-ended (any share above its `s_min` is `"High"`), so `s_max = 1` here is a wide ceiling, not a literal source value. The classes key the Meisinger denitrification matrix (meisinger_denitrification).

Usage

```
som_ranges
```

Format

A tibble with columns:

som_content Soil organic matter class: `"High"`, `"Medium"` or `"Low"`.

som_min Lower bound of the soil organic matter share interval (fraction).

som_max Upper bound of the soil organic matter share interval (fraction); the `"High"` row's value is a wide ceiling, not a literal source bound (see Description).

Source

Nitrogen coefficient workbook of an earlier regional historical reconstruction, companion to the Meisinger & Randall (1991) denitrification matrix. doi:10.2136/1991.managingnitrogen.c5.

Examples

```
som_ranges
```

---

spatialize_country_n_to_crops

Spatialize a polity-level nitrogen total to crops and grid cells.

Description

Promotes a single polity-total nitrogen input (one row per `year`, `area_code`, for one fertiliser type) to the crop level by the same harvested-area-share logic used by `build_crop_soil_n2o_extension()` (`year`/`area_code`-matched, weighted by each crop's share of harvested cropland area), then optionally further to the grid level by distributing each polity-crop total across cells in proportion to the cell's share of that polity-crop's total crop-pattern area (`type_ha * harvest_fraction`, summed over LUH2 cropland classes; the exact formula used by `make_lpjml_covariate()`'s `crop_pattern` weighting). A polity-crop with no crop-pattern hectares (crop absent from the static pattern raster, or its pattern area sums to zero in the polity) is instead spread uniformly across the polity's cropland cells, weighted by each cell's cropland area, so the grid output still re-aggregates to the polity total. Such reallocations emit a warning naming the affected crops and the reallocated nitrogen.

Both grid weights carry a polity share of the cell, because a border cell's cropland belongs to more than one polity. That share is a polycell's `land_area_ha` over the cell's total `land_area_ha`, the share of the cell's land `build_polycell_support()` measures geodesically. The transitional alternative is `polity_frac`, the subcell-count share `build_cell_polity()` carries, which is quantised to 1/36 of a cell.

The share is of the cell's **land**, never `land_area_ha / cell_area_ha`: what it splits is already land-only (LUH2 cropland hectares), so dividing by the whole cell would remove the inland water and ice a second time. And it is a **share**, never an absolute area: `crop_pattern_ha * land_area_ha` would weight every cell a second time by its own size and is no partition of the cell at all.

Because the cell weights are renormalised within each polity-crop-year, either key redistributes the polity total and neither can change it. Swapping the key moves **where** the nitrogen lands, never how much of it there is.

Usage

```
spatialize_country_n_to_crops(
  country_totals,
  crop_shares,
  cell_polity,
  resolution = c("polity_crop", "grid"),
```

```
  split = c("auto", "land_area_ha", "polity_frac"),
  polity_validity = c("keep", "flag", "drop"),
  data = list()
)
```

Arguments

`country_totals`
: A tibble with `year`, `area_code`, `n_t`: the polity-level nitrogen total for one fertiliser type.

`crop_shares`
: A tibble with `year`, `area_code`, `item_cbs_code`, `area_share`: harvested-area-weighted crop shares within each country-year, e.g. from `build_crop_soil_n2o_extension` internal crop-area-share helper.

`cell_polity`
: The cell-to-polity support (`lon`, `lat`, `area_code`, `cell_area_ha` and the `split` key column). Only required when `resolution` includes `"grid"`. Rows are keyed on `area_code`: `build_polycell_support()` keys on `polity_code` and does not derive the reporting vocabulary (DA-23), and `polity_area_crosswalk` folds distinct polities into one `area_code` or leaves it `NA`, so a support must be converted to one row per cell and `area_code` **before** it is passed here. That conversion is refused rather than performed silently. The share denominator is the land of the rows supplied, so a caller that simply drops the unkeyable polycells renormalises the survivors onto the whole cell and hands them hectares that are not theirs, invisibly, because the shares still sum to 1. Keep that land in the denominator instead, under a bucket no `country_totals` row can join: it then dilutes nobody's share and receives no nitrogen.

`resolution`
: Which resolution(s) to return: `"polity_crop"` (default, `year`/`area_code`/`item_cbs_code` totals only) or `"grid"` (also distributes to `lon`/`lat` grid cells; requires `crop_patterns` and `type_cropland` in `data`).

`split`
: Which polity share of the cell weights each grid cell: `"auto"` (default) takes `land_area_ha` when the support carries it and `polity_frac` otherwise; `"land_area_ha"` and `"polity_frac"` demand that key and abort when it is absent. The resolved key is recorded in the `method_polity_split` output column, so a table's split is readable from the table. Read only at `resolution = "grid"`; the `"polity_crop"` output splits nothing across cells.

`polity_validity`
: What to do with a row whose `(area_code, year)` resolves to a polity that did not exist in that year (the cell-polity crosswalk has no year dimension, so an early-20th-century cell is labelled with its present-day territory). `"keep"` (default) keeps every row, which is the historical behaviour, and warns naming the rows, years and area codes involved. `"flag"` keeps them and adds the per-row logical `reporting_polity_out_of_span`, marking exactly which rows are stand-ins. `"drop"` removes them. All three warn; only `"drop"` changes the numbers. See `polity_coverage_gaps()`, which reports the same rows for an already-built table.

`data`
: Optional named list of pre-loaded grid inputs, used only when `resolution = "grid"`: `crop_patterns` (`lon`, `lat`, `item_prod_code`, `harvest_fraction`)

and `type_cropland` (`lon`, `lat`, `year`, `luh2_type`, `type_ha`). When absent, each is read from the `spatialize-crop-patterns` / `spatialize-type-cropland` pin, or from the local parquet at `WHEP_CROP_PATTERNS_PATH` / `WHEP_TYPE_CROPLAND_PATH` when that override is set. `item_cbs_code` in `crop_shares`/`country_totals` is matched to the `item_prod_code` column of `crop_patterns` via items_prod_full (the same crosswalk `build_crop_land_extension()` uses).

Value

A tibble. For `resolution = "polity_crop"`: `year`, `area_code`, `item_cbs_code`, `n_t`. For `resolution = "grid"`: `lon`, `lat`, `area_code`, `year`, `item_cbs_code`, `n_t` and `method_polity_split`. Either gains `reporting_polity_out_of_span` when `polity_validity = "flag"`; this output carries no reporting-polity columns, so the flag is attached directly rather than derived from them.

Examples

```
spatialize_country_n_to_crops(
  country_totals = tibble::tribble(
    ~year, ~area_code, ~n_t,
    2010L, 10L, 100
  ),
  crop_shares = tibble::tribble(
    ~year, ~area_code, ~item_cbs_code, ~area_share,
    2010L, 10L, 2511L, 0.7,
    2010L, 10L, 2513L, 0.3
  ),
  cell_polity = NULL,
  resolution = "polity_crop"
)
```

---

split_manure_management

Split livestock excretion across manure-management systems.

Description

Splits the excreted nitrogen, carbon and volatile solids from `estimate_n_excretion()` across manure-management systems (MMS), separating the in-situ grazing stream (pasture/range/paddock, deposited where it falls) from the collected/housed streams routed to storage. The split conserves mass: the per-species MMS shares sum to one.

Usage

```
split_manure_management(excretion, options = list())
```

Arguments

excretion	A tibble from estimate_n_excretion() with year, territory (a stringified area_code, see estimate_n_excretion()), sub_territory, livestock_category, n_excretion, c_excretion and vs_excretion.
options	A named list. mms_source selects how the MMS shares in regional_mms_distribution are read: • "regional_default" (default): every territory takes the table's region == "Global" rows. • "region_specific": each territory takes the rows of the region it resolves to, and the Global rows when its region has none. mms_shares selects which half of regional_mms_distribution is read: "gleam_2_0" (default), the GLEAM 2.0 Supplement S1 Tab. 4.2-4.11 ingest, or "placeholder", the unsourced table it replaced (whep#958).

Value

A tibble with one row per year x territory x sub_territory x livestock_category x mms_type, plus species_gen, loss_category, stream ("grazing" or "collected"), n_stream, c_stream, vs_stream and method_mms.

Examples

```
excretion <- tibble::tribble(
  ~year, ~territory, ~sub_territory, ~livestock_category,
  ~n_excretion, ~c_excretion, ~vs_excretion,
  2020L, "203", NA, "Cattle_milk", 100, 1900, 60,
  2020L, "203", NA, "Pigs", 30, 270, 20
)
split_manure_management(excretion)
```

---

split_regime_yield *Split a cell-crop's production between its rainfed and irrigated regimes.*

Description

Applies the irrigated:rainfed yield ratio R of build_regime_yield_ratio() to a cell-crop's production P and its rainfed and irrigated harvested areas A_r, A_i, keeping the production: Y_r = P / (A_r + R * A_i) and Y_i = R * Y_r, so A_r * Y_r + A_i * Y_i = P.

R starts from ratio_unbounded and two plausibility bounds can lower it, never below 1. The result is the output's ratio, the only ratio fit to weight anything:

- **Irrigated-yield ceiling.** Where Y_i would exceed the item's Y_max, R is lowered until Y_i = Y_max, R = Y_max * A_r / (P - Y_max * A_i). Y_max is the 99th percentile of the item's national yields (production over harvested area) in get_primary_production(), pooled over countries and the years 1961-2023.

- **Rainfed-yield floor.** Where `Y_r` would fall below the item's `Y_min`, the 1st percentile of the same pool, `R` is lowered until `Y_r = Y_min`, `R = (P - Y_min * A_r) / (Y_min * A_i)`.

Lowering `R` raises `Y_r` and lowers `Y_i`, so the floor, applied after the ceiling, cannot break it. An item with no FAOSTAT yields of its own takes the pooled yields of the items that directly carry one of its SPAM crops in regime_yield_crop_mapping (`spam_basis "direct"` or `"direct_aggregate"`), stamped `"bound_via_spam_crop"`.

A cell-crop with area in one regime only is a trivial split: all its production is on that regime, no ratio applies (`ratio` is 1, whatever `ratio_unbounded` is) and the bounds are not tested. Only a cell-crop with no area at all, or with both regimes and no `ratio_unbounded`, gets `NA`.

Usage

```
split_regime_yield(cells, bound = c("global", "region"), production = NULL)
```

Arguments

`cells`	A tibble with `area_code`, `item_prod_code`, `production_t` (production in the units of `get_primary_production()`'s `"tonnes"` rows), `rainfed_ha`, `irrigated_ha` and `ratio_unbounded` (as `build_regime_yield_ratio()` returns it). Other columns are kept.
`bound`	Which pool of national yields sets `Y_max` and `Y_min`: `"global"` (default, all countries) or `"region"` (the countries of the cell's WHEP region, the `region` column of regions_full; where the region has no yields for the item, the world's pool, stamped `_global`).
`production`	Optional `get_primary_production()` output (`year`, `area_code`, `item_prod_code`, `unit`, `value`) used instead of building it.

Value

`cells` with:

- `ratio`: the irrigated:rainfed yield ratio used, after the bounds (1 on a trivial split).
- `yield_rainfed`, `yield_irrigated`: production per hectare of each regime.
- `yield_max`, `yield_min`: the bounds applied; `NA` where the item has no yields to set them.
- `method_regime_split`: `"yield_ratio"`, `"trivial_rainfed_only"`, `"trivial_irrigated_only"`, `"no_area"` or `"no_ratio"`.
- `method_regime_bound`: the ceiling: `"not_binding"`, `"clipped"`, `"clipped_at_one"` (lowered to 1 and still above `Y_max`: the mean yield itself exceeds it), `"no_bound"`, `"not_applicable_trivial"` or `"not_applicable"` (no area, or no ratio).
- `method_rainfed_floor`: the floor: `"not_binding"`, `"rainfed_floor"`, `"rainfed_floor_at_one"` (lowered to 1 and still below `Y_min`: the mean yield itself is below it), `"no_bound"`, `"not_applicable_trivial"` or `"not_applicable"`.
- `method_bound_source`: which yields set the bounds: `"own_yields"`, `"bound_via_spam_crop"`, either suffixed `"_global"` where a regional pool fell back to the world's, or `"none"`.
- `method_yield_bound`: the `bound` chosen.

Examples

```
cells <- tibble::tribble(
  ~area_code, ~item_prod_code, ~production_t, ~rainfed_ha, ~irrigated_ha,
  ~ratio_unbounded,
  203L, 15L, 500, 100, 50, 1.8
)
production <- tibble::tribble(
  ~year, ~area_code, ~item_prod_code, ~unit, ~value,
  2010L, 203L, 15L, "tonnes", 3000,
  2010L, 203L, 15L, "ha", 1000
)
split_regime_yield(cells, production = production)
```

---

stamp_inputs_supplied

Record which optional inputs a build consumed

Description

Write the provenance stamp `check_inputs_supplied()` reads: a character column naming the optional inputs this build actually had, comma-separated and sorted, or `"none"` when it had none. The point of the stamp is that it is a **label**: unlike a total, it cannot be satisfied by arithmetic, so a consumer can tell an absent layer from a measured zero years later and without the producer's arguments in hand.

Usage

```
stamp_inputs_supplied(data, supplied, column = "inputs_supplied")
```

Arguments

`data`	A tibble to stamp.
`supplied`	Keys of the inputs that were supplied. May be empty.
`column`	Name of the stamp column.

Value

`data` with `column` added or overwritten.

Examples

```
stamp_inputs_supplied(tibble::tibble(x = 1:2), c("ice", "water"))
stamp_inputs_supplied(tibble::tibble(x = 1:2), character())
```

`subsoil_no3_reduction`

Subsoil nitrate reduction shares by source, climate and irrigation.

Description

Fraction of leaching nitrate reduced (denitrified) in the subsoil below the rooting zone, by nitrogen source, climate zone and irrigation category. Applied after topsoil denitrification to compute the nitrate that reaches groundwater.

Usage

```
subsoil_no3_reduction
```

Format

A tibble with columns:

fert_type Nitrogen source: one of `"Synthetic"`, `"SOM"`, `"Deposition"`, `"Solid"`, `"Excreta_cattle_monog"`, `"Excreta_other"`, `"Liquid"`, `"Human"`, `"BNF"`. `"Human"` was keyed `"Urban"` before it was renamed; the former key is still read, with a deprecation warning.

climate Climate zone: `"MED"` or `"ATL"`.

irrig_cat Irrigation category: `"Rainfed"` or `"Irrigated"`.

no3_red Fraction of leaching nitrate reduced in the subsoil.

Source

Nitrogen coefficient workbook of an earlier regional historical reconstruction, parameterised from Mediterranean and Atlantic subsoil denitrification literature consistent with the IPCC (2019) indirect N2O framework.

Examples

```
subsoil_no3_reduction
```

`summarise_conservation`

Summarise a footprint conservation report.

Description

Roll up the per-origin report from `check_footprint_conservation()` into a single global verdict.

Usage

```
summarise_conservation(report)
```

Arguments

report	Tibble returned by check_footprint_conservation().

Value

A one-row tibble with n_origin, n_ok, n_flagged, n_dropped, n_under_traced, n_over_traced, total_direct, total_embodied and global_rel_discrepancy.

Examples

```
z_mat <- matrix(c(0, 5, 10, 0), nrow = 2)
x_vec <- c(100, 200)
y_mat <- matrix(c(85, 195), ncol = 1)
extensions <- c(50, 30)
labels <- tibble::tibble(
  area_code = c(1L, 1L),
  item_cbs_code = c(1L, 2L)
)
compute_footprint(
  x_vec = x_vec, y_mat = y_mat, extensions = extensions,
  labels = labels, z_mat = z_mat
) |>
  check_footprint_conservation(extensions, labels, x_vec) |>
  summarise_conservation()
```

---

summarize_bnf *Summarise biological nitrogen fixation results.*

Description

Aggregates the per-row BNF components into group totals, shares and mean modifiers.

Usage

```
summarize_bnf(x, group_by = "item_prod_code")
```

Arguments

x	A tibble with crop_bnf_t, weed_bnf_t, nonsymbiotic_bnf_t and bnf_t (the output of calculate_bnf()).
group_by	Character vector of grouping columns (default "item_prod_code"); use NULL for an overall summary.

Value

A tibble with per-group counts, BNF totals, component percentages and mean environmental factors.

Examples

```
tibble::tibble(
  item_prod_code = "176", crop_npp_n_t = 10, product_n_t = 5,
  weed_npp_n_t = 4, land_use = "Cropland", legumes_seeded = 0,
  seeded_cover_crop_share = 0, area_ha = 40
) |>
  calculate_bnf() |>
  summarize_bnf()
```

---

temperature_adjustment

Temperature adjustment factors for NEm.

Description

Adjustment multipliers for net energy maintenance under cold stress, thermoneutral, and heat stress conditions.

Usage

```
temperature_adjustment
```

Format

A tibble with `temp_range`, `temp_min`, `temp_max`, `adjustment_factor`.

Source

NRC 2001; IPCC 2019.

Examples

```
temperature_adjustment
```

uncertainty_ranges *Uncertainty ranges for emission parameters.*

Description

Lower and upper multipliers for key emission parameters (Ym, MCF, Bo, EF_N2O, Nex).

Usage

```
uncertainty_ranges
```

Format

A tibble with `parameter`, `lower_mult`, `upper_mult`, `distribution`.

Source

IPCC 2019 Refinement, Vol 4, Ch 10.

Examples

```
uncertainty_ranges
```

unit_cropland_extent *The cropland each unit holds, on the basis the capacity ceiling uses*

Description

Sum a gridded cropland extent onto the units of an allocation layer, weighting every cell by `cell_area_frac`. This is the denominator `reconcile_admin_allocation()`'s implied cropping intensity divides by, and it is deliberately the same quantity `.capacity_bases()` multiplies by the multi-cropping factor: an intensity taken against any other land basis would not be comparable with the ceiling it is flagged against.

A physical cell shared by two units contributes its cropland to each in proportion, so the units' extents inside one cell add up to that cell's own cropland and nothing is double counted. Rows are selected per year by the layer's own validity convention, so an interval-keyed layer returns the unit that existed in each cropland year.

Usage

```
unit_cropland_extent(allocation_layer, gridded_cropland)
```

Arguments

`allocation_layer`
: The country grid the allocation ran on, from `build_allocation_layer()`: `lon`, `lat`, `area_code`, `cell_area_frac` and, where a depth is granted, `level_polity_code`.

`gridded_cropland`
: Per-cell cropland extent, as `build_gridded_landuse()` takes it: `lon`, `lat`, `year`, `cropland_ha`.

Value

A tibble of `year`, `area_code`, `level_polity_code`, `cropland_ha` and `n_cells`.

See Also

`reconcile_admin_allocation()`.

Examples

```
layer <- tibble::tibble(
  lon = c(0.25, 0.75, 0.75), lat = 50.25, area_code = 1L,
  level_polity_code = c("A1", "A1", "A2"), level = 1L,
  cell_area_frac = c(1, 0.25, 0.75)
)
cropland <- tibble::tibble(
  lon = c(0.25, 0.75), lat = 50.25, year = 2000L,
  cropland_ha = c(100, 400)
)
unit_cropland_extent(layer, cropland)
```

---

validate_national_trade

Validate national net trade balance

Description

Validates the provincial GRAFS model by comparing Spain's national net trade computed bottom-up (summing provincial net balances per item) against official FAOSTAT figures. Because internal inter-province flows cancel out pairwise, the sum of provincial net balances equals Spain's true international net trade.

Usage

```
validate_national_trade(n_prov_destiny = NULL, example = FALSE)
```

Arguments

n_prov_destiny	Optional pre-computed output from `create_n_prov_destiny()`. If `NULL`, calls that function internally.
example	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble with columns `year`, `item`, `net_prov`, `net_fao`, and `diff_net` (all in MgN).

Examples

```
validate_national_trade(example = TRUE)
```

---

validate_national_trade_raw

Validate national net trade against raw historical FAO series

---

Description

Validates the provincial GRAFS model's bottom-up national net trade against the original historical Export/Import series for Spain contained in `Europe_FAO_completed.xlsx` (1849-1960), the raw source data behind the package's processed trade figures. Comparison is restricted to the item and year combinations actually reported in the raw sheets, since coverage there is sparser than in the processed dataset.

Usage

```
validate_national_trade_raw(n_prov_destiny = NULL, example = FALSE)
```

Arguments

n_prov_destiny	Optional pre-computed output from `create_n_prov_destiny()`. If `NULL`, calls that function internally.
example	If `TRUE`, return a small hardcoded output without downloading remote data. Default is `FALSE`.

Value

A tibble with columns `year`, `item`, `net_prov`, `net_fao`, and `diff_net` (all in MgN).

Examples

```
validate_national_trade_raw(example = TRUE)
```

`whep_clear_cache` *Clear the build pipeline cache*

Description

Removes cached results from `build_primary_production()`, `build_commodity_balances()`, and `build_processing_coefs()` so that the next call rebuilds from scratch.

Usage

```
whep_clear_cache()
```

Value

Invisible `NULL`.

Memory that clearing the cache does not return

Clearing the cache releases the cached tibbles to R, but the process may keep far more resident memory than it holds. That excess is not held by the cache or by any reachable object, so neither this function nor `gc()` recovers it (whep#777).

It is memory R has already freed and the C allocator keeps. On Linux, glibc serves a large allocation with its own mapping, which goes back to the operating system on free, only while it is bigger than the "mmap threshold". Each such free raises that threshold (up to 32 MB), so later vectors below it come from the heap, and freed heap blocks stay resident. The build chain frees many vectors of that size.

Measured with `get_primary_production()` and `get_wide_cbs()` for 2008-2012, after `gc()`, holding 1.17 GB of live R objects: 5.5-5.8 GB resident by default, 3.3 GB with a fixed threshold, identical outputs. Fixing the threshold switches the raising off. glibc reads it when the process starts, so it must be set in the shell that launches R, not in `.Renviron`:

```
GLIBC_TUNABLES=glibc.malloc.mmap_threshold=131072 Rscript build.R
```

Other platforms and allocators are unaffected by the variable.

Examples

```
whep_clear_cache()
```

whep_coef_table *Read a WHEP coefficient table.*

Description

Reads one of the coefficient tables shipped as CSV under `inst/extdata/coefs`. These tables are small and versioned inside the package (read at runtime, not stored remotely), so no download is needed. The long `bnf_provenance` sidecar records cell-level evidence for every non-missing numeric value in `bnf`; it documents the runtime table but does not replace it as a coefficient input.

Usage

```
whep_coef_table(name)
```

Arguments

`name`	Coefficient table name (the file stem), for example `"bio_coefs"`, `"bnf"` or `"bnf_provenance"`.

Value

A tibble with the coefficient table.

Examples

```
whep_coef_table("residue_feed_fraction")
```

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()`.

Frozen predecessor-pipeline references

Four aliases in `whep_inputs` are not outputs of this package: `primary_prod`, `commodity_balance_sheet`, `processing_coefs` and `feed_intake`. They are the 2025-07-14 snapshot of the predecessor R-script pipeline this package replaced, kept frozen so `inst/scripts/compare_global_whep.R` can benchmark `whep` against it. No package function reads them, their schema is the old one (CamelCase, name-keyed, no polity columns) and they stop in 2021, so reading one warns.

Their numbers are not what current code produces, so they are references, not substitutes. `primary_prod` (2,443,516 rows, 1961–2021) against a fresh `build_primary_production()` over the same 2015–2021 window:

- The pin has no `slaughtered_heads` row in any of its years; the build emits 12,195 over 2015–2021. Three `item_prod` codes are pin-only (378, 773, 1163; 1,332 rows) and three appear in no pin year at all (1051, and the modelled grassland items 3001 and 3002).
- 317,587 rows against the build's 328,242 for the window, a 3.2% shortfall.
- Its 2020–2021 fodder harvested area is carried forward from 2019 — 85.93 Mha in each year, 468 country-item series over 96 areas, equal to 2019 to the last digit — where the build emits no fodder row at all after 2019, because `eu-agridb-fodder` stops in 2019, `faostat-production-old` in 2013, and `faostat-production` carries none of the 16 fodder items.
- Even in a year both cover they disagree: 2019 fodder harvested area is 85.93 Mha in the pin against 90.42 Mha from the build.

Build the current series with the matching `build_*()` or `get_*()` function instead.

The batch pin on the build path

Two further aliases were published in that same 2025-07-14 batch — `crop_residues` and `bilateral_trade`, alongside the four above, between 12:33:43Z and 12:33:50Z. Unlike the four, these two are read on the default build path, which is why no warning is attached to them: every commodity balance build reads them, so a warning at the read is noise rather

than information. `bilateral_trade` has since been regenerated by this package (#1122), so `crop_residues` is the only one still registered at its 2025-07-14 version.

They did **not** share a provenance despite sharing a timestamp. Each was established separately (#1054), because the timestamp alone establishes nothing.

`crop_residues` is predecessor output:

Read by `get_primary_residues()`, and from there by `build_commodity_balances()`. All 475,688 of its `Product` rows equal the `primary_prod` pin's `tonnes` values exactly — no key unmatched on either side, no value differing at a relative tolerance of 1e-6 — so it is a downstream artifact of the same predecessor run, carrying that run's production series into the commodity balance.

Its residue quantities are those production numbers times a residue-to-product ratio that varies by year, and times a **recovery rate**: the pin holds the residue the predecessor recovered from the field, not the residue produced (#1195). The predecessor's code (`Global/R/crop_npp.r`, then `afsetools::residue_use()`) writes each row as

`product * kg_residue_kg_product_FM * HI_changes(region, year) * residue_dm_product_dm(region, category) / residue_dm_product_dm("West Europe", category) * recovery_rates(region, category)`,

with region and category keyed as in `residue_recovery.csv`. Rebuilt from the predecessor's own coefficient tables, that formula reproduces all 425,934 pin rows whose region it can key, at a relative tolerance of 1e-6. The year variation is `HI_changes` alone: a harvest-index change factor per region, given at eight anchor years from 1910 to 2000, linear in between and flat after 2000 (East Europe 1.10 in 1962 to 1.00 in 2000), which the predecessor attributes to Krausmann et al. (2013), *PNAS* 110:10324, Table M2 (assumed, unverified). It is a regional historical adjustment, not a yield response, and after 2000 the ratio of every region and category is constant. That is why `biomass_coefs$kg_residue_kg_product_FM` alone reproduces only 3,189 of the 472,790 comparable keys. Of the 33,411 keys that carry a residue of exactly 0, 30,620 are cells whose recovery rate is 0 (fodder crops in seven of the eight regions; roots and tubers, cassava, sugar beet and dry beans in West Europe and in North America and Oceania) and 1,919 are crops whose `biomass_coefs` ratio is 0.

WHEP ships the recovery rate and the regional ratio the predecessor used (both equal in all 160 cells), but not the `HI_changes` table. So `get_primary_residues()` undoes the recovery exactly, dividing by the same rate, but cannot recompute the predecessor's residue from production. It then re-keys the regional ratio on Wirsenius's own region membership, and estimates cereal residue (#1448), and that of the non-cereal crops with their own IPCC Table 11.2 line (#1518), from the pin's production and harvested area instead of reading it.

It is not a small input. The pin holds 7.63 Gt of recovered residue for 2010 (Straw 3.60 Gt, Other crop residues 2.49 Gt, Firewood 1.54 Gt) and 327.7 Gt over 1961–2021, which `get_primary_residues()` turns into 7.64 Gt (Straw 3.92 Gt, Other crop residues 2.82 Gt, Firewood 0.90 Gt) and 366 Gt of residue produced. 3,998 of its 249,095 output rows carry `NA` polity columns because the pin is name-keyed.

Driving the same residue model off a fresh `get_primary_production()` would move those numbers. Measured for 2010 at (`area_code`, `item_prod`): 1.958 Gt of current production sits on 6,425 keys the pin never sees (809 Mt of it primary crops, including the modelled temporary-grassland item), 64.8 Mt on 296 keys is pin-only, and of the 8,010 shared keys 538 disagree, putting 1.102 Gt — 11.01% of the pin's shared-key mass — more than 1%

apart. The largest class is rice, 673.9 Mt in the pin against 463.3 Mt fresh, a ratio of 0.6876: current code puts rice on a milled-equivalent basis while the pin's is paddy, which is a basis difference rather than an error, since straw scales with the field crop. The second is fodder, where the current build is close to twice the pin on every forage and silage item, the pin predating that work. Replacing the pin therefore means choosing a residue model, which is a science decision and not a refresh. For cereals that model is chosen by `get_primary_residues(cereal_residue = )`, and for the non-cereals with an IPCC line by `non_cereal_residue =`; both still run on the pin's production.

`bilateral_trade` is built by `build_detailed_trade()`:

Read by `get_bilateral_trade()` and by the live-animal branch of `build_commodity_balances()`. The registered version, `20261001T135651Z-ca7b3`, is `build_detailed_trade()`'s output verbatim, with its defaults and `extend_time = FALSE`, run on the `faostat-trade-bilateral` pin `20260407T095142Z-b3f81` (#1122): 10,038,220 rows over 1986–2021, polity-keyed, carrying 64,450,133,319 tonnes and 87,840,869,640 head.

The 2025-07-14 version it replaces, `20250714T123347Z-2c392`, held the same FAOSTAT Detailed Trade Matrix values in the predecessor pipeline's name-keyed schema — against the raw pin over 2010, all 296,642 shared `tonnes` keys and all 5,102 shared `Head` keys agree exactly — but without FAOSTAT's `1000 Head` rows: 76,141,882 thousand head of live broiler chicken, turkey, duck, rabbit and goose trade over 1986–2021, which PR #1113 recovered in the producer. The regenerated pin carries 7.50x its 11,707,083,640 head and 22.5% less than its 83,147,095,412 tonnes, all of that tonnage difference in CBS items 5001-5003, which no commodity balance carries: item 1293's unbacked tonnage, screened by `method_unbacked_quantity` (#1023), and three CBS names with no `item_cbs_code`. On the tonnes that do reach a trade matrix the two versions differ by 0.056%. `.clean_bilateral_trade()` reads either schema.

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`, or from another registry.

Usage

```
whep_list_file_versions(file_alias, registry = NULL)
```

Arguments

`file_alias`	Internal name of the requested file. You can find the possible values in the `whep_inputs` dataset.
`registry`	The table that maps `file_alias` to a board and a frozen version. `NULL`, the default, is `whep_inputs`, this package's own inputs. Another package or project passes its own registry, a table with the columns `alias`, `board_url` and `version`, usually read with `whep_registry()`; it is validated the same way before it is used. See the section *Reading from another registry*.

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.

Other packages and projects can use the same reader for their own pinned inputs by passing a `registry`: a table with the same `alias`, `board_url` and `version` columns as `whep_inputs`, usually loaded and validated from a CSV with `whep_registry()`.

Usage

```
whep_read_file(
  file_alias,
  type = "parquet",
  version = NULL,
  years = NULL,
  year_col = "year",
  registry = 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.
- `nc` / `nc4`: Returns the path to the downloaded NetCDF instead of its contents, because these grids are read lazily by `ncdf4`/`terra` and are far too large to materialise as a tibble.
- `raw`: Returns the file path without any processing.

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 when the file has a registry version. Each release will have its own frozen versions. The version is the string that can be found in `whep_inputs` in the `version` column. A blank registry version requests the latest board version.
- `"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`.

`years` Optional integer vector of years to keep. For `parquet` the filter is pushed into the file, so only the row groups whose statistics overlap the requested range are read from disk and the exact set is applied afterwards. This is what makes a single-year read of a large monthly pin affordable: `lpjml-soc-hydrology` holds 193,317,960 rows over 1901-2022, and one year of it is 1.3 seconds and 34 MB instead of ~12 GB materialised. For `csv` the filter is applied after reading. The formats returned as a path (`nc`, `nc4`, `raw`, archives) cannot honour it and abort rather than ignore it. `NULL`, the default, reads the whole file.

`year_col` Name of the year column `years` filters on.

`registry` The table that maps `file_alias` to a board and a frozen version. `NULL`, the default, is `whep_inputs`, this package's own inputs. Another package or project passes its own registry, a table with the columns `alias`, `board_url` and `version`, usually read with `whep_registry()`; it is validated the same way before it is used. See the section *Reading from another registry*.

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.

Reading from another registry

A registry other than `whep_inputs` is authoritative for its aliases. The file is downloaded from that registry's board, cached, and read back from the cache when the board is unreachable, exactly as for `whep_inputs`. Two behaviours belong to `whep_inputs` alone, so with another registry the example board bundled with this package is not consulted, so an alias such as `read_example` in another registry resolves to that registry's board. The predecessor-pipeline warnings below are not raised, since they describe this package's pins.

```
reg <- whep::whep_registry("inst/extdata/input_registry.csv")
whep::whep_read_file("crop_yields", registry = reg)
whep::whep_list_file_versions("crop_yields", registry = reg)
```

Frozen predecessor-pipeline references

Four aliases in `whep_inputs` are not outputs of this package: `primary_prod`, `commodity_balance_sheet`, `processing_coefs` and `feed_intake`. They are the 2025-07-14 snapshot of the predecessor R-script pipeline this package replaced, kept frozen so `inst/scripts/compare_global_whep.R` can benchmark `whep` against it. No package function reads them, their schema is the old one (CamelCase, name-keyed, no polity columns) and they stop in 2021, so reading one warns.

Their numbers are not what current code produces, so they are references, not substitutes. `primary_prod` (2,443,516 rows, 1961–2021) against a fresh `build_primary_production()` over the same 2015–2021 window:

- The pin has no `slaughtered_heads` row in any of its years; the build emits 12,195 over 2015–2021. Three `item_prod` codes are pin-only (378, 773, 1163; 1,332 rows) and three appear in no pin year at all (1051, and the modelled grassland items 3001 and 3002).
- 317,587 rows against the build's 328,242 for the window, a 3.2% shortfall.
- Its 2020–2021 fodder harvested area is carried forward from 2019 — 85.93 Mha in each year, 468 country-item series over 96 areas, equal to 2019 to the last digit — where the build emits no fodder row at all after 2019, because `eu-agridb-fodder` stops in 2019, `faostat-production-old` in 2013, and `faostat-production` carries none of the 16 fodder items.
- Even in a year both cover they disagree: 2019 fodder harvested area is 85.93 Mha in the pin against 90.42 Mha from the build.

Build the current series with the matching `build_*()` or `get_*()` function instead.

The batch pin on the build path

Two further aliases were published in that same 2025-07-14 batch — `crop_residues` and `bilateral_trade`, alongside the four above, between 12:33:43Z and 12:33:50Z. Unlike the

four, these two are read on the default build path, which is why no warning is attached to them: every commodity balance build reads them, so a warning at the read is noise rather than information. `bilateral_trade` has since been regenerated by this package (#1122), so `crop_residues` is the only one still registered at its 2025-07-14 version.

They did **not** share a provenance despite sharing a timestamp. Each was established separately (#1054), because the timestamp alone establishes nothing.

`crop_residues` is predecessor output:

Read by `get_primary_residues()`, and from there by `build_commodity_balances()`. All 475,688 of its `Product` rows equal the `primary_prod` pin's `tonnes` values exactly — no key unmatched on either side, no value differing at a relative tolerance of 1e-6 — so it is a downstream artifact of the same predecessor run, carrying that run's production series into the commodity balance.

Its residue quantities are those production numbers times a residue-to-product ratio that varies by year, and times a **recovery rate**: the pin holds the residue the predecessor recovered from the field, not the residue produced (#1195). The predecessor's code (`Global/R/crop_npp.r`, then `afsetools::residue_use()`) writes each row as

```
product * kg_residue_kg_product_FM * HI_changes(region, year) * residue_dm_product_dm(region, category) / residue_dm_product_dm("West Europe", category) * recovery_rates(region, category),
```

with region and category keyed as in `residue_recovery.csv`. Rebuilt from the predecessor's own coefficient tables, that formula reproduces all 425,934 pin rows whose region it can key, at a relative tolerance of 1e-6. The year variation is `HI_changes` alone: a harvest-index change factor per region, given at eight anchor years from 1910 to 2000, linear in between and flat after 2000 (East Europe 1.10 in 1962 to 1.00 in 2000), which the predecessor attributes to Krausmann et al. (2013), *PNAS* 110:10324, Table M2 (assumed, unverified). It is a regional historical adjustment, not a yield response, and after 2000 the ratio of every region and category is constant. That is why `biomass_coefs$kg_residue_kg_product_FM` alone reproduces only 3,189 of the 472,790 comparable keys. Of the 33,411 keys that carry a residue of exactly 0, 30,620 are cells whose recovery rate is 0 (fodder crops in seven of the eight regions; roots and tubers, cassava, sugar beet and dry beans in West Europe and in North America and Oceania) and 1,919 are crops whose `biomass_coefs` ratio is 0.

WHEP ships the recovery rate and the regional ratio the predecessor used (both equal in all 160 cells), but not the `HI_changes` table. So `get_primary_residues()` undoes the recovery exactly, dividing by the same rate, but cannot recompute the predecessor's residue from production. It then re-keys the regional ratio on Wirsenius's own region membership, and estimates cereal residue (#1448), and that of the non-cereal crops with their own IPCC Table 11.2 line (#1518), from the pin's production and harvested area instead of reading it.

It is not a small input. The pin holds 7.63 Gt of recovered residue for 2010 (Straw 3.60 Gt, Other crop residues 2.49 Gt, Firewood 1.54 Gt) and 327.7 Gt over 1961–2021, which `get_primary_residues()` turns into 7.64 Gt (Straw 3.92 Gt, Other crop residues 2.82 Gt, Firewood 0.90 Gt) and 366 Gt of residue produced. 3,998 of its 249,095 output rows carry `NA` polity columns because the pin is name-keyed.

Driving the same residue model off a fresh `get_primary_production()` would move those numbers. Measured for 2010 at (`area_code`, `item_prod`): 1.958 Gt of current production sits on 6,425 keys the pin never sees (809 Mt of it primary crops, including the modelled

temporary-grassland item), 64.8 Mt on 296 keys is pin-only, and of the 8,010 shared keys 538 disagree, putting 1.102 Gt — 11.01% of the pin's shared-key mass — more than 1% apart. The largest class is rice, 673.9 Mt in the pin against 463.3 Mt fresh, a ratio of 0.6876: current code puts rice on a milled-equivalent basis while the pin's is paddy, which is a basis difference rather than an error, since straw scales with the field crop. The second is fodder, where the current build is close to twice the pin on every forage and silage item, the pin predating that work. Replacing the pin therefore means choosing a residue model, which is a science decision and not a refresh. For cereals that model is chosen by `get_primary_residues(cereal_residue = )`, and for the non-cereals with an IPCC line by `non_cereal_residue =`; both still run on the pin's production.

`bilateral_trade` is built by `build_detailed_trade()`:

Read by `get_bilateral_trade()` and by the live-animal branch of `build_commodity_balances()`. The registered version, `20261001T135651Z-ca7b3`, is `build_detailed_trade()`'s output verbatim, with its defaults and `extend_time = FALSE`, run on the `faostat-trade-bilateral` pin `20260407T095142Z-b3f81` (#1122): 10,038,220 rows over 1986–2021, polity-keyed, carrying 64,450,133,319 tonnes and 87,840,869,640 head.

The 2025-07-14 version it replaces, `20250714T123347Z-2c392`, held the same FAOSTAT Detailed Trade Matrix values in the predecessor pipeline's name-keyed schema — against the raw pin over 2010, all 296,642 shared `tonnes` keys and all 5,102 shared `Head` keys agree exactly — but without FAOSTAT's `1000 Head` rows: 76,141,882 thousand head of live broiler chicken, turkey, duck, rabbit and goose trade over 1986–2021, which PR #1113 recovered in the producer. The regenerated pin carries 7.50x its 11,707,083,640 head and 22.5% less than its 83,147,095,412 tonnes, all of that tonnage difference in CBS items 5001-5003, which no commodity balance carries: item 1293's unbacked tonnage, screened by `method_unbacked_quantity` (#1023), and three CBS names with no `item_cbs_code`. On the tonnes that do reach a trade matrix the two versions differ by 0.056%. `.clean_bilateral_trade()` reads either schema.

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"
)
```

`whep_registry` *Read a registry of pinned input files*

Description

Reads and validates a registry CSV, so another package or project can fetch its own pinned inputs with `whep_read_file()` and `whep_list_file_versions()` instead of the ones listed in `whep_inputs`.

A registry has one row per input and the same columns as `whep_inputs`:

- `alias`: the name the input is read by. It must be non-empty and unique.
- `board_url`: the public link to the `_pins.yaml` manifest of the `pins` board holding it. It must be an `https://saco.csic.es/public.php/dav/files/` public link ending in `/_pins.yaml`, as every board in `whep_inputs` is.
- `version`: the frozen pins version read by default, such as `"20250714T123343Z-114b5"`. A blank cell or `"latest"` reads the newest version on the board.

Further columns, such as a description, are kept. Every column is read as text, so a version string is never reinterpreted.

Usage

```
whep_registry(path)
```

Arguments

`path`	Path to the registry CSV.

Value

A tibble with one row per input, ready to pass as the `registry` argument of `whep_read_file()` and `whep_list_file_versions()`.

Examples

```
# This package's own registry is itself a valid registry.
system.file("extdata", "whep_inputs.csv", package = "whep") |>
  whep_registry()
```

`write_parquet_checked` *Write a Parquet file and verify it before returning.*

Description

Wrapper around `nanoparquet::write_parquet()` that reopens the file it just wrote and runs `assert_parquet_integrity()` on it. Use it for pipeline artifacts large enough that a silent write failure would go unnoticed; the verification is metadata-only, so it costs milliseconds regardless of file size.

Usage

```
write_parquet_checked(data, path, deep = FALSE, ...)
```

Arguments

`data`	Data frame to write.
`path`	Destination path.
`deep`	Whether to verify by decoding every row group as well as by checking the layout. Reads the whole file back.
`...`	Passed to `nanoparquet::write_parquet()`.

Value

Invisibly, `path`.

Examples

```
path <- tempfile(fileext = ".parquet")
write_parquet_checked(data.frame(x = 1:10), path)
unlink(path)
```

`write_table_checked` *Write a table to disk safely and verifiably.*

Description

One writer for WHEP's tabular artifacts, with the four properties an ad-hoc `nanoparquet::write_parquet()` or `readr::write_csv()` call does not have (whep#375):

- **The parent directory is created**, recursively. This is not a convenience: `nanoparquet::write_parquet()` given a path whose parent does not exist returns `NULL` and writes nothing at all, without warning, so a mistyped output directory looks like a successful build until something tries to read the file back.
- **The write is atomic.** The table goes to a temporary file beside the target and is renamed onto it only after it has been read back and verified. An interrupted, failed or corrupt write therefore leaves the previous artifact exactly as it was, instead of replacing hours of pipeline output with a truncated file.
- **The artifact is verified before it is published.** For Parquet that is `assert_parquet_integrity()` plus a row and column-name check; for CSV the file is re-read and its header and row count compared. `write_table_checked()` is the atomic, format-agnostic layer *on top of* `write_parquet_checked()`, which stays the in-place Parquet primitive it already was.
- **Overwriting is a decision.** `overwrite = FALSE` refuses an existing target instead of clobbering it.

Column order, column types and text encoding are preserved by round trip, including for zero-row tables: a Parquet round trip returns an identical tibble, and a CSV is always written as UTF-8. Prefer Parquet for anything large: CSV cannot carry types, its verification has to re-read the whole file, and a zero-column table is refused outright because its CSV is a zero-byte file, indistinguishable from a write that never happened.

Usage

```
write_table_checked(
  data,
  path,
  format = c("auto", "parquet", "csv"),
  overwrite = TRUE,
  sidecars = character(),
  ...
)
```

Arguments

`data`	Table to write. Must be a data frame; a tibble is returned unchanged by the round trip.
`path`	Destination path. Its parent is created if needed.
`format`	Output format: `"parquet"`, `"csv"`, or `"auto"` to take it from the file extension.
`overwrite`	Whether an existing target may be replaced.
`sidecars`	Which sidecars to write: any of `"schema"` and `"provenance"`. Defaults to none.
`...`	Passed to the underlying writer, `nanoparquet::write_parquet()` or `readr::write_csv()`.

Value

Invisibly, a one-row tibble describing what was written: `path`, `format`, `n_rows`, `n_cols`, `bytes`, `md5`, `schema_path` and `provenance_path` (the last two `NA` when the sidecar was not requested).

Sidecar contract

`sidecars` optionally writes YAML files beside the artifact, named after it: `<path>.schema.yaml` and `<path>.provenance.yaml`. They are written after the data file has landed, so the data file is never waiting on them.

The schema sidecar is `{format: "whep-table-schema/1", n_rows: <int>, columns: [{name, type}]}` with `columns` in the table's own column order. `type` is the column's first class, with `"numeric"` reported as `"double"` so that every atomic type name is one `vector()` accepts (`"integer"`, `"double"`, `"character"`, `"logical"`, `"list"`); a classed column reports its class (`"factor"`, `"Date"`, `"POSIXct"`). A consumer can therefore rebuild the prototype with no WHEP-specific lookup. This is deliberately the same shape a declarative schema validator (whep#373) and a typed empty-table constructor (whep#374) would consume; neither exists yet, and this function does not implement either.

The provenance sidecar is `{format: "whep-table-provenance/1"}` plus `path`, `table_format`, `n_rows`, `n_cols`, `bytes`, `md5`, `written_at` (UTC, ISO 8601), `whep_version`, `r_version` and `writer`. Every field except `written_at` is a function of the artifact, so two builds of the same table differ only in that field.

Examples

```
path <- tempfile(fileext = ".parquet")
table <- tibble::tibble(area_code = 724L, year = 2020L, value = 1.5)
manifest <- write_table_checked(table, path, sidecars = "schema")
manifest$n_rows
nanoparquet::read_parquet(path)
unlink(c(path, manifest$schema_path))
```

Index

∗ **datasets**
admin_items_eurostat, 22
admin_items_jrc, 23
admin_items_nass, 24
admin_items_sidra, 25
admin_species_eurostat, 32
admin_species_nass, 33
admin_species_sidra, 34
amg_h_by_input_type, 46
animals_codes, 47
biomass_coefs, 60
cb_processing, 348
cbs_trade_codes, 349
cft_mapping, 350
climate_mcf, 369
climate_mcf_ipcc, 370
coello_synthetic_n, 372
conv_bouwman, 390
conv_krausmann, 391
crops_eurostat, 404
crops_manure_n, 405
drainage_ranges, 419
europe_fao_spain_trade, 427
feed_characteristics, 431
feed_taxonomy, 432
fertiliser_n2o_modifiers, 433
gleam_animal_weights, 473
gleam_crop_residue_nitrogen, 474
gleam_crop_residue_params, 475
gleam_dressing_percentages, 476
gleam_energy_use_ef, 477
gleam_enteric_params, 478
gleam_feed_categories, 479
gleam_feed_composition, 479
gleam_feed_conversion_ratios, 480
gleam_feed_digestibility, 481
gleam_field_operation_ef, 482
gleam_fracremove, 482
gleam_geographic_hierarchy, 483
gleam_livestock_categories, 484
gleam_mechanization_levels, 485
gleam_milk_production, 486
gleam_mms_shares, 487
gleam_processing_transport_ef, 487
grazing_energy_coefs, 490
human_kgn_cap_reference, 495
human_kgn_cap_total_reference, 496
human_n_reference, 497
hwsd_texture_usda, 498
indirect_n2o_ef, 498
ipcc_2006_enteric_ef, 501
ipcc_2006_manure_ef, 502
ipcc_2006_mcf_temp, 503
ipcc_2019_bo, 504
ipcc_2019_cfi, 504
ipcc_2019_enteric_ef_cattle, 505
ipcc_2019_enteric_ef_other, 506
ipcc_2019_manure_ch4_ef_cattle, 507
ipcc_2019_manure_ch4_ef_other, 507
ipcc_2019_mcf_manure, 508
ipcc_2019_n2o_ef_direct, 510
ipcc_2019_n_excretion, 509
ipcc_2019_soil_n_ef, 511
ipcc_2019_ym, 512
ipcc_enteric_ef_tier1, 513
ipcc_manure_ef3, 514
ipcc_tier2_bo_values, 516
ipcc_tier2_energy_coefs, 516
ipcc_tier2_manure_ash, 517
ipcc_tier2_n_retention, 518
ipcc_tier2_ym_values, 518
items_cbs, 519
items_full, 519
items_prim, 521

items_prod, 522
items_prod_full, 523
lassaletta_grassland_share, 526
liv_lu_coefs, 528
livestock_constants, 528
livestock_production_defaults, 530
manner_default_technique_mix, 532
manner_incorporation_factor, 533
manner_params, 534
manner_rain_factor, 535
manner_rate_factor, 535
manure_inorganic_n, 536
max_intake_share, 537
meisinger_denitrification, 538
mueller_synthetic_n, 539
n2o_efs_disaggregated, 542
n2o_efs_ipcc2006, 543
n_attenuation_constants, 540
n_boundary_params, 541
nourish_levels, 545
nourishment_thresholds, 546
polities, 571
polities_cats, 573
polity_area_crosswalk, 578
polity_containment, 583
polity_label_aliases, 588
polity_label_item_corrections, 589
primary_double, 596
regime_yield_crop_mapping, 661
regional_mms_distribution, 662
regions_full, 664
residue_humification, 667
sjos_levels, 698
smil_2001_synthetic_n_global, 699
soc_soil_cover_curve, 706
soc_turnover_params, 707
soil_cn_ratios, 708
soil_hydraulic_by_texture, 710
som_ranges, 711
subsoil_no3_reduction, 718
temperature_adjustment, 720
uncertainty_ranges, 721
whep_inputs, 725

add_area_code, 11
add_area_code(), *182*
add_area_name, 12
add_area_name(), *42*
add_footprint_product_stage, 13
add_item_cbs_code, 14
add_item_cbs_name, 15
add_item_prod_code, 16
add_item_prod_name, 17
add_item_prod_name(), *253*
add_manure_bedding, 18
add_manure_bedding(), *49*, *200*, *224*, *267*
add_polity_code, 19
add_polity_code(), *68*, *69*, *79*, *85*, *92*, *93*, *113*, *114*, *116*, *119*, *121*, *127*, *128*, *130*, *137*, *140*, *143*, *152*, *157*, *158*, *177*, *185*, *187*, *199*, *203*, *208*, *218*, *226*, *231–233*, *239*, *241*, *245*, *257*, *259*, *261*, *266*, *277*, *281*, *285*, *286*, *324*, *445*, *454*, *456*, *471*, *575*, *577–579*, *584–586*, *588*, *591*, *592*, *613*, *641*, *649*, *674*, *677*
admin_coverage_prototype, 22
admin_coverage_prototype(), *689*
admin_items_eurostat, 22
admin_items_jrc, 23
admin_items_nass, 24
admin_items_sidra, 25
admin_shares_prototype, 25
admin_shares_prototype(), *26*, *421*, *600*
admin_shares_schema, 26
admin_shares_schema(), *25*, *55*, *56*, *421*, *599*, *606*, *668–670*, *672*
admin_source_vocabularies, *22–25*, 30, *32–34*
admin_species_eurostat, 32
admin_species_nass, 33
admin_species_sidra, 34
aggregate_grass_to_polity, 34
aggregate_unit_extent, 35
aggregate_unit_extent(), *55*, *353*, *692*
align_extension, 38
align_extension(), *144*
allocate_grazing_to_products, 39
allocate_grazing_to_products(), *158*, *159*
allocate_level_crops, 40
allocate_level_crops(), *162*, *190–192*, *654*, *655*, *657*, *688*
allocate_manure_to_land, 43
allocate_manure_to_land(), *182*, *199*,

200
allocate_manure_transport, 45
allocate_manure_transport(), *181*, *199*
amg_h_by_input_type, 46
animals_codes, 47
apply_management_losses, 48
apply_management_losses(), *18*, *44*, *173*, *195*, *199*, *296*, *303*, *318*, *499*, *514*, *515*
assert_footprint_invariants, 50
assert_parquet_integrity, 51
assert_parquet_integrity(), *734*, *735*
assert_table_schema, 52
assert_table_schema(), *419*, *421*, *681*, *682*
attach_provenance, 52
attach_provenance(), *467*, *639*, *658*
attach_scope, 53
attach_scope(), *442*, *468*
attribute_fallow_to_crops, 54
attribute_fallow_to_crops(), *122*, *131*

backcast_admin_shares, 55
backcast_admin_shares(), *41*, *192*, *353*, *692–694*, *697*
balance_io_flows, 58
balance_ras, 59
balance_ras(), *59*
biomass_coefs, 60, *141*, *399*, *459–461*, *463*, *520*
build_ag_land_support, 67
build_ag_land_support(), *205*, *220*, *223*, *224*, *234*, *237*, *272*
build_allocation_layer, 69
build_allocation_layer(), *41*, *43*, *625*, *686*, *687*, *722*
build_carbon_balance, 72, *706*
build_carbon_balance(), *80*, *148–150*, *219*, *223*, *620*, *627*, *633*
build_carbon_inputs, *74*, 80
build_carbon_inputs(), *74*, *75*, *235*
build_cbs_prices, 86
build_cbs_prices(), *188*
build_cell_climate_zone, 87
build_cell_climate_zone(), *170*, *175*, *176*
build_cell_moisture_regime, 88
build_cell_moisture_regime(), *226–228*
build_cell_polity, 93
build_cell_polity(), *68*, *155*, *163*, *169*, *182*, *215*, *246*, *247*, *280*, *392*, *470*, *688*, *712*
build_commodity_balances, 96
build_commodity_balances(), *86*, *124*, *126*, *188*, *255*, *451*, *452*, *464–466*, *472*, *612*, *724*, *727*, *728*, *732*, *733*
build_constant_territory_series, 107
build_constant_territory_series(), *531*
build_critical_n_binding, 110
build_critical_n_binding(), *211*, *271*, *608*
build_crop_land_extension, 112
build_crop_land_extension(), *452*, *453*, *714*
build_crop_soil_n2o_extension, 114
build_crop_soil_n2o_extension(), *236*, *321*, *712*, *713*
build_crop_water_consumption, 117
build_crop_water_use, 119
build_crop_water_use(), *117*, *118*
build_cropgrids_land_extension, 121
build_cropgrids_land_extension(), *131*, *132*, *177*, *443*
build_detailed_trade, 123
build_detailed_trade(), *450*, *452*, *465*, *728*, *733*
build_energy_co2_extension, 128
build_energy_co2_extension(), *483*
build_fao_arable_fallow_extension, 130
build_fao_arable_fallow_extension(), *189*, *351*, *352*, *354*
build_feed_demand, 136
build_feed_intake_local, 138
build_feed_intake_local(), *84*, *171*, *276*, *455*
build_food_supply, 140
build_food_supply(), *61*, *63*, *66*, *105*, *202*, *203*, *270*, *300*, *418*, *544*, *593*, *646*
build_footprint, 144
build_footprint(), *114*, *218*, *219*, *268*, *269*
build_grass_availability, 145
build_grass_availability(), *35*
build_grass_availability_lpjml, 146

build_grass_availability_lpjml(), *93*, *139*, *145*
build_grass_natural_carbon_inputs, 147
build_grass_natural_carbon_inputs(), *75*, *80–82*, *85*
build_grassland_intensity_classes, 153
build_grassland_intensity_classes(), *271*
build_grassland_land_extension, 156
build_grassland_land_extension(), *132*, *133*, *144*, *158*, *193*, *243*
build_grazing_feed_footprint, 158
build_gridded_landuse, 159
build_gridded_landuse(), *41*, *43*, *81*, *83*, *112*, *113*, *167*, *235*, *236*, *274*, *275*, *350*, *452*, *684*, *685*, *687*, *689*, *722*
build_gridded_livestock, 164
build_gridded_livestock(), *27*, *170*, *222*, *224*, *615*, *616*, *686*
build_gridded_livestock_emissions, 170
build_hayr_land_extension, 177
build_historical_land_areas, 179
build_historical_land_areas(), *253*, *576*
build_human_n, 180
build_human_n(), *93*, *219*, *223–225*, *237*, *279*, *495–497*
build_intake_dispersion, 185
build_io_model, 187
build_io_model(), *13*, *38*, *98*, *103*, *144*, *193*, *268*, *377*, *380–383*, *449*, *450*
build_land_balance_footprint, 189
build_land_balance_footprint(), *131*, *133*, *135*, *354*
build_level_crop_targets, 190
build_level_crop_targets(), *42*, *43*, *656*, *657*
build_livestock_ghg_extension, 193
build_livestock_ghg_extension(), *50*, *114*, *115*, *128*, *144*, *321*
build_livestock_nutrient_flows, *74*, 199
build_livestock_nutrient_flows(), *84*, *148*, *151*, *219*, *222*, *224*, *273*, *274*, *276*
build_loss_wedge, 201
build_loss_wedge(), *272*
build_n_boundary_country, 204
build_n_boundary_country(), *272*, *273*
build_n_boundary_exceedance, 208
build_n_boundary_exceedance(), *153*, *204*, *205*, *218*, *219*, *268*, *270–272*, *368*
build_n_boundary_percapita, 213
build_n_boundary_percapita(), *231*, *270*
build_n_deposition, 215
build_n_deposition(), *93*, *219*, *222*, *224*, *225*, *237*, *248*, *392*, *642*, *643*
build_n_exceedance_extension, 218
build_n_exceedance_extension(), *268*
build_n_inputs, 219
build_n_inputs(), *67*, *183*, *184*, *231–235*, *237*, *238*
build_n_loss_drivers, 226
build_n_loss_drivers(), *235*, *238*
build_n_pathway_exceedance, 229
build_n_pathway_exceedance(), *270*, *271*
build_n_percapita, 231
build_n_percapita(), *214*, *270*, *646*
build_nitrogen_balance, *74*, 233
build_nitrogen_balance(), *81*, *209*, *210*, *221*, *222*, *224*, *226*, *228*, *229*, *238*, *271*, *321*, *323*, *324*, *326*
build_nourishment_band, 239
build_nourishment_band(), *271–273*, *544*
build_polycell_land_uses, 242
build_polycell_support, 246
build_polycell_support(), *215*, *217*, *430*, *470*, *488*, *581*, *593*, *614*, *618*, *622*, *641*, *643*, *645*, *712*, *713*
build_primary_prices, 250
build_primary_production, *163*, 251, *574*, *666*
build_primary_production(), *97*, *107*, *156*, *165*, *251*, *287*, *351*, *458*, *577*, *661*, *724*, *726*, *731*
build_processing_coefs, 254
build_processing_coefs(), *724*
build_protein_quality, 255
build_protein_quality(), *240*, *241*, *260*, *272*

build_protein_requirement, 257
build_protein_requirement(), *240*, *653*
build_protein_score, 259
build_protein_score(), *255*
build_regime_yield_ratio, *74*, 262
build_regime_yield_ratio(), *81*, *82*, *235*, *236*, *637*, *652*, *715*, *716*
build_residue_bedding_supply, 267
build_residue_bedding_supply(), *18*, *200*
build_sjos_n_footprint, 268
build_sjos_n_footprint(), *218*, *270*
build_sjos_nitrogen, 270
build_soil_carbon_inputs, 273
build_soil_carbon_inputs(), *80–83*, *85*, *151*, *200*
build_supply_use, 277
build_supply_use(), *188*
build_total_population_grid, 279
build_total_population_grid(), *181*, *182*, *223*, *496*
build_trade_prices, 281
build_trade_prices(), *86*, *251*
build_urban_n *(build_human_n)*, 180
build_water_balance, 282
build_water_balance(), *117–119*

calculate_bnf, 286
calculate_bnf(), *219*, *221*, *222*, *719*
calculate_cohorts_systems, 287
calculate_cohorts_systems(), *595*
calculate_critical_n, 289
calculate_critical_n(), *608*, *609*
calculate_crop_bnf, 291
calculate_crop_bnf(), *286*, *348*
calculate_crop_npp, 291
calculate_crop_npp(), *273*, *293*
calculate_crop_npp_components, 292
calculate_crop_npp_components(), *273*, *286*
calculate_crop_residues, 293
calculate_crop_residues(), *81*, *106*, *236*, *291*, *292*, *459*, *462*
calculate_crop_roots, 294
calculate_crop_roots(), *66*, *81*, *236*, *291*, *292*
calculate_enteric_ch4, 295
calculate_food_gini, 300
calculate_indirect_n2o_nh3, 301
calculate_indirect_n2o_nh3(), *226*, *233*, *235*
calculate_livestock_emissions, 302
calculate_livestock_emissions(), *193*, *194*, *237*, *529*
calculate_lmdi, 306
calculate_lmdi(), *407*, *409*, *411*, *415*
calculate_manner_nh3, 313
calculate_manner_nh3(), *315*, *316*, *325*, *326*, *532*, *533*, *535*, *536*
calculate_manner_nh3_default, 315
calculate_manner_nh3_default(), *325*, *326*, *532*
calculate_manure_emissions, 316
calculate_manure_emissions(), *115*
calculate_n_leaching, 321
calculate_n_leaching(), *226*, *228*, *233–238*
calculate_n_surplus, 323
calculate_n_surplus(), *205*, *209*, *270*, *271*
calculate_nh3, 325
calculate_nh3(), *226*, *228*, *233*, *235*, *238*, *301*, *638*
calculate_nonsymbiotic_bnf, 326
calculate_nonsymbiotic_bnf(), *286*
calculate_npp_carbon_nitrogen, 327
calculate_npp_carbon_nitrogen(), *219*, *222*, *234*, *267*, *273*, *286*, *292*
calculate_nue_crops, 328
calculate_nue_livestock, 329
calculate_potential_npp, 330
calculate_potential_npp(), *293*
calculate_residue_destinies, 331
calculate_residue_destinies(), *200*, *222*, *234*, *267*, *327*, *461*
calculate_soc_amg, 336
calculate_soc_century, 337
calculate_soc_dynamics, 338
calculate_soc_dynamics(), *273*
calculate_soc_hsoc, 339, *339*
calculate_soc_icbm, 340
calculate_soc_lpjml, 341
calculate_soc_rothc, *342*, 343, *710*
calculate_soil_n2o, 344
calculate_soil_n2o(), *226*, *233*, *235*, *301*, *322*, *543*
calculate_system_nue, 346

calculate_uncertainty_bounds, 347
calculate_weed_bnf, 348
calculate_weed_bnf(), *286*
cb_processing, *103*, 348
cbs_trade_codes, 349
cft_mapping, 350, *531*, *636*, *637*, *661*, *662*, *685*
check_arable_composition, 351
check_arable_composition(), *133*, *134*, *189*
check_extent_jumps, 353
check_extent_jumps(), *56*
check_fodder_land_share, 354
check_footprint_conservation, 355
check_footprint_conservation(), *58*, *377*, *378*, *718*, *719*
check_inputs_supplied, 357
check_inputs_supplied(), *360*, *717*
check_keys_complete, 358
check_keys_complete(), *524*
check_labels_supplied, 360
check_parquet_integrity, 361
check_parquet_integrity(), *51*
check_series_jumps, 362
check_series_jumps(), *103*, *134*, *254*, *352*, *353*, *695*, *697*, *698*
check_supply_use_balance, 364
check_table_schema, 365
check_table_schema(), *25–27*, *29*, *52*, *419*, *421*, *682*
classify_sjos_n, 368
classify_sjos_n(), *207*, *269*, *270*
climate_mcf, *87*, *174*, *197*, *298*, *304*, *319*, 369, *370*, *371*, *503*, *664*
climate_mcf_ipcc, *174*, *197*, *298*, *304*, *305*, *319*, *369*, 370, *370*, *516*
coello_synthetic_n, *116*, *224*, 372
combine_cov, 373
combine_row_evidence, 374
combine_row_evidence(), *680*, *681*
compare_footprint_methods, 375
compare_footprint_methods(), *189*, *379*
compute_footprint, 376
compute_footprint(), *13*, *38*, *50*, *58*, *114*, *144*, *189*, *193*, *355*, *356*, *379*, *381*, *383*, *384*, *452*, *556*, *597*
compute_footprint_balance, 379
compute_footprint_balance(), *39*, *158*, *189*, *190*, *539*
compute_footprint_paths, 380
compute_footprint_paths(), *382*
compute_fp_product_paths, 382
compute_leontief_inverse, 384
compute_leontief_inverse(), *377*
compute_trade_flows_raw, 385
consolidate_sources, 386
conv_bouwman, 390
conv_krausmann, 391
correct_n_deposition, 392
correct_n_deposition(), *217*, *611*, *643*
create_alfredos_typologies, 394
create_finn_indicator, 395
create_finn_indicator(), *555*, *568*, *570*
create_grafs_plot_df, 396
create_n_nat_destiny, 397
create_n_nat_destiny(), *407*, *408*, *570*
create_n_production, 398
create_n_prov_destiny, 398
create_n_prov_destiny(), *64*, *396*, *406*, *407*, *409*, *410*, *412–415*, *568*, *570*
create_n_soil_inputs, 400
create_typologies_grafs_spain, 401
create_typologies_of_josette, 402
create_typologies_whep, 403
crops_eurostat, *23*, *32*, 404
crops_manure_n, 405, *587*

decompose_crop_livestock_conn, 405
decompose_crop_livestock_conn(), *407*, *412*, *553*
decompose_cropland_surplus, 406
decompose_cropland_surplus(), *408*, *410–412*, *547*, *548*, *550–553*
decompose_destiny_mix, 407
decompose_destiny_mix(), *553*, *554*
decompose_manure_losses, 408
decompose_manure_losses(), *411*, *412*, *547*, *549–552*
decompose_semi_natural_surplus, 410
decompose_semi_natural_surplus(), *411*, *412*, *547*, *549–552*
decompose_specialization_cov, 411
decompose_specialization_cov(), *406*, *408*, *412*, *567*
decompose_terr_losses, 412
decompose_terr_losses(), *411*, *413*, *560*, *563–565*

decompose_terr_losses_periods, 413
decompose_terr_losses_periods(), *548*, *561*, *562*
decompose_urban_losses, 414
decompose_urban_losses(), *412*, *547*, *549–552*
decompose_weighted_ratio, 415
disaggregate_ussr, 417
dplyr::coalesce(), *98*
dplyr::n_distinct(), *525*
drainage_ranges, 419

empty_table_from_schema, 419
empty_table_from_schema(), *25*, *26*
ensure_admin_shares, 420
ensure_admin_shares(), *28*
ensure_columns, 422
ensure_columns(), *365*, *419*, *421*
estimate_energy_demand, 423
estimate_energy_demand(), *530*
estimate_n_excretion, 424
estimate_n_excretion(), *44*, *45*, *199*, *200*, *274*, *456*, *714*, *715*
europe_fao_spain_trade, 427
evidence_conflicts, 428
evidence_conflicts(), *374*, *680*, *681*
evidence_for, 429
evidence_for(), *680*, *681*
expand_polycell_years, 430
expand_polycell_years(), *246*
expand_trade_sources, 431

feed_characteristics, 431
feed_taxonomy, 432
fertiliser_n2o_modifiers, *301*, *345*, 433
fill_linear, *373*, 434
fill_linear(), *125*, *437*, *500*, *501*
fill_proxy_growth, 435
fill_proxy_growth(), *21*, *55*, *56*, *58*, *575*
fill_sum, 437
fill_sum(), *437*
folded_reporting_areas, 439, *665*
folded_reporting_areas(), *94*, *163*, *169*, *582*, *684*
footprint_scope, 441
footprint_scope(), *53*
footprint_sensitivity, 442

get_arable_permanent_land, 443
get_arable_permanent_land(), *131*, *132*, *243*, *244*, *263–265*
get_bilateral_trade, 445
get_bilateral_trade(), *158*, *188*, *452*, *465*, *539*, *728*, *733*
get_crop_land_extension, 452
get_crop_land_extension(), *443*
get_feed_intake, 454
get_feed_intake(), *39*, *136*, *156–158*, *171*, *198*, *335*, *425*
get_livestock_cbs(), *97*, *448*, *449*
get_polity_geometries, 456
get_polity_geometries(), *108*, *246*, *593*, *676*
get_primary_production, *74*, 457
get_primary_production(), *81*, *82*, *115*, *122*, *129*, *131*, *154*, *157*, *158*, *175*, *178*, *193*, *194*, *198*, *235*, *251*, *253*, *451*, *465*, *715*, *716*, *727*, *732*
get_primary_residues, 459
get_primary_residues(), *114*, *116*, *334*, *335*, *451*, *464*, *727*, *732*
get_processing_coefs, 465
get_processing_coefs(), *98*
get_provenance, 467
get_provenance(), *52*, *640*, *653*
get_scope, 468
get_scope(), *442*
get_soc_climate_drivers, *74*, *76*, 468
get_soc_climate_drivers(), *93*, *710*
get_wide_cbs, 471
get_wide_cbs(), *86*, *98*, *106*, *124*, *188*, *255*, *269*, *365*, *446*, *466*
gleam_animal_weights, 473
gleam_crop_residue_nitrogen, 474
gleam_crop_residue_params, 475
gleam_dressing_percentages, *128*, 476
gleam_energy_use_ef, *128*, 477
gleam_enteric_params, 478
gleam_feed_categories, 479
gleam_feed_composition, 479
gleam_feed_conversion_ratios, *426*, 480
gleam_feed_digestibility, *426*, 481
gleam_field_operation_ef, 482
gleam_fracremove, 482
gleam_geographic_hierarchy, 483
gleam_livestock_categories, 484
gleam_mechanization_levels, 485

gleam_milk_production, 486
gleam_mms_shares, 487
gleam_processing_transport_ef, 487
glwd_water_fraction, 488
glwd_water_fraction(), *618*
grass_access_shares, 489
grass_access_shares(), *146, 147*
grazing_energy_coefs, 490
gridded_fallow_weights, 490
gridded_fallow_weights(), *54*, *132*, *354*

harmonize_interpolate, 491
harmonize_simple, 493
human_kgn_cap_reference, *181*, 495, *496, 497*
human_kgn_cap_total_reference, *181*, *279*, 496, *497*
human_n_reference, *181*, *495, 496*, 497
hwsd_texture_usda, 498, *650*

indirect_n2o_ef, *49*, *172*, *195*, *296*, *302*, *317*, 498
interp_vec, 500
ipcc_2006_enteric_ef, 501
ipcc_2006_manure_ef, 502
ipcc_2006_mcf_temp, 503
ipcc_2019_bo, 504
ipcc_2019_cfi, 504
ipcc_2019_enteric_ef_cattle, *173*, *196*, *297*, *303*, *318*, 505, *506*
ipcc_2019_enteric_ef_other, *173*, *196*, *297*, *303*, *318*, *505*, 506
ipcc_2019_manure_ch4_ef_cattle, 507
ipcc_2019_manure_ch4_ef_other, 507
ipcc_2019_mcf_manure, *369*, *503*, 508
ipcc_2019_n2o_ef_direct, 510, *515*, *663, 664*
ipcc_2019_n_excretion, 509
ipcc_2019_soil_n_ef, *301*, *321*, *344*, 511
ipcc_2019_ym, 512
ipcc_enteric_ef_tier1, *173*, *196*, *297*, *303*, *318*, *506*, 513
ipcc_manure_ef3, *49*, *173*, *196*, *297*, *303*, *318*, *510*, 514
ipcc_tier2_bo_values, *174*, *197*, *298*, *305*, *319*, *371*, 516
ipcc_tier2_energy_coefs, *175*, *198*, *299*, *302*, *305*, *320*, 516
ipcc_tier2_manure_ash, 517
ipcc_tier2_n_retention, 518
ipcc_tier2_ym_values, *478*, 518
items_cbs, 519
items_full, *63, 64*, *66*, *141*, *143*, 519
items_prim, 521
items_prod, 522
items_prod_full, *30*, *32*, *44*, *113*, *132*, *200*, *227*, *274*, *344*, *352*, *355*, *373*, 523, *661*, *700*, *714*

key_lattice_gaps, 524
key_lattice_gaps(), *359, 360*

lassaletta_grassland_share, 526, *674*
liv_lu_coefs, 528
livestock_constants, 528
livestock_emissions_to_kt, 529
livestock_production_defaults, 530

make_lpjml_covariate, 531
make_lpjml_covariate(), *712*
manner_default_technique_mix, *315, 316*, *326*, 532
manner_incorporation_factor, *316*, *532*, 533
manner_params, *314, 315*, *533*, 534, *535*
manner_rain_factor, 535
manner_rate_factor, 535
manure_engine_options, *370*, *372*, *506*, *513, 514*, *663*
manure_inorganic_n, *314*, 536
max_intake_share, 537
meisinger_denitrification, *321*, 538, *711*
melt_bilateral_trade, 539
mueller_synthetic_n, 539, *587*, *674, 675*

n2o_efs_disaggregated, *227*, *301*, *345*, 542
n2o_efs_disaggregated(), *543*
n2o_efs_ipcc2006, *345*, 543
n_attenuation_constants, 540
n_boundary_params, *214*, *232*, 541
normalize_nourishment, 544
normalize_nourishment(), *206*, *214*, *269, 270*, *368*, *546*
nourish_levels, 545
nourishment_thresholds, *544*, 546, *699*

plot_compart_factor, 547

plot_compart_factor(), *548*, *552*
plot_compart_factor_periods, 548
plot_compart_factor_periods(), *550*
plot_compart_factor_roll, 549
plot_compart_factor_roll(), *550*
plot_compart_factor_roll_panel, 550
plot_compart_factor_yearly, 552
plot_compart_factor_yearly(), *549*
plot_crop_livestock_conn, 553
plot_destiny_mix, 553
plot_finn_circularity, 554
plot_finn_circularity(), *570*
plot_footprint_sankey, 555
plot_input_output, 558
plot_input_output_livestock, 558
plot_input_output_system, 559
plot_loss_decomp, 560
plot_loss_decomp(), *547*, *565*
plot_loss_decomp_periods, 561
plot_loss_decomp_periods(), *548*, *562*
plot_loss_decomp_periods_panel, 562
plot_loss_decomp_periods_panel(), *564*
plot_loss_decomp_rolling, 563
plot_loss_decomp_rolling(), *564*
plot_loss_decomp_rolling_panel, 564
plot_loss_decomp_yearly, 565
plot_loss_decomp_yearly(), *552*, *563*
plot_national_trade_flows_raw, 566
plot_national_trade_validation, 566
plot_specialization_cov, 567
plot_typology_indicators_panel, 568
plot_typology_periods_panel, 570
polities, *108*, *160*, *166*, *253*, *265*, *484*, 571, *573*, *578*, *581*, *583*, *584*, *586–588*, *599*, *665*, *673–676*, *678*, *679*
polities_cats, 573, *588*, *667*
polity_anchor_drift, 575
polity_anchor_drift(), *21*, *586*
polity_area_crosswalk, *19*, *20*, *79*, *94*, *95*, *115*, *163*, *166*, *167*, *169*, *381*, *383*, *439*, *444*, *497*, *572*, *574*, 578, *583*, *584*, *587*, *588*, *590*, *594*, *624*, *665*, *666*
polity_bucket_coverage, 582
polity_bucket_coverage(), *577*, *581*, *586*, *592*
polity_containment, 583, *622*, *624*, *625*, *677*
polity_coverage_gaps, 584
polity_coverage_gaps(), *21*, *35*, *68*, *69*, *73*, *79*, *85*, *92*, *93*, *113*, *114*, *116*, *119*, *121*, *127*, *128*, *130*, *137*, *140*, *143*, *152*, *153*, *157*, *158*, *177*, *182*, *185*, *187*, *199*, *203*, *208*, *216*, *218*, *221*, *226*, *231–234*, *239*, *241*, *242*, *245*, *257*, *259*, *261*, *266*, *267*, *277*, *280*, *281*, *284–286*, *324*, *325*, *445*, *454*, *469*, *471*, *576*, *577*, *592*, *613*, *641*, *649*, *713*
polity_identity_conventions, 586
polity_identity_conventions(), *483*
polity_label_aliases, *581*, *583*, *584*, 588, *672*, *673*, *675*, *676*
polity_label_item_corrections, 589, *675*, *676*
polity_mapping_provenance, 590
polity_mapping_provenance(), *576*, *577*, *684*
polycell_example_geometries, 593
population_source_reach, 593
population_source_reach(), *612*, *647*
prepare_livestock_emissions, 595
prepare_livestock_emissions(), *176*, *193*
primary_double, *351*, 596
propagate_fp_uncertainty, 597
propagate_fp_uncertainty(), *373*

read_admin_family, 598
read_admin_family(), *600*, *672*
read_admin_shares, 599
read_admin_shares(), *27*, *688*
read_admin_stats_eurostat, 600
read_admin_stats_eurostat(), *32*, *672*
read_admin_stats_nass, 603
read_admin_stats_nass(), *31*, *32*, *672*
read_admin_stats_sidra, 606
read_admin_stats_sidra(), *32*, *672*
read_critical_n, 608
read_critical_n(), *110*, *111*, *155*, *209*, *210*, *230*, *271*, *289*, *290*
read_cru_climate, 610
read_cru_climate(), *468*
read_emep_deposition, 611
read_emep_deposition(), *392*, *643*
read_fbs_population, 612

read_fbs_population(), *141*, *646*, *648*
read_glaciated_areas, 614
read_glaciated_areas(), *247*, *249*
read_glw_density, 615
read_glw_density(), *167*, *168*, *686*, *687*
read_glwd_water, 618
read_glwd_water(), *247*, *488*, *489*
read_habitual_cv, 619
read_hwsd_topsoil_soc, 619
read_hyde_population, 621
read_level_country_grid, 622
read_level_country_grid(), *71*, *685–687*
read_lpjml_crop_cover, *73*, 626
read_lpjml_grass_productivity, 628
read_lpjml_grass_productivity(), *167*
read_lpjml_hydrology, 629
read_lpjml_hydrology(), *118*, *284*, *468*, *469*, *636*, *637*
read_lpjml_litterfall, 632
read_lpjml_natural_cover, *73*, 633
read_lpjml_natural_cover(), *633*
read_lpjml_npp, 634
read_lpjml_npp(), *147*, *151*, *636*
read_lpjml_regime_yield, 635
read_lpjml_regime_yield(), *263*, *266*, *267*, *661*
read_lpjml_wind, 638
read_lpjml_wind(), *236*, *326*
read_luh2_landuse, *74*, 639
read_luh2_landuse(), *36*, *57*, *67*, *68*, *148*, *243*
read_luh2_terrestrial, 641
read_luh2_terrestrial(), *247*
read_n_deposition, 642
read_n_deposition(), *216*, *225*, *392*
read_polycell_support, 644
read_polycell_support(), *78*, *94*, *95*, *161*, *167*, *179*, *180*, *243*, *249*, *250*, *470*, *622*, *625*
read_population, 645
read_population(), *232*, *271*, *593*, *612*
read_soil_hydraulic, 650
read_soil_hydraulic(), *468*, *469*, *498*, *710*
read_soil_ph, 651
read_soil_ph(), *620*, *650*
read_spam_yields, 652
read_spam_yields(), *82*, *262*, *263*, *661*
read_wpp_population, 653
read_wpp_population(), *279*, *280*, *496*, *612*, *646*, *648*
reconcile_admin_allocation, 654
reconcile_admin_allocation(), *688*, *722*
record_provenance, 658
record_provenance(), *52*, *53*, *680*, *681*
redistribute_feed, *74*, 659
redistribute_feed(), *136*, *137*, *147*, *151*, *199*, *200*, *222*, *424*, *425*, *456*
regime_yield_crop_mapping, *262*, *265*, *266*, 661, *716*
regional_mms_distribution, *173*, *196*, *297*, *304*, *318*, *320*, *370*, *487*, *515*, 662, *715*
regions_full, *182*, *439*, *483*, *574*, *575*, *578*, *588*, 664, *683*, *716*
residue_humification, *77*, *81*, *147*, *148*, *151*, *273*, *274*, 667
resolve_admin_shares, 668
resolve_admin_shares(), *22*, *27*, *28*, *41*, *686*, *688*, *692*
resolve_admin_units, 671
resolve_admin_units(), *192*, *599*, *688*
resolve_polity_label, 674
resolve_polity_label(), *581*, *587–590*, *671–673*
resolve_polity_lineage, 677
resolve_polity_lineage(), *153*, *265*, *648*
rlang::arg_match(), *167*, *219*, *269*, *615*
row_evidence, 680
row_evidence(), *28*, *374*
row_evidence_schema, 682
row_evidence_schema(), *374*, *428*, *429*, *681*
row_promotion_status, 683
row_promotion_status(), *591*
run_spatialize, 684
run_spatialize(), *22*, *167*, *350*
run_typology_sensitivity, 690

seam_gate, 691
seam_gate(), *688*, *696*
seam_gate_tolerances, 696
seam_gate_tolerances(), *692*
sjos_levels, *206*, *368*, 698
smil_2001_synthetic_n_global, *265*, 699

soc_crop_group, 700
soc_crop_group(), *81*, *120*
soc_rate_modifier_amg, 701
soc_rate_modifier_century, 702
soc_rate_modifier_icbm, 703
soc_rate_modifier_icbm(), *710*
soc_rate_modifier_lpjml, *342*, 704
soc_rate_modifier_rothc, *338*, *339*, 705, *706*
soc_soil_cover_curve, *73*, *627*, *633*, 706
soc_turnover_params, *337*, *342*, *343*, 707
soil_cn_ratios, 708
soil_hydraulic_by_texture, *498*, *650*, 710
som_ranges, 711
spatialize_country_n_to_crops, 712
spatialize_country_n_to_crops(), *83*, *219*, *224*, *237*, *275*
split_manure_management, 714
split_manure_management(), *18*, *48*, *49*, *199*
split_regime_yield, 715
split_regime_yield(), *81*, *235*, *236*, *262*, *264*
stamp_inputs_supplied, 717
stamp_inputs_supplied(), *357*, *358*
stats::approx(), *500*
subsoil_no3_reduction, *321*, 718
summarise_conservation, 718
summarise_conservation(), *51*
summarize_bnf, 719

temperature_adjustment, 720

uncertainty_ranges, 721
unit_cropland_extent, 721
unit_cropland_extent(), *655*, *657*
urban_kgn_cap_reference *(human_kgn_cap_reference)*, 495
urban_n_reference *(human_n_reference)*, 497

validate_national_trade, 722
validate_national_trade_raw, 723

whep_clear_cache, 724
whep_coef_table, 725
whep_inputs, *94*, *598*, *600*, *644*, *650*, *658*, 725, *726*, *728–731*, *733*, *734*
whep_list_file_versions, 728
whep_list_file_versions(), *733*, *734*
whep_polity_columns, *578*, *588*
whep_read_file, 729
whep_read_file(), *94*, *122*, *142*, *157*, *444*, *491*, *531*, *598*, *613*, *638*, *644*, *648*, *650*, *733*, *734*
whep_registry, 733
whep_registry(), *729*, *730*
write_parquet_checked, 734
write_parquet_checked(), *735*
write_table_checked, 735