Package: framrsquared.dev (via r-universe)

July 9, 2026

Type Package

Title [Development version of] FRAM Database Interface

Version 0.8.1.9001

Description [Development version of] a convenient tool for interfacing with FRAM access databases in R environments.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports cli, DBI, dplyr, framrosetta (>= 0.1.0), ggplot2, glue, here, janitor, lifecycle, lubridate, odbc, patchwork, purrr, readr, renv, rlang, RSQLite, scales, stats, stringr, tibble, tidyr, tools

RoxygenNote 7.3.2

Depends R (>= 4.1)

Suggests knitr, testthat (>= 3.0.0), rmarkdown, forcats, openxlsx2, withr

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Remotes FRAMverse/framrosetta

URL https://framverse.github.io/framrsquared/

BugReports https://github.com/FRAMverse/framrsquared

Config/pak/sysreqs make libicu-dev unixodbc-dev libx11-dev

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

Date/Publication 2026-04-27 20:53:47 UTC

RemoteUrl https://github.com/FRAMverse/framrsquared

RemoteRef dev

RemoteSha a5a9d3ad7b27f8f900a4ea2daa392e111a06fd8a

Contents

add_total_mortality . . . 4
addstock_check . . . 4
aeq_mortality . . . 5
bkfram_checks_coho . . . 6
calc_fram_scaling . . . 7
calculate_stock_comp . . . 8
change_run_id . . . 9
check_bp_coverage . . . 10
coho_mark_rates . . . 10
cohort_abundance . . . 11
compare_databases . . . 12
compare_fishery_input_flags . . . 15
compare_fishery_inputs . . . 15
compare_inputs . . . 16
compare_inputs_chart . . . 17
compare_non_retention_input_flags . . . 18
compare_non_retention_inputs . . . 18
compare_recruits . . . 19
compare_runs . . . 20
compare_sl_ratio . . . 21
compare_stock_fishery_rate_scalers . . . 21
connect_fram_db . . . 22
copy_fishery_scalers . . . 23
copy_run . . . 24
copy_tamm . . . 25
create_soncc_pasteable . . . 26
disconnect_all_fram_connections . . . 27
disconnect_fram_db . . . 28
fetch_quarto_templates . . . 28
fetch_table . . . 29
fetch_table_bkchin . . . 30
filter_ak . . . 31
filter_bc . . . 32
filter_ca . . . 32
filter_coast . . . 33
filter_commercial_wa_nt . . . 34
filter_hatchery . . . 35
filter_marine . . . 36
filter_mixed . . . 36
filter_net . . . 37
filter_or . . . 38
filter_puget_sound . . . 39
filter_sport . . . 39
filter_stt . . . 40
filter_stt_nt . . . 41
filter_wa . . . 42

filter_wild . . . 42
fishery_mortality . . . 43
initialize_project . . . 44
label_fisheries_db . . . 45
label_flags . . . 46
label_stocks_db . . . 46
label_timesteps_db . . . 47
list_extant_fram_connections . . . 48
make_batch_runs . . . 48
management_week . . . 49
modify_table . . . 50
mortality_scalers . . . 51
msf_encounters . . . 52
msf_landed_catch . . . 53
msf_mortalities . . . 53
msp_mortality . . . 54
na_non_retention_from_flag . . . 55
na_scalers_from_flag . . . 55
plot_impacts_per_catch_heatmap . . . 56
plot_stock_comp . . . 58
plot_stock_mortality . . . 59
plot_stock_mortality_time_step . . . 60
population_statistics . . . 61
post_season_abundance . . . 62
remove_run . . . 63
run_info . . . 63
sensitivity_custom . . . 64
sensitivity_exact . . . 65
sensitivity_scaled . . . 67
statistical_week . . . 69
stock_fate . . . 70
stock_mortality . . . 71
terminal_fisheries . . . 71
terminal_info . . . 72
terminal_stocks . . . 73
translate_nr_flag . . . 73
translate_scalers_flag . . . 74
truns_fisheries . . . 75
truns_stocks . . . 75

Index **77**

add_total_mortality *Sum separate mortality columns into new "total_mortality" column*

Description

Convenience function for combining the separate mortality columns of the Mortality table. Note: this does *not* account for AEQ for Chinook on its own, but can be used on the output of `aeq_mortality()` to give total mortalities in AEQs.

Usage

```
add_total_mortality(.data)
```

Arguments

`.data`	Dataframe with separate mortality columns `landed_catch`, `non_retention`, `shaker`, `drop_off`, and `msf_` versions of each. Typically comes from `fetch_table("Mortality")` or `aeq_mortality()`.

Value

`.data` with additional `$total_mortality` column (numeric vector) just before `$landed_catch`.

addstock_check *Check FRAM database after adding new stock*

Description

Either provides the step by step process of adding new stock to a FRAM database, or walks through fram database run and checks the tables for potential errors associated with adding new stock.

Usage

```
addstock_check(
  file_name = NULL,
  run_id = NULL,
  old_stockcount = 78,
  override_db_checks = FALSE
)
```

Arguments

file_name	filepath to database. If NULL, provide summary of process instead. Default = NULL.
run_id	RunID associated with the new stock in the FRAM database. If left as NULL, provide summary of process instead. Default = NULL.
old_stockcount	The number of stocks previously present to treat as the "baseline" – several checking steps will focus solely on newly added stocks. Defaults to 78.
override_db_checks	Ignore species, database type. When FALSE, function will stop if the database is not Chinook or if it's a transfer file. Defaults to FALSE.

Value

Invisibly returns either 0 (if giving the step-by-step process) or the number of errors detected (when actually running the check).

Examples

```
## Not run:
## review process
addstock_check()
## check database for additional stock
addstock_check("2024 Pre-Season Chinook DB - first test.mdb",
run_id = 138)

## End(Not run)
```

---

aeq_mortality *Extract AEQ mortality from Chinook FRAM database.*

Description

Calculates AEQ mortality for Chinook, translating mortalities from dead fish to Adult EQuivalents (which are the units used to calculate ERs and management objectives). This metric accounts for the probability of a fish dying of natural causes before reaching escapement (e.g, the mortality of an age 2 fish in timestep 1 has a smaller AEQ than the mortality of an age 5 fish in timestep 5). By default, aeq_mortality() also expands for Model Stock Proportion (MSP), but this can be turned off by setting optional argument msp = FALSE.

Usage

```
aeq_mortality(fram_db, run_id = NULL, msp = TRUE, label = TRUE)
```

Arguments

fram_db	Fram database object
run_id	numeric, RunID(s) as ID or ID:ID
msp	Do we use MSP expansion? Logical, default true.
label	Add human-readable columns for flags, fisheries, stocks? Based on the Stock and Fishery tables of the current database. Logical, defaults to TRUE.

Details

aeq_mortality() returns a dataframe that superficially resembles fetch_table("Mortality"). However, the mortality values in landed_catch through msf_drop_off are presented in units of AEQ. Additionally, base_period_id, aeq_constant, and terminal_flag, which were used to calculate the AEQ values, are included is columns of this dataframe. These are left for understanding/diagnostics/debugging purposes, and can be ignored when working with the AEQ mortalities.

Value

Tibble resembling output of fetch_table("Mortality"), but with a few modifications. See Details.

Examples

```
## Not run:
fram_db |> aeq_mortality(run_id = 132)

## End(Not run)
```

---

bkfram_checks_coho *Performs error checks of a backwards FRAM run Returns nested tibble with diagnostics*

Description

Performs error checks of a backwards FRAM run Returns nested tibble with diagnostics

Usage

```
bkfram_checks_coho(fram_db, backward_run_id = NULL, forward_run_id = NULL)
```

Arguments

fram_db	fram database object, supplied through connect_fram_db
backward_run_id	numeric, RunID
forward_run_id	numeric, RunID

Examples

```
## Not run:
fram_db |> bkfram_checks_coho(backward_run_id = 132, forward_run_id = 133)

## End(Not run)
```

`calc_fram_scaling` **[Experimental]** *Calculate match/replace df based on scaling*

Description

Uses a match/replace-style table like in `modify_table()`, but allows user to specify scaling factors for individual columns rather than absolute values, and returns the corresponding match/replace df to be used in `modify_table()`. This is intended to support sensitivity analyses structured as "carry out 100 runs, with stock recruit scalers for stock X running from 5% to 500% of the current value" (`calc_fram_scaling()` is only one part of the pipeline for this). See `modify_table()` for details of setting up a match/replace dataframe; the only difference here is that the columns to be scaled should start with "scale_" instead of "replace_", and should contain the scalers.

Usage

```
calc_fram_scaling(fram_db, table_name, df)
```

Arguments

`fram_db`	FRAM database
`table_name`	name of FRAM table
`df`	As the match/replace dataframe of `modify_table`, but with "scale_" columns instead of "replace_" columns. Columns must start with either "match_" or "scale_", and should otherwise match the names of columns in the corresponding FRAM table. Columns starting with "scale_" define the scaling factor to be applied to values in that column (for rows matched with the "match_" columns). For example, scaling the StartCohort values to 50% in the Cohort table might be achieved with columns "match_RunID", "match_StockID", "match_age", "match_TimeStep", "scale_StartCohort", with values of 0.5 in scale_Startcohort.

Details

Note: In the StockRecruit table, RecruitScaleFactor and RecruitCohortSize should have a fixed relationship (scale factor = cohort size / base period size). For this reason, if applying scaling to only one of those columns, `calc_fram_scaling` will automatically apply the same scaling to the other. If the scaling for both is provided and they do not match, `calc_fram_scaling` will error out.

Value

A match/replace df for use in `modify_table()`, with "replace_" values generated by scaling the corresponding values in the FRAM database. Includes additional "match_" columns for all columns except "PrimaryKey"

Examples

```
## in run 31, decrease stock 1's recruit numbers by 50% and double 2's recruit numbers

df <- data.frame(match_RunID = c(31, 31), match_StockID = 1:2, scale_RecruitScaleFactor = c(.5, 2))
## check match/scale dataframe
df

## Not run:
library(here)
fram_db <- connect_fram_db(here("example_fram_db.mdb"))

df_scaled <- calc_fram_scaling(fram_db, "StockRecruit", df)
## here's what the values become:
df_scaled

## we can then modify the database
modified <- modify_table(fram_db, "StockRecruit", df_scaled)

disconnect_fram_db(fram_db)

## End(Not run)
```

---

calculate_stock_comp *Plot stock composition*

Description

Produces a dataframe of stock composition for a given timestep and fishery. Low frequency stocks are grouped into geographic area.

Usage

```
calculate_stock_comp(
  fram_db,
  run_id,
  fishery_id,
  time_step,
  group_threshold = 0.01
)
```

Arguments

fram_db	Fram database object
run_id	numeric, RunID
fishery_id	numeric, Fishery ID
time_step	numeric, Time Step
group_threshold	numeric, Stock percentages below this number will be grouped. Default is 1%, setting to zero will turn grouping off

Value

Tibble identify run, age, fishery, timestep, stock, and marks tatus. Provides calculated total mortality `$total_mort`, the proportion of all mortality in this fishery associated with that row (`ts`), and the sum `ts` for marked and unmarked fish of a given stock (`$total`) which can be used for sorting purposes.

See Also

`plot_stock_comp()`

Examples

```
## Not run:
fram_db |> stock_comp(run_id = 132)

## End(Not run)
```

---

change_run_id *Changes a run's ID number in a FRAM database*

Description

Changes a run's ID number in a FRAM database

Usage

```
change_run_id(fram_db, old_run_id, new_run_id)
```

Arguments

fram_db	FRAM database object
old_run_id	FRAM run ID to be changed
new_run_id	New FRAM run ID

Value

nothing

Examples

```
## Not run: fram_db |> change_run_id(old_run_id = 132, new_run_id = 300)
```

`check_bp_coverage` *Check the FisheryScalers and NonRetention have valid values*

Description

Compares FisheryScalers and NonRetention entries against base period to ensure there are no inputs that can't be represented in FRAM.

Usage

```
check_bp_coverage(fram_db, run_id)
```

Arguments

`fram_db`	FRAM database connection
`run_id`	FRAM run id

Value

Invisibly returns a list of tables identifying the fishery x timesteps that are not in base period

Examples

```
## Not run:
fram_db |> check_fishery_coverage(run_id = 156)

## End(Not run)
```

`coho_mark_rates` **[Experimental]** *Predict Fram encounter markrates by fisheries*

Description

Returns a tibble displaying predicted FRAMencounter mark rates by fishery, fishery type, and time-step. Only works for Coho database.

Usage

```
coho_mark_rates(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)

Value

Dataframe identifying the run, fishery, timestep, year, and base period. Provides total marked ($AD) and unmarked ($UM) mortalities, and the markrate ($mark_rate). Separate rows for NS and MSF fisheries, distinguished by $fishery_type.

Examples

```
## Not run:
fram_db |> coho_mark_rates(run_id)

## End(Not run)
```

---

cohort_abundance **[Experimental]** *Calculate starting cohort abundance*

Description

The starting cohort abundance listed in the database can be wrong. This function calculates the value by multipying the Stock Recruit Scalar by the base period abundance.

Usage

```
cohort_abundance(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)

Value

dataframe identify run_id, stock, age, and then providing the starting cohort abundance ($recruit_cohorts).

Examples

```
## Not run:
fram_db |>  cohort_abundance(run_id = 145)

## End(Not run)
```

compare_databases **[Experimental]** *Compare tables in two equivalent FRAM databases*

Description

Function supports QAQC practices by comparing the tables of two FRAM databases and identifying (and quantifying) differences.

Usage

```
compare_databases(
  fram_db_1,
  fram_db_2,
  runid_use = NULL,
  tables_use = NULL,
  slim = TRUE,
  quiet = TRUE
)
```

Arguments

fram_db_1	connection to FRAM database that contains the results from running baseline FRAM runs (e.g., our "original" version).
fram_db_2	connection to FRAM database that contains the results from running modified FRAM runs (e.g., running a new version of FRAM or using modified input values)
runid_use	Numeric vector of the run_ids to compare. Optional. If not provided, compare all run ids in the databases.
tables_use	Vector of character strings. Optional. If provided, only compare the listed tables.
slim	Logical. Optional, defaults to TRUE. If TRUE, do not include $tabs_file1 and $tabs_file2 in output list.
quiet	Logical, defaults to TRUE. When TRUE, suppress messages showing individual steps.

Details

The key output is the $ratios tibble, which contains every comparison of relevant table entries in long-form. These comparisons are implemented by first aligning corresponding table rows using appropriate key columns (e.g. run_id, fishery_id, stock_id, age, time_step, etc).

In $ratios, the table and variable columns specify the table column being compared, respectively. prop_err, abs_err, and scale_err provide measures of the changes between the "original" value (from fram_db_1) and the "comparison" value (from fram_db_2). More on those below. The original and new columns give the actual values being compared. run_id through time_step specify the rows being compared. bkfram_off_by_fish and bkfram_off_by_prop provide the context for the comparison (more on that below).

Quantifying error

Because FRAM involves numerical solvers, we expect some small differences in table entries even when comparing two effectively equivalent databases. `compare_databases()` provides three metrics for these changes. In each case, it is assumed that `fram_db_1` is the reference file; the "error" measures all show how much the value in `fram_db_2` changed relative to the corresponding value in `fram_db_1`. The simplest measure of error is the `abs_err`. This is the absolute value of the difference between the two values. If we're looking at an entry with table = "Mortality" and variable = "landed_catch", then an abs.err of 5 means that the `fram_db_2` entry was five fish more or less than the `fram_db_1` entry. You can confirm this by looking at the `original` and `new` columns. While `abs_err` is the most easily interpreted, it is often not very meaningful when looking across tables and variables. After all, an `abs_err` value of 5 could mean a a relatively meaningless change of five fish for a landed catch entry that was originally thousands of fish, but the same value of 5 would be a huge change in fishing effort if it were for a fishery scaler entry.

One way to make error comparable across tables and variables is to calculate the proportional error. If an entry changed by 0.01%, that's not meaningful, while if it changed by 10%, that is. `$prop_err` provides this proportional error, where -0.5 means the entry in fram_db_2 was 50% less than the corresponding value in fram_db_1, and a value of 2 means the entry in fram_db_2 was 200% more than the corresponding value in fram_db_1. ' This gives error in context of the original value, and is often a good a way to look for problems. However, we sometimes find very large `$prop_err` values for changes that aren't concerning. For example, we may have an entry for landed catch in the mortality table that was 0.00001 fish in fram_db_1, and 0 fish in fram_db_2. In all practicality these two values are identical, and the 0.00001 fish difference is likely one of random jitter in the numerical solver or rounding differences. However, our `$prop_err` value for this cell is `-1`, the most extreme negative change we can get. We can jointly look at `$abs_err` and `$prop_err` to address the potential for misleadingly large errors `$prop_err`, but it would be nice to have a single error metric that provides error in context without being sensitive to very small entries in fram_db_1.

`scale_err` is an elaboration on `$prop_err` that provides broader context. `$prop_err` takes the absolute error and scales by the original value in fram_db_1. `$scale_err` generalizes this idea, first calculating the average error for each table-variable combination, and then scaling the absolute error by the corresponding table-variable average. That is, if an entry for landed_catch in the Mortality table was 0.001 in fram_db_1, and was then 0.002 in fram_db_2, and the average of all landed_catch entries in fram_db_1 was 1000, then the `prop_err` would be 1 (since fram_db_2 had double the value of fram_db_1, or `(0.002-0.001)/0.001`), and the `scale.err` would be `0.000001` (`(0.002-0.001)/1000`). This better captures our intuition that a difference of 0.001 fish in the landed catch isn't a big deal, since those values are typically huge. `scale_err` is thus a measure of error that is comparable across variables and tables, essentially answer the question "Has this entry changed a lot for this kind of variable and table?".

While `scale_err` is frequently the most useful error metric, `compare_databases()` provides all three. There may be contexts in which it's important to focus on the proportional error. For example, large proportional errors landed catch for the catch rare stocks can be important, but the much larger catch from other stock could water down the `scale_err` metric.

Addressing the backwardsFRAM wiggle

For post-season runs, the backwards FRAM algorithm is employed; its solver stops when the estimated fish numbers are within 1 fish of the target size. This means that there is the potential for substantial "wiggle" in bkFRAm values when comparing two databases. This wiggle can propagate to other tables, especially for stock-age-timesteps in which the target values were quite small (so a wiggle of +/- 1 fish would be a proportionally large amount). For this reason, it can be useful to see how

our measures of error correspond to the errors in the corresponding bk fram table. For every table entry for which this makes sense (e.g., has a stock id, age, and timestep), `$bkfram_off_by_fish` gives the absolute error in the corresponding row of the BackwardsFram table, and `bkfram_off_by_prop` give the relative error (as a proportion) in the corresponding row of the BackwardsFram table. If this bkfram wiggle were the cause of observed errors, we would expect the largest errors to correspond to the largest `$bkfram_off_by_fish` or `$bkfram_off_by_prop` values.

Suggestions

For simple plotting to see if the original and new values fall on the 1:1 line, `ggplot2::geom_point()` can be used, with `$ratios$original` and `$ratios$new` for x and y, and a facet_wrap by `table` (and perhaps `variable`) to make plots readable. For identifying meaningful change, scale.err is likely the best measure of error. It can be helpful to plot scale.err against bkfram.off.by.fish or bkfram.off.by.prop to see if the table entries with the largest error correspond to the stock-fishery-age-timestep in which there's the largest wiggle in the backwards fram solutions.

When digging into individual tables, it can sometimes be helpful to look at the comparisons in `$ratios_detailed`, which contains additional columns which did not fit into the standardized formatting of `$ratios`.

Value

List of lists and tibbles containing comparison information:

- `$ratios` tibble comparing every entry of every relevant column of every table. See "Details" for column descriptions.
- `$ratios_detailed` list of tibbles showing the contents of `$ratios` broken into tables, with additional non-compared columns present (e.g., `stock_name` in `$ratios_detailed$Stock`). Not shown if `slim` is TRUE.
- `$nrow_tracker` dataframe providing the number of rows in each table of file1 (`$nrow_original`), file2 (`$nrow_new`), and the joined comparison (`$nrow_comparison`). Useful to track down cause of many-to-many join warnings that can result from duplicated table entries; unless there are duplicate entries, `$nrow_comparison` should be less than or equal to the minimum of `$nrow_original` and `$nrow_new`.
- `$tabs_file1` List containing the original fetched tables from `file1`. Not returned if argument `slim` is TRUE
- `$tabs_file2` List containing the original fetched tables from `file2`. Not returned if argument `slim` is TRUE.

Examples

```
## Not run:
fram_db_1 = connect_fram_db("Valid2022_Round_7_1_1_11142023_REFERENCE_fixed - fork rebuild.mdb")
fram_db_2 = connect_fram_db("Valid2022_Round_7.1.1_11142023 - green river split.mdb")
out = tables_compare(fram_db_1, fram_db_2)

## End(Not run)
```

compare_fishery_input_flags

Compares the fishery flags of two runs

Description

Compares the fishery flags of two runs

Usage

```
compare_fishery_input_flags(fram_db, run_ids, verbose = TRUE)
```

Arguments

fram_db	FRAM database object
run_ids	Vector of two run_ids
verbose	If TRUE, print an update to screen when there are no differences in recruits.

Value

Tibble with all fishery x timesteps that changed fishery input flags. $fishery_id, $fishery_label, and $time_step identify the fishery x timestep combination, and $flag_original and $flag_comparison show the flags in the first and second runs, respectively.

See Also

Other comparisons: compare_fishery_inputs(), compare_inputs(), compare_inputs_chart(), compare_non_retention_inputs(), compare_recruits(), compare_runs(), compare_sl_ratio(), compare_stock_fishery_rate_scalers()

Examples

```
## Not run: fram_db |> compare_fishery_input_flags(c(55, 56))
```

compare_fishery_inputs

Compares the fishery inputs of two runs

Description

Compares the fishery inputs of two runs

Usage

```
compare_fishery_inputs(fram_db, run_ids, tolerance = 0.01, verbose = TRUE)
```

Arguments

fram_db	FRAM database object
run_ids	Vector of two run_ids
tolerance	Minimum % change needed to flag a difference. Set to 0 to flag any changes at all. Defaults to 0.01.
verbose	If TRUE, print an update to screen when there are no differences in recruits.

Value

All fishery x timesteps in which the fishery inputs changed by at least (tolerance x 100) % between the specified runs. $fishery_id, $fishery_label, and $timestep identify the fishery x timestep, $parameter identifies which parameter changed (.e.g, quota, msf_quota, etc.). $original and $comparison show the values from the first and second runs, respectively. $prop_diff shows the proportional change from the first to second run (e.g., 0.16 = 16% increase).

See Also

Other comparisons: compare_fishery_input_flags(), compare_inputs(), compare_inputs_chart(), compare_non_retention_inputs(), compare_recruits(), compare_runs(), compare_sl_ratio(), compare_stock_fishery_rate_scalers()

Examples

```
## Not run: fram_db |> compare_fishery_inputs(c(55, 56))
```

---

compare_inputs *Compare FisheryScalers tables between two runs.*

Description

Generates a dataframe that compares fishery scalers table for two runs identified by run_id's.

Usage

```
compare_inputs(fram_db, run_ids)
```

Arguments

fram_db	FRAM database object
run_ids	Vector of two run_ids

Value

Data frame of differences. *_original columns show the values in the first run of run_ids, while *_comparison show the values of the second run of run_ids. Quota and MSF quota have been combined into $total_quota_*. $prop_diff = proportional change in quota (comparing the appropriate quotas based on fishery flags). $reg_change = change in regulations.

See Also

Other comparisons: compare_fishery_input_flags(), compare_fishery_inputs(), compare_inputs_chart(), compare_non_retention_inputs(), compare_recruits(), compare_runs(), compare_sl_ratio(), compare_stock_fishery_rate_scalers()

Examples

```
## Not run: fram_db |> compare_inputs(c(100,101))
```

---

compare_inputs_chart *Generate heat map of changed values between two run inputs.*

Description

Can be a very busy chart if not filtered down. Consider using a `filter_*()` function on the dataframe before piping into `compare_input_chart`.

Usage

```
compare_inputs_chart(.data)
```

Arguments

`.data`	Dataframe origination from the compare_inputs() function

Value

ggplot object with heatmap of changes in inputs.

See Also

Other comparisons: compare_fishery_input_flags(), compare_fishery_inputs(), compare_inputs(), compare_non_retention_inputs(), compare_recruits(), compare_runs(), compare_sl_ratio(), compare_stock_fishery_rate_scalers()

Examples

```
## Not run: fram_db |> compare_inputs(c(100, 101)) |> compare_inputs_chart()
```

compare_non_retention_input_flags

Compares the non retention flags of two runs

Description

Compares the non retention flags of two runs

Usage

```
compare_non_retention_input_flags(fram_db, run_ids, verbose = TRUE)
```

Arguments

fram_db	FRAM database object
run_ids	Two run ids
verbose	If TRUE, print an update to screen when there are no differences in recruits.

See Also

compare_runs()

Examples

```
## Not run: fram_db |> compare_non_retention_input_flags(c(55, 56))
```

compare_non_retention_inputs

Compares the non retention inputs of two runs

Description

Compares the non retention inputs of two runs

Usage

```
compare_non_retention_inputs(fram_db, run_ids, verbose = TRUE)
```

Arguments

fram_db	FRAM database object
run_ids	Vector of two run_ids
verbose	If TRUE, print an update to screen when there are no differences in recruits.

Value

Tibble with all non-retention parameters that changed between the first and second runs. `$fishery_id`, `$fishery_label`, and `$time_step` identify the fishery x timestep that changed, `$parameter` identifies the parameter, and `$original` and `$comparison` present the values in the first and second runs, respectively.

See Also

Other comparisons: `compare_fishery_input_flags()`, `compare_fishery_inputs()`, `compare_inputs()`, `compare_inputs_chart()`, `compare_recruits()`, `compare_runs()`, `compare_sl_ratio()`, `compare_stock_fishery_ra`

Examples

```
## Not run: fram_db |> compare_non_retention_inputs(c(55, 56))
```

---

compare_recruits *Compares the recruit scalers of two runs*

Description

Compares the recruit scalers of two runs

Usage

```
compare_recruits(fram_db, run_ids, tolerance = 0.01, verbose = TRUE)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Vector of two run_ids
`tolerance`	Minimum % change needed to flag a difference. Set to 0 to flag any changes at all. Defaults to 0.01.
`verbose`	If `TRUE`, print an update to screen when there are no differences in recruits.

Value

tibble with `$stock_id`, `$age`, and `$stock_name` identifying stock x age combinations in which the recruit cohort sizes changed by at least `tolerance x 100%`. `recruit_cohort_original` and `$..._comparison` give the recruit cohort for the first and second run_ids provided. These are calculated directly from "StockRecruit" column RecruitScaleFactor and the "BaseCohort" table, as the RecruitCohort column of the "StockRecruit" table can be misleading. `$prop_diff` gives the proportional change from the original to comparison runs (ie 0.16 = 16% increase).

See Also

Other comparisons: `compare_fishery_input_flags()`, `compare_fishery_inputs()`, `compare_inputs()`, `compare_inputs_chart()`, `compare_non_retention_inputs()`, `compare_runs()`, `compare_sl_ratio()`, `compare_stock_fishery_rate_scalers()`

Examples

```
## Not run: fram_db |> compare_recruits()
```

`compare_runs` *Compare key aspects of two FRAM runs*

Description

Outputs summary of comparisons to the console (or optionally to a text file instead). Summary includes output of `compare_non_retention_flags()`, `compare_non_retention_inputs()`, `compare_sl_ratio()` (Chinook only), `compare_recruits()`, `compare_fishery_input_flags()`, `compare_fishery_inputs()`, and `compare_stock_fishery_rate_scalers()` (Coho only).

Usage

```
compare_runs(fram_db, run_ids, save_file = NULL, tolerance = 0.01)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Two run ids. Run names must differ; change in FRAM if necessary.
`save_file`	If provided, diagnostics text is sent to file instead of console. If file already exists, will overwrite. Character, defaults to NULL.
`tolerance`	Minimum proportional change to flag as a "difference" for relevant comparisons (comparison of recruits, fishery inputs). Numeric, defaults to 0.01 (e.g., 1% change).

Value

invisibly returns a list of the comparison dataframes: `$retention_flags`, `$retention_inputs`, `$sl_ratio`, `$recruits`, `fishery_flags`, `$fishery_inputs`, `$sfrs`

See Also

Other comparisons: `compare_fishery_input_flags()`, `compare_fishery_inputs()`, `compare_inputs()`, `compare_inputs_chart()`, `compare_non_retention_inputs()`, `compare_recruits()`, `compare_sl_ratio()`, `compare_stock_fishery_rate_scalers()`

Examples

```
## Not run: fram_db |> compare_runs(c(55, 56))
```

compare_sl_ratio *Compare Sublegal Ratio tables between two runs.*

Description

Provides a dataframe that compares the "SLRatio" table for two runs identified by run_ids. Only works for Chinook databases (Coho do not have an SLRatio table).

Usage

```
compare_sl_ratio(fram_db, run_ids)
```

Arguments

fram_db	FRAM database object
run_ids	Vector of two run_ids

Value

Data frame of differences. *_original columns show the values in the first run of run_ids, while *_comparison show the values of the second run of run_ids. *_diff = original - comparison.

See Also

Other comparisons: compare_fishery_input_flags(), compare_fishery_inputs(), compare_inputs(), compare_inputs_chart(), compare_non_retention_inputs(), compare_recruits(), compare_runs(), compare_stock_fishery_rate_scalers()

Examples

```
## Not run: fram_db |> compare_sl_ratio(c(100,101))
```

compare_stock_fishery_rate_scalers
Compares the stock fishery rate scalers of two runs

Description

Only relevant for Coho runs.

Usage

```
compare_stock_fishery_rate_scalers(fram_db, run_ids)
```

Arguments

fram_db	FRAM database object
run_ids	Two run ids

Value

Tibble of any stock x fishery x timesteps in which the Stock Fishery Rate Scalers (SFRS) changed. $stock_id, $stock_label, $fishery_id, $fishery_label, and $time_step identify the stock x fishery x timestep, and $sfrs_original and $sfrs_comparison list the SFRS values in the first and second runs, respectively.

See Also

Other comparisons: compare_fishery_input_flags(), compare_fishery_inputs(), compare_inputs(), compare_inputs_chart(), compare_non_retention_inputs(), compare_recruits(), compare_runs(), compare_sl_ratio()

Examples

```
## Not run: fram_db |> compare_stock_fishery_rate_scalers(c(55, 56))
```

---

connect_fram_db *Connect to FRAM database*

Description

This produces a connection object to a FRAM database, which is a necessary precursor for almost all framrsquared functions. For most users, can just treat the connection object as a black box that's used as an argument in other functions. See details for an explanation of the returned object.

Usage

```
connect_fram_db(db_path, read_only = FALSE, quiet = FALSE)
```

Arguments

db_path	Path to a FRAM database.
read_only	Logical, defaults to FALSE. Optional argument to flag this connection as read-only (if set to TRUE). If TRUE, framrsquared functions that modify the database will abort rather than run. Use as a safety feature when working with a database that *must not* be modified.
quiet	Logical, defaults to FALSE. Optional argument; when TRUE, silences success message and database summary.

Details

The returned object of `connect_fram_db()` is a list of useful objects for other framrsquared functions.

`$fram_db_connection` connection object, used for SQL calls.

`$fram_db_connection_id` connection name in .fram_connections, used for orphan cleanup via `disconnect_all_fram_connections()`.

`$fram_db_type` "full" or "transfer", useful for validation for specific functions.

`$fram_db_species` "COHO" or "CHINOOK"

`$fram_db_medium` filetype, typeically "mdb"

`$fram_read_only` Optional user-specified safety measure, TRUE or FALSE. Functions that modify the fram database should error out if TRUE.

framrsquared creates a special environment in the global environment, `.fram_connections`. Whenever a new connection is made with `connect_fram_db`, it is added to that environment as well; whenever the connection is closed with `disconnect_fram_db()`, the connection is removed from that environment. This tracking allows `disconnect_all_fram_connections()` to clear out any orphan connections made by assigning a connection to an existing connection object.

See Also

Other connections: `disconnect_all_fram_connections()`, `disconnect_fram_db()`, `list_extant_fram_connections(`

Examples

```
## Not run: fram_db <- connect_fram_db('<path>')
fram_db |> fetch_table("Mortality")
## End(Not run)
```

`copy_fishery_scalers` *Copying scaler inputs from one run to another*

Description

Experimental. DANGEROUS.

Usage

```
copy_fishery_scalers(fram_db, from_run, to_run, fishery_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`from_run`	Run ID to be copied from
`to_run`	Run ID to be copied to
`fishery_id`	ID or IDs for specific fishery(s) to copy inputs to/from. If not provided, interactive option to copy inputs for all fisheries.

Value

Nothing

Examples

```
## Not run: framdb |> copy_fishery_scalers(from_run = 132, to_run = 133, fishery_id = 87)
```

---

copy_run **[Experimental]** *Copies a run any number of times*

Description

Useful for setting up scenario modeling or sensitivity analyses. If also working with TAMMs, consider `make_batch_runs()` to combine copying run and TAMMs.

Usage

```
copy_run(
  fram_db,
  target_run,
  times = 1,
  label = "copy",
  force_many_runs = FALSE,
  verbose = TRUE
)
```

Arguments

`fram_db`	FRAM database object
`target_run`	Run ID to be copied from
`times`	Number of copies
`label`	Label of each copy e.g. copy 1, copy 2
`force_many_runs`	`copy_runs` has failsafe to keep total run number no more than 500. This is expected to be the approximate limit for .mdb file size after runs have been run. When `force_man_runs` is `TRUE`, ignore this failsafe.
`verbose`	Show warning message about run count? Official FRAM is hard-coded to only handle databases with <= 150 runs in them. If `TRUE` (default), provides alert when updated database will exceed this.

Details

FRAM is stored in an access database; these have hard size limits of 2GB. Chinook and Coho are expected to reach this limit with ~540 runs. This function includes a failsafe to prevent databases from exceeding 500 runs. This can be overridden with optional `force_many_runs` argument.

Value

Invisibly returns the run ids of the new runs.

See Also

`copy_tamm()`, `make_batch_runs()`

Examples

```
## Not run: framdb |> copy_run(target_run = 141, times = 1)
```

`copy_tamm` **[Experimental]** *Copy TAMM for FRAM batch runs*

Description

Preps a folder for batch running in the 'Run Multiple Runs' screen of the FRAM automation fork (https://github.com/FRAMverse/FRAM_automation), for use with the `advanced` approach to identify multiple runs. Typically users will want to use `make_batch_runs()` instead (which copies runs and then uses `copy_tamm` to create tamms that match the new runs).

Usage

```
copy_tamm(tamm_name, target_folder, run_id_vec, overwrite = FALSE)
```

Arguments

`tamm_name`	TAMM file to copy, including file path. Character string
`target_folder`	directory to put new batch TAMM files into. Character string
`run_id_vec`	vector of run_ids (numeric or character), corresponding to run ids in a FRAM database.
`overwrite`	If one or more files already exist in `target_folder` with names matching the combination of `tamm_name` and run ids, overwrite (`TRUE`) or leave those files untouched (`FALSE`). Defaults to `FALSE` for safety; recommend setting to `TRUE` to avoid confusion when iterating on work.

Details

One TAMM file will be copied multiple times in the `target_folder` with suffixes based on the `run_id_vec` argument. For automatic use in FRAM, those suffixes should match the run ids of the associated FRAM runs. The "Use folder" button on the "Run Multiple Runs" screen can then use the target folder to set up large batch runs.

Value

invisibly returns logical vector of `file.copy()` success.

See Also

copy_run(), make_batch_runs()

Examples

```
## Not run: copy_tamm(tamm_name = "C:/TAMMs/Chin2020.xlsx",
target_folder = "C:/Batch_run_5", run_id_vec = 10:20)
## End(Not run)
```

---

create_soncc_pasteable

Calculate SONCC ER breakdown for STT

Description

Creates a copy-pasteable excel file with the ER breakdown needed for the SONCC calculator.

Usage

```
create_soncc_pasteable(fram_db, run_id, filename)
```

Arguments

fram_db	FRAM database connection
run_id	Run id of run to calculate SONCC for
filename	filename (including filepath) to save copy-paste ready SONCC breakdowns. Should end in .xlsx

Value

nothing

See Also

Other soncc: calculate_soncc(), format_soncc_pasteable()

Examples

```
## Not run:
library(here)
fram_db <- connect_fram_db(here("2026NOF_CohoFRAMdatabase_DRAFT.mdb"))
create_soncc_pasteable(fram_db, run_id = 152, filename = here("soncc_copy_ready.xlsx"))

## End(Not run)
```

`disconnect_all_fram_connections` *Clear all connections*

Description

Removes all FRAM connections in use in current R session.

Usage

```
disconnect_all_fram_connections()
```

Details

It is relatively easy to create an "orphan" connection using framrsquared by assigning a connection to `fram_db`, and then assigning another connection to `fram_db` without disconnecting the first connection using `disconnect_fram_db()`. Orphaned connections can make it frustrating to work with database files (moving, deleting, etc) without restarting rstudio or rebooting your computer. `disconnect_all_fram_connections()` disconnects any existing connections made by framrsquared in this R session.

Value

nothing

See Also

Other connections: `connect_fram_db()`, `disconnect_fram_db()`, `list_extant_fram_connections()`

Examples

```
## Not run:
fram_db = connect_fram_db("Chin2025.mdb")
# create an orphan by overwriting fram_db with a new connection:
fram_db = connect_fram_db("Chin2025.mdb")
disconnect_fram_db(fram_db)

list_extant_fram_connections()
## oops, still a connection left.

disconnect_all_fram_connections()

list_extant_fram_connections

## End(Not run)
```

disconnect_fram_db *Safely disconnect from FRAM database*

Description

Safely disconnect from FRAM database

Usage

```
disconnect_fram_db(fram_db, quiet = TRUE)
```

Arguments

fram_db	FRAM database R object
quiet	Logical. Optional; when true, silences success message.

See Also

Other connections: connect_fram_db(), disconnect_all_fram_connections(), list_extant_fram_connections()

Examples

```
## Not run: disconnect_fram_db(fram_db)
```

fetch_quarto_templates *Creates quarto template files*

Description

Creates template files from specified organization in the specified path. Generally initialize_project() will be more useful for new R projects, while fetch_quarto_templates() can be helpful when working with existing projects. See initialize_project() for details on adding template files for new organizations.

Usage

```
fetch_quarto_templates(
  to.path,
  organization = c("WDFW"),
  color = "coffee",
  overwrite = FALSE
)
```

Arguments

to.path	Character string. Destination file path for template files. Typically, root of Rproject directory.
organization	Character, defaults to "WDFW". Specifies the set of quarto templates to use. Currently only supports "WDFW".
color	Character string, defaults to "coffee". Specifies quarto template to use; organizations may have several.
overwrite	Boolean. Overwrite _quarto.yml and style.css files if they already exist? Defaults to FALSE.

Value

Nothing.

See Also

initialize_project()

---

fetch_table *Fetch a complete table from a FRAM database.*

Description

Fetch a FRAM database table in R-friendly form (tibble with clean names). If no table_name argument is given, instead list available table names.

Usage

```
fetch_table(fram_db, table_name = NULL, label = TRUE, warn = TRUE)

fetch_table_(fram_db, table_name = NULL, warn = TRUE)
```

Arguments

fram_db	FRAM database object
table_name	Atomic character of name of table to be fetched. Optional; if not provided, a list of available table names will be printed.
label	Add human-readable columns for flags, fisheries, stocks? Based on the Stock and Fishery tables of the current database. Logical, defaults to TRUE.
warn	Print a warning when fetching BackwardsFRAM table from a Chinook database? Logical, defaults to TRUE.

Details

WARNING: the Chinook "BackwardsFRAM" table uses a *different* stock_id numbering system from every other table. To avoid errors when joining that with other tables, instead fetch with `fetch_table_bkchin()`. `fetch_table_()` is an alias for `fetch_table()` with the optional argument `label` set to FALSE.

Value

Tibble version of .Mdb table, with CamelCase switched to snake_case. By default, adds `fishery_label` and `stock_label` (human readable columns) to tables with `fishery_id` and `stock_id` in them, respectively. (see optional argument `label`).

Examples

```
## Not run:
fram_db <- connect_fram_db("validate2024.mdb")
fram_db |> fetch_table('Mortality')
disconnect_fram_db(fram_db)

## End(Not run)
```

---

fetch_table_bkchin *Safely fetch Chinook BackwardsFRAM table*

Description

The BackwardsFRAM table uses a stock_id different numbering system from all other tables, and includes sums of joint stocks (e.g. for a marked and unmarked pair of stocks, BackwardsFRAM will typically have an additional stock which represents the sum of those two). Because the numbering is different but the column name is the same, joining the Chinook BackwardsFRAM table with other tables can cause problems.

Usage

```
fetch_table_bkchin(fram_db)
```

Arguments

`fram_db`	FRAM database object

Details

This function augments fetch_table by renaming the `stock_id` column to `bk_stock_id`, and then adding on the associated stock_id (with NAs when the bkfram stock is one of these new "sum" stocks and the associated bkfram stock names). This function only works for Chinook databases.

The resulting dataframe will necessarily NOT be an exact match to the BackwardsFRAM table in the FRAM database. The stock_id column will differ (containing normal stock ID values instead of bk stock ID values), and there will be two additional columns.

Value

See fetch_table()

Examples

```
## Not run:
##Potentially problematic stock_id won't align with other tables
fram_db |> fetch_table('BackwardsFRAM')
## "safe" version of the table; stock_id WILL align with other tables
fram_db |> fetch_table_bkchin()

## End(Not run)
```

---

filter_ak *Filters a dataframe to Alaska (AK) fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. .data must have a fishery_id column name.

Usage

```
filter_ak(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a fishery_id column. If return_ids = TRUE, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: filter_bc(), filter_ca(), filter_coast(), filter_commercial_wa_nt(), filter_marine(), filter_net(), filter_or(), filter_puget_sound(), filter_sport(), filter_stt(), filter_stt_nt(), filter_wa()

Examples

```
framrosetta::fishery_chinook_fram |> filter_ak(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_ak(species = "COHO")
```

filter_bc *Filters a dataframe to Canadian (BC) fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_bc(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_ca()`, `filter_coast()`, `filter_commercial_wa_nt()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt()`, `filter_stt_nt()`, `filter_wa()`

Examples

```
framrosetta::fishery_chinook_fram |> filter_bc(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_bc(species = "COHO")
```

filter_ca *Filters a dataframe to California (CA) fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_ca(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a fishery_id column. If return_ids = TRUE, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: filter_ak(), filter_bc(), filter_coast(), filter_commercial_wa_nt(), filter_marine(), filter_net(), filter_or(), filter_puget_sound(), filter_sport(), filter_stt(), filter_stt_nt(), filter_wa()

Examples

```
framrosetta::fishery_chinook_fram |> filter_ca(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_ca(species = "COHO")
```

---

filter_coast *Filters a dataframe to Coastal fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. .data must have a fishery_id column name.

Usage

```
filter_coast(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_commercial_wa_nt()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt()`, `filter_stt_nt()`, `filter_wa()`

Examples

```
framrosetta::fishery_chinook_fram |> filter_coast(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_coast(species = "COHO")
```

---

filter_commercial_wa_nt

Filters a dataframe to WA non-treaty commercial fisheries.

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_commercial_wa_nt(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_coast()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt()`, `filter_stt_nt()`, `filter_wa()`

Examples

```
framrosetta::fishery_chinook_fram |> filter_commercial_wa_nt(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_commercial_wa_nt(species = "COHO")
```

`filter_hatchery` *Filters a Coho stock dataframe to hatchery stocks*

Description

Currently only works on Coho datasets. `.data` must have a `stock_id` column name.

Usage

```
filter_hatchery(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the stock ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `stock_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other stock_filters: `filter_mixed()`, `filter_wild()`

Examples

```
framrosetta::stock_coho_fram |> filter_hatchery()
```

filter_marine *Filters a dataframe to marine fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_marine(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_coast()`, `filter_commercial_wa_nt()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt()`, `filter_stt_nt()`, `filter_wa()`

Examples

```
framrosetta::fishery_chinook_fram |> filter_marine(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_marine(species = "COHO")
```

filter_mixed *Filters a Coho stock dataframe to mixed stocks*

Description

Currently only works on Coho datasets. `.data` must have a `stock_id` column name.

Usage

```
filter_mixed(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the stock ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a stock_id column. If return_ids = TRUE, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other stock_filters: filter_hatchery(), filter_wild()

Examples

```
framrosetta::stock_coho_fram |> filter_mixed()
```

---

filter_net *Filters a dataframe to net fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. .data must have a fishery_id column name.

Usage

```
filter_net(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a fishery_id column. If return_ids = TRUE, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: filter_ak(), filter_bc(), filter_ca(), filter_coast(), filter_commercial_wa_nt(), filter_marine(), filter_or(), filter_puget_sound(), filter_sport(), filter_stt(), filter_stt_nt(), filter_wa()

Examples

```
framrosetta::fishery_chinook_fram |> filter_net(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_net(species = "COHO")
```

`filter_or` *Filters a dataframe to Oregon (OR) fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_or(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: filter_ak(), filter_bc(), filter_ca(), filter_coast(), filter_commercial_wa_nt(), filter_marine(), filter_net(), filter_puget_sound(), filter_sport(), filter_stt(), filter_stt_nt(), filter_wa()

Examples

```
framrosetta::fishery_chinook_fram |> filter_or(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_or(species = "COHO")
```

filter_puget_sound *Filters a dataframe to Puget Sound fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_puget_sound(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_coast()`, `filter_commercial_wa_nt()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_sport()`, `filter_stt()`, `filter_stt_nt()`, `filter_wa()`

Examples

```
framrosetta::fishery_chinook_fram |> filter_puget_sound(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_puget_sound(species = "COHO")
```

filter_sport *Filters a dataframe to sport fisheries.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_sport(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a fishery_id column. If return_ids = TRUE, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: filter_ak(), filter_bc(), filter_ca(), filter_coast(), filter_commercial_wa_nt(), filter_marine(), filter_net(), filter_or(), filter_puget_sound(), filter_stt(), filter_stt_nt(), filter_wa()

Examples

```
framrosetta::fishery_chinook_fram |> filter_sport(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_sport(species = "COHO")
```

---

filter_stt *Filters a Coho dataframe to STT fisheries*

Description

Currently only works on Coho datasets. .data must have a fishery_id column name.

Usage

```
filter_stt(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_coast()`, `filter_commercial_wa_nt()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt_nt()`, `filter_wa()`

Examples

```
framrosetta::fishery_coho_fram |> filter_stt()
```

---

filter_stt_nt *Filters a Coho dataframe to non-treaty STT fisheries*

Description

Currently only works on Coho datasets. `.data` must have a `fishery_id` column name.

Usage

```
filter_stt_nt(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_coast()`, `filter_commercial_wa_nt()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt()`, `filter_wa()`

Examples

```
framrosetta::fishery_coho_fram |> filter_stt_nt()
```

filter_wa *Filters a dataframe to fisheries in Washington state.*

Description

Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. `.data` must have a `fishery_id` column name.

Usage

```
filter_wa(.data, species = NULL, return_ids = FALSE)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to `NULL`
`return_ids`	Return the fishery ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a `fishery_id` column. If `return_ids = TRUE`, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other fishery_filters: `filter_ak()`, `filter_bc()`, `filter_ca()`, `filter_coast()`, `filter_commercial_wa_nt()`, `filter_marine()`, `filter_net()`, `filter_or()`, `filter_puget_sound()`, `filter_sport()`, `filter_stt()`, `filter_stt_nt()`

Examples

```
framrosetta::fishery_chinook_fram |> filter_wa(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_wa(species = "COHO")
```

filter_wild *Filters a Coho stock dataframe to wild stocks*

Description

Currently only works on Coho datasets. `.data` must have a `stock_id` column name.

Usage

```
filter_wild(.data, species = NULL, return_ids = FALSE)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL
return_ids	Return the stock ids used in filtering rather than a filtered dataframe? Logical, defaults to FALSE

Value

A dataframe filtered based on a stock_id column. If return_ids = TRUE, instead returns numeric vector of the fishery IDs used to filter.

See Also

Other stock_filters: filter_hatchery(), filter_mixed()

Examples

```
framrosetta::stock_coho_fram |> filter_wild()
```

---

fishery_mortality *Returns a tibble matching the Fishery Mortality screen.*

Description

Returns a tibble matching the Fishery Mortality screen.

Usage

```
fishery_mortality(fram_db, run_id = NULL, fishery_id = NULL, msp = TRUE)
```

Arguments

fram_db	FRAM database object
run_id	atomic or vector of run_ids to filter to. Can improve speed. Optional, defaults to NULL.
fishery_id	atomic or vector of fishery_id to filter to. Can improve speed. Optional, defaults to NULL.
msp	Use Model Stock Proportion? Logical, defaults to TRUE. Only relevant for Chinook.

Value

Tibble identifying, run, fishery, age, timestep, and providing calculations of mortalities by type, aggregating across NS and MSF.

Examples

```
## Not run:
fram_db |> fishery_mortality(run_id = 101)

## End(Not run)
```

`initialize_project` **[Experimental]** *Initializes a FRAM project*

Description

By default, creates suggested folder structure from best coding practices, and adds WDFW-style yaml and style.css files to give quarto files consistent appearance. If you belong to another organization and want this function to support your own organization-specific quarto styling, reach out to the developers with a `_quarto.yml` and (optionally) `style.css` file.

Usage

```
initialize_project(
  folders = c("scripts", "original_data", "cleaned_data", "figures", "results",
    "results/quarto_output"),
  quarto = TRUE,
  organization = c("WDFW"),
  renv = FALSE,
  template_overwrite = TRUE,
  color = "coffee",
  quiet = TRUE
)
```

Arguments

`folders`	Vector of folders to create
`quarto`	Boolean. If TRUE, add quarto yaml file and style.css
`organization`	Character, defaults to "WDFW". Specifies the set of quarto templates to use. Currently only supports "WDFW".
`renv`	Boolean to initialize renv. Defaults to FALSE.
`template_overwrite`	Boolean. Overwrite _quarto.yml and style.css files if they already exist? Defaults to TRUE
`color`	Character string, defaults to "coffee". Specifies quarto template to use; organizations may have several.
`quiet`	Boolean, defaults to FALSE. If TRUE, suppresses informational messages.

Details

Dev note: new template files for additional organizations should be added to `inst/` in a subfolder matching an R-friendly organization name, and the same name should be added to the `organization` parameter description here and the `supported_organizations` in `fetch_quarto_templates()`.

Value

nothing

See Also

`fetch_quarto_templates()`

Examples

```
## Not run:
framrsquared.dev::initialize_project()

## End(Not run)
```

---

label_fisheries_db *Label fisheries based on FRAM database*

Description

Like `framrosetta::label_fisheries()`, but uses an active FRAM database to label fisheries, rather than the look-up table present in the framrosetta package. Primarily used in `fetch_table()`, robust to changes in base period.

Usage

```
label_fisheries_db(.data, fram_db)
```

Arguments

`.data`	Dataframe containing ‘fishery_id‘ column (or analogous column with different name specied by ‘fisheries_col‘ argument)
`fram_db`	FRAM database connection

Value

`.data` with additional column, `$fishery_label`

See Also

Other labelers: `label_flags()`, `label_stocks_db()`, `label_timesteps_db()`

label_flags *Provide flag translations to dataframe*

Description

Adds a column with a text version of flags for either non-retention or fishery scalers.

Usage

```
label_flags(.data, species = NULL, warn = TRUE)
```

Arguments

.data	Dataframe with either $fishery_flag or $non_retention_flag columns. Typically a fetched FisheryScalers or NonRetentions table.
species	Optional, identifying species if .data doesn't. If provided, should be "CHINOOK" or "COHO" (or variants)
warn	Logical, defaults to TRUE. Warn if neither flag column is present in dataframe?

Value

dataframe .data with additional character vector columns $fishery_flag_label and/or non_retention_flag_label depending on the presence of $fishery_flag and $non_retention.

See Also

Other labelers: label_fisheries_db(), label_stocks_db(), label_timesteps_db()

Examples

```
## Not run:  fram_db |>
fetch_table("FisheryScalers") |>
label_flags()

## End(Not run)
```

label_stocks_db *Label stocks based on FRAM database*

Description

Like framrosetta::label_stocks(), but uses an active FRAM database to label stocks, rather than the look-up table present in the framrosetta package. Primarily used in fetch_table(), robust to changes in base period.

Usage

```
label_stocks_db(.data, fram_db)
```

Arguments

.data	Dataframe containing 'stock_id' column (or analogous column with different name specied by 'stocks_col' argument)
fram_db	FRAM database connection

Value

.data with additional column, $stock_label

See Also

Other labelers: label_fisheries_db(), label_flags(), label_timesteps_db()

---

label_timesteps_db *Label timesteps based on FRAM database*

Description

Label timesteps based on FRAM database

Usage

```
label_timesteps_db(.data, fram_db)
```

Arguments

.data	Database with time_step column.
fram_db	FRAM database connection

Value

.data with additional column, $time_step_label

See Also

Other labelers: label_fisheries_db(), label_flags(), label_stocks_db()

list_extant_fram_connections *Describe existing fram connections (including orphans)*

Description

Provides information in the terminal, including the number of existing FRAM database connections created in this R session using framrsquared, as well as the files those connections connect to. Note that there may be multiple connections to the same file.

Usage

```
list_extant_fram_connections()
```

Value

Invisibly returns the number of extant FRAM connections

See Also

Other connections: `connect_fram_db()`, `disconnect_all_fram_connections()`, `disconnect_fram_db()`

make_batch_runs *Make batch runs*

Description

Make multiple copies of a FRAM run, make copies of specified TAMM in `target_folder` with run_id suffixes matching newly created runs. Intended to streamline using the multi-run fork of FRAM. Primarily for use internally, in the sensitivity analysis functions

Usage

```
make_batch_runs(
  fram_db,
  target_run,
  tamm_name,
  target_folder,
  times = 1,
  label = "copy",
  force_many_runs = FALSE,
  verbose = TRUE
)
```

Arguments

fram_db	Fram database connection
target_run	Run id of target run
tamm_name	Filepath/name of tamm to copy
target_folder	Location TAMMs should be saved
times	number of run copies to make. Numeric, defaults to 1.
label	Title suffix for newly created runs. Character, defaults to 'copy'. An index number is added after this suffix to distinguish copied runs.
force_many_runs	Ignore limits on number of runs in fram database? Logical, defaults to FALSE.
verbose	Print details to console? Logical, defaults to TRUE.

See Also

copy_tamm(), copy_run()

---

management_week *Vectorized approach to calculating the management week*

Description

Vectorized approach to calculating the management week

Usage

```
management_week(date)
```

Arguments

date	An atomic or vector of dates

Value

a numeric vector with same length as argument date

Examples

```
management_week(as.Date(Sys.Date()))
## Not run:
data_fram |>
  mutate(mngmt_week = management_week(date_field))

## End(Not run)
```

`modify_table` **[Experimental]** *Modify FRAM database based on match/replace dataframe*

Description

Uses a special match/replace dataframe to modify values in a FRAM table.

Usage

```
modify_table(fram_db, table_name, df)
```

Arguments

`fram_db`	FRAM database
`table_name`	Name of FRAM table
`df`	The match/replace dataframe or tibble with specially named columns. Columns must start with either "match_" or "replace_", and should otherwise match the names of columns in `table`. For example, modifications to the Cohort table might be achieved with columns "match_RunID", "match_StockID", "match_age", "match_TimeStep", "replace_StartCohort". See Details.

Details

At a high level, modifying a FRAM table requires identify which rows to change, and then replacing the values of one or more of the columns of those row with new values. We often want to make multiple changes at once, and `modify_table` is written around using a dataframe to define the matching and replacing, so that it is relatively easy to check all of the changes being made. This dataframe (hereafter the "match/replace dataframe") should have column names starting with "match_" and "replace_", and ending with the exact match of column names in the FRAM table identified with argument `table_name`. For each row of argument `df`, `modify_table()` will use columns starting with "match_" as conditions to identify rows in the FRAM database to modify, and then for those rows will replace the values of columns identified with "replace_" with the corresponding values in the `df` columns.

As a simple example, imagine we want to see how modifying the size limits for Area 7 Sport (chinook fishery id 36) affect our ERs. We should start by using copy_run to create multiple duplicate runs. Once that is done, we can use `modify_table` to change just the `MinimumSize` values of the "SizeLimits" table for just those rows for which fishery id was 36. If our run ids were 100, 101, and 102, and we wanted to look at minimum sizes of 450, 550, and 650, our `df` argument might look like `data.frame(match_RunID = c(100, 101, 102), match_FisheryID = c(36, 36, 36), replace_MinimumSize = c(450, 550, 650))`.

We can create `df` programmatically to combine different run ids with multiple changes at once or to apply some kind of randomized parameter sampling scheme. Or we could even use an excel sheet to write out the experiment in a `df` format and then read in the sheet and feed it into `modify_table`.

See Also

calc_fram_scaling()

Examples

```
## Example: For ages 3, 4, and 5 of stock 100 in run 396,
##   in the StockRecruit scalar change the recruit scale factor
##   to have values of 1, 2, and 3 respectively, and the recruit
##   cohort size to have values of 100, 101, and 102.
df_total <- data.frame(match_Age = c(3, 4, 5))
df_total$match_StockID <- 100
df_total$match_RunID <- 396
df_total$replace_RecruitScaleFactor <- 1:3
df_total$replace_RecruitCohortSize <- 100:102
## let's look at our match/replace dataframe:
df_total
## Not run:
fram_db |> modify_db(table_name = "StockRecruit", df = df_total)

## End(Not run)
```

---

mortality_scalers *Quantify the proportion of fishery mortalities associated with stock(s) of interest*

Description

Supports guestimating the impact of making changes to a fishery on a particular stock (or group of stocks) by finding the ratio of mortalities in each fishery that can be attributed to the focal stock or stocks. Accounts for *all* sources of mortality (e.g., includes non-retention, dropoff, etc). Chinook runs are returned in units of AEQ.

Usage

```
mortality_scalers(fram_db, run_id, stock_id, msp = FALSE)
```

Arguments

fram_db	FRAM database object
run_id	Run ID
stock_id	A focal stock or stocks.
msp	Should we use MSP (Model Stock Proportion) expansion? Logical, defaults to FALSE. Only relevant for Chinook.

Value

Tibble identify run, fishery, and timestep. `$fishery_mortality` gives the total mortalities in each fishery x timestep, `$stock_mortality` gives the total mortalities of the focal stock or stocks, and `$stock_mortality_ratio` gives the fraction of fishery mortalities that can be attributed to the focal stock or stocks.

See Also

`plot_impacts_per_catch_heatmap()`

Examples

```
## Not run: fram_db |> mortality_scalers(run_id = 101, stock_id = c(17:18))
```

---

msf_encounters *Reproduce MSF encounters screen*

Description

Produces the MSF screen report numbers for encounters. Returns different format depending database.

Usage

```
msf_encounters(fram_db, run_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`run_id`	Run ID

See Also

`msf_mortalities()`, `msf_landed_catch()`

Examples

```
## Not run: fram_db |> msf_encounters(run_id = 101)
```

msf_landed_catch *Reproduce MSF landed catch screen*

Description

Produces the MSF screen report numbers for landed catch. Returns different format depending database.

Usage

```
msf_landed_catch(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID

See Also

msf_encounters(), msf_mortalities()

Examples

```
## Not run: fram_db |> msf_landed_catch(run_id = 101)
```

msf_mortalities *Reproduce MSF mortalities screen*

Description

Produces the MSF screen report numbers for mortalities. Returns different format depending database.

Usage

```
msf_mortalities(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID

Value

Tibble identifying run, fishery, timestep. For Coho database, $marked and $unmarked columns. for Chinook database, also identifies by legal and sublegal.

See Also

msf_encounters(), msf_landed_catch()

Examples

```
## Not run: fram_db |> msf_mortalities_coho_(run_id = 101)
```

`msp_mortality` *Expand Chinook mortality table using Model-Stock Proportion*

Description

See https://framverse.github.io/fram_doc/calcs_data_chin.html#46_Model-Stock_Proportion.

Usage

```
msp_mortality(fram_db, run_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`run_id`	One or more run ids (optional)

Value

Mortality table with mortality values expanded by msp

See Also

aeq_mortality()

Examples

```
## Not run:
fram_db |> msp_mortality(run_id = 132)

## End(Not run)
```

na_non_retention_from_flag *NA's unused CNR input columns.*

Description

Turns values in `$cnr_input_*` columns of a non-retention table into NAs if the `$non_retention_flag` column indicates they're not being used. For COHO databases, `cnr_input_1`

Usage

```
na_non_retention_from_flag(.data)
```

Arguments

`.data`	Dataframe of the Fishery Scalers table

Value

dataframe `.data`, with some values of `cnr_input1:cnr_input4` converted to NAs.

Examples

```
data = data.frame(non_retention_flag = 0:4,
                  cnr_input1 = sample(50:150, size = 5),
                  cnr_input2 = sample(50:150, size = 5),
                  cnr_input3 = sample(50:150, size = 5),
                  cnr_input4 = sample(50:150, size = 5)
)
## needs a species attribute -- this is automatically applied when using `fetch_table()`
attr(data, "species") <- "CHINOOK"
## here's what it looks like before applying the function
data
## applying the function:
data |>
  na_non_retention_from_flag()
```

na_scalers_from_flag *NA's unused scalers*

Description

Turns values in scaler columns (`fishery_scale_factor`, `msf_fishery_scale_factor`, `quota`, and `msf_quota`) into NAs if the `fishery_flag` column indicates they're not being used. e.g if `fishery_flag` is 1, `fishery_scale_factor` value will be left alone, but the `msf_fishery_scale_Factor`, `quota`, and `msf_quota` values will be turned into NAs.

Usage

```
na_scalers_from_flag(.data)
```

Arguments

.data	Dataframe of the Fishery Scalers table

Value

dataframe .data but with some values of the scaler columns replaced with NAs.

Examples

```
data = data.frame(fishery_flag = c(1, 2, 7, 8, 17, 18, 27, 28),
                  fishery_scale_factor = runif(8)*2,
                  quota = sample(500:10000, size = 8),
                  msf_fishery_scale_factor = runif(8)*2,
                  msf_quota = sample(500:10000, size = 8)
)
## here's what it looks like before applying the function
data
## applying the function:
data |>
  na_scalers_from_flag()
```

---

plot_impacts_per_catch_heatmap

Make plots to show the amount of landed catch_per_impact

Description

Identify how much reduction in landed catch at each fishery that would be needed to reduce the impacts on a focal stock by 1 fish. Does *not* include CNR.

Usage

```
plot_impacts_per_catch_heatmap(
  fram_db,
  run_id,
  stock_id,
  filters_list = list(filter_wa, filter_sport),
  filter_out = NULL,
  msp = TRUE,
  digits_round = 1,
  outer_text_size = 18,
  cell_text_size = 5,
  short_title = FALSE,
  per_thousand_catch = FALSE,
```

```
  verbose = TRUE,
  warn = TRUE
)
```

Arguments

fram_db	fram database connection
run_id	run_id of interest
stock_id	stock_id of interest. Can accept multiple stock_ids, and will plot the impact on the combined stocks.
filters_list	list of framrsquared filter functions to apply before plotting. Defaults to `list(filter_wa, filter_sport)`, which filters to WA sport fisheries.
filter_out	Numeric vector of fishery IDs to filter out. Helpful for removing individual fisheries if they aren't appropriate for the plot (e.g. only non-retention).
msp	Use Model Stock Proportion? Logical, defaults to TRUE. Only relevant for Chinook databases.
digits_round	How many digits should cell values be rounded to? Numeric, defaults to 1.
outer_text_size	Controls size of plot text elements except cell text. Different plot elements scale relative to this value. Numeric defaults to 18.
cell_text_size	Controls size of text size in heatmap cells. Numeric, defaults to 5. Different units from `outer_text_size`.
short_title	Should the abbreviated stock name (e.g., "M-ssdnph") be used (`TRUE`) or the longer name (e.g., "South Puget SOund Net Pens Marked"). Logical, defaults to `FALSE`; `TRUE` may be useful when plots must be small.
per_thousand_catch	Should plot be presented in units of Impacts per Thousand Landed Catch (TRUE) or landed catch per impact (FALSE). Logical, defaults to FALSE.
verbose	Print plot info to console? Logical, defaults to `TRUE`.
warn	Print warning if multiple stocks are provided? Logical, defaults to `TRUE`.

Value

ggplot2 object

See Also

`plot_stock_mortality()`

Examples

```
## Not run:
path = "FRAM compilations - readonly/2024-Pre-Season-Chinook-DB/2024 Pre-Season Chinook DB.mdb"
run_id = 132
stock_id = 3
plot_impacts_per_catch_heatmap(path,
```

```
                                run_id = 132,
                                stock_id = 5)

## End(Not run)
```

`plot_stock_comp` *Plot stock composition*

Description

Produces a stock composition chart. Low frequency stocks are grouped into geographic area.

Usage

```
plot_stock_comp(fram_db, run_id, fishery_id, time_step, group_threshold = 0.01)
```

Arguments

`fram_db`	Fram database object
`run_id`	numeric, RunID
`fishery_id`	numeric, Fishery ID
`time_step`	numeric, Time Step
`group_threshold`	numeric, Stock percentages below this number will be grouped. Default is 1%, setting to zero will turn grouping off

Value

ggplot object

See Also

`calculate_stock_comp()`

Examples

```
## Not run:
fram_db |> stock_comp(run_id = 132)

## End(Not run)
```

`plot_stock_mortality` *Plot total mortalities by fishery*

Description

Creates an ordered bar chart with the top number of mortalities per fishery.

Usage

```
plot_stock_mortality(
  fram_db,
  run_id,
  stock_id,
  top_n = 10,
  filters_list = NULL,
  msp = TRUE,
  split_cnr = FALSE,
  fishery_title_short = FALSE,
  stock_title_short = FALSE,
  warn = TRUE
)
```

Arguments

`fram_db`	fram database object, supplied through connect_fram_db
`run_id`	numeric, RunID
`stock_id`	numeric, ID of focal stock. Can accept multiple ids, but this should only be used when combining FRAM stocks makes sense (e.g., combining marked and unmarked components of the same stock).
`top_n`	numeric, Number of fisheries to display
`filters_list`	list of framrsquared filter functions to apply before plotting.
`msp`	Use Model Stock Proportion? Logical, defaults to TRUE.
`split_cnr`	Produce separate panels for CNR and non-CNR mortality? Logical, defaults to FALSE.
`fishery_title_short`	Use abbreviated fishery names instead of full names? Useful if horizontal space is limited. Logical, defaults to FALSE.
`stock_title_short`	Use abbreviated stock name instead of full name in title? Will always use abbreviated stock name if multiple stock ids are provided.
`warn`	Warn if providing multiple stocks?

Value

ggplot2 object.

See Also

plot_impacts_per_catch_heatmap(), plot_stock_mortality_time_step()

Examples

```
## Not run:
fram_db |> plot_stock_mortality(run_id = 132, stock_id = 17)
fram_db |> plot_stock_mortality(run_id = 132, stock_id = 17,
        filters_list = list(filter_wa, filter_sport))

## End(Not run)
```

---

plot_stock_mortality_time_step

Plot total mortalities by fishery and timestep

Description

Creates an ordered bar chart with the top number of mortalities per fishery and time step.

Usage

```
plot_stock_mortality_time_step(
  fram_db,
  run_id,
  stock_id,
  top_n = 10,
  filters_list = NULL,
  msp = TRUE,
  split_cnr = FALSE,
  fishery_title_short = FALSE,
  stock_title_short = FALSE,
  warn = TRUE
)
```

Arguments

fram_db	fram database object, supplied through connect_fram_db
run_id	numeric, RunID
stock_id	numeric, ID of focal stock. Can accept multiple ids, but this should only be used when combining FRAM stocks makes sense (e.g., combining marked and unmarked components of the same stock).
top_n	numeric, Number of fisheries to display
filters_list	list of framrsquared filter functions to apply before plotting.
msp	Use Model Stock Proportion? Logical, defaults to TRUE.

split_cnr	Produce separate panels for CNR and non-CNR mortality? Logical, defaults to FALSE.
fishery_title_short	Use abbreviated fishery names instead of full names? Useful if horizontal space is limited. Logical, defaults to FALSE.
stock_title_short	Use abbreviated stock name instead of full name in title? Will always use abbreviated stock name if multiple stock ids are provided.
warn	Warn if providing multiple stocks?

Value

ggplot2 object

See Also

plot_stock_mortality(), plot_impacts_per_catch_heatmap()

Examples

```
## Not run:
fram_db |> stock_mortality_time_step(run_id = 132, stock_id = 17)

## End(Not run)
```

---

population_statistics *Replicate Population Statistics screen*

Description

Returns a tibble matching the Population Statistics screen.

Usage

```
population_statistics(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID

Value

Tibble identifying run, stock, age, timestep. Then provides the number of fish present at each substep within a timestep: before any mortalities ($starting_cohort), after natural mortalities but before marine fishing mortalities $post_pre_terminal, the number of fish reaching maturation ($maturation; only relevant for Chinook), and the number of fish reaching escapement ($escapement)

Examples

```
## Not run: fram_db |> population_statistics(run_id = 101)
```

`post_season_abundance` *Generates post-season January age 3 abundances by stock from post-season databases.*

Description

Used for forecasting. Only works for Coho post-season databases. Intended for use with databases that have one run per run year.

Usage

```
post_season_abundance(fram_db, units = c("ja3", "oa3"), run_ids = NULL)
```

Arguments

`fram_db`	FRAM database object
`units`	Default January Age 3 'ja3', optional ocean age 3 'oa3'
`run_ids`	Numeric vector of run_ids to use, necessary when there are multiple runs with the same run_year in the database. Optional, defaults to NULL.

Value

Tibble identify stock, origin ("Hatchery" or "Wild"), and a column for abundances for each run. Abundance columns are labeled by year (if there is only one run per run year) or by run id (e.g., `run_54`) if multiple runs share the same year.

Examples

```
## Not run: framdb |> post_season_abundance()
```

remove_run *Removes a run in a FRAM database*

Description

Removes a run in a FRAM database

Usage

```
remove_run(fram_db, run_id)
```

Arguments

fram_db	FRAM database object
run_id	FRAM run ID or IDs to be deleted

Value

Nothing.

Examples

```
## Not run: fram_db |> delete_run(run_id = 132)
```

run_info *Provides a print out of Run ID information*

Description

Provides a print out of Run ID information

Usage

```
run_info(fram_db, run_id)
```

Arguments

fram_db	FRAM database object
run_id	FRAM run ID

Examples

```
## Not run: fram_db |> run_info(run_id = 132)
```

`sensitivity_custom` **[Experimental]** *Generate sensitivity analyses runs based on a list of match/replace dataframes*

Description

For complex sensitivity analyses, it may be easiest to programmatically create a series of match/replace dataframes (`?modify_table`), one for each sensitivity run. `sensitivity_custom()` uses a list of these dataframes to create a series of sensitivity analyses runs. Otherwise behaves as `sensitivity_exact()` or `sensitivity_scaled()`. Saved log is a .rds file that contains `scenario_list` but with each list item named with the matching RunID.

Usage

```
sensitivity_custom(
  fram_db,
  template_run,
  table_name,
  scenario_list,
  tamm_template = NULL,
  tamm_target_folder = NULL,
  label = "sensitivity",
  save_log = TRUE
)
```

Arguments

`fram_db`	Fram database
`template_run`	Run ID of the run that should be used as a template for the sensitivity analyses.
`table_name`	Name of FRAM table that will be modified for the sensitivity analyses. For list items that are named, be ignored in favor of item name.
`scenario_list`	List of match/replace dataframes as described in documentation of `modify_table()`. If present, list item names are assumed to identify the table to be changed.
`tamm_template`	Optional; character string of filepath of a TAMM to be used as a template. If provided (and tamm_target_folder provided), `sensitivity_scaled` will make a tamm for each sensitivity analysis run, using names that work with the FRAM multirun fork "Use folder" option.
`tamm_target_folder`	Folder to copy TAMMs into. Will create if it does not exist.
`label`	Label added to each of the generated run names to identify this sequence of sensitivity analyses. String, defaults to "sensitivity"
`save_log`	Should a log .csv of the specifics used (row ids, match criterion, scaling or replacement values) be saved in the same folder as the FRAM database? Logical, defaults to TRUE.

Details

Current framework does not support automating creation of sensitivity analyses in which changes are being made to multiple tables for a single run.

Value

Invisibly returns object `scenario_list`, but with list items named with the corresponding RunID.

Examples

```
## Not run:
## silly quick-and-dirty example: try these
## two scenarios: mark release rates of 0.05 and 0.01 for fisheries 1 and 2
## for timestep 1, or flipping those. Modifications to FisheryScalers table
fram_db <- connect_fram_db(here("Valid2024_sens_test.mdb"))

custom_scenarios = list(data.frame(match_FisheryID = c(1, 2),
                                   match_TimeStep = c(1, 1),
                                   replace_MarkReleaseRate = c(.05, .01)),
                        data.frame(match_FisheryID = c(1, 2),
                                   match_TimeStep = c(1, 1),
                                   replace_MarkReleaseRate = c(.01, .05))
)

tamm_template = here("Coho2513NOF-165.xlsx")
tamm_target_folder = here("sens_test_custom/")
fram_db |>
 sensitivity_custom(template_run = 28,
                    table_name = 'FisheryScalers',
                    scenario_list = custom_scenarios,
                    tamm_template = tamm_template,
                    tamm_target_folder = tamm_target_folder,
                    label = "markrelease custom")
disconnect_fram_db(fram_db)

## End(Not run)
```

`sensitivity_exact` **[Experimental]** *Generate sensitivity analyses runs based on exact values*

Description

As `sensitivity_scaled`, but provide exact values for the sensitivity analyses (in argument `exact_values`) instead of scaling factors.

Usage

```
sensitivity_exact(
  fram_db,
  template_run,
  table_name,
  match_df,
  exact_values,
  cols_to_vary,
  tamm_template = NULL,
  tamm_target_folder = NULL,
  label = "sensitivity",
  save_log = TRUE
)
```

Arguments

fram_db	Fram database
template_run	Run ID of the run that should be used as a template for the sensitivity analyses.
table_name	Name of FRAM table that will be modified for the sensitivity analyses
match_df	dataframe that defines which rows should be modified during sensitivity analyses. To modify some values for marked and unmarked Stillaguamish stocks, we would use data.frame(StockID = c(17, 18)). To modify values only for Stillaguamish age 2s, we would use expand_grid(StockID = c(17, 19), Age = 2). Unlike match/replace dataframes for modify_table(), column names do not need to start with "match_" (but this function will still work if they do).
exact_values	numeric vector of values to exact values to use for sensitivity analyses.
cols_to_vary	Character or character vector of column names of FRAM table table_name that should be rescaled.
tamm_template	Optional; character string of filepath of a TAMM to be used as a template. If provided (and tamm_target_folder provided), sensitivity_scaled will make a tamm for each sensitivity analysis run, using names that work with the FRAM multirun fork "Use folder" option.
tamm_target_folder	Folder to copy TAMMs into. Will create if it does not exist.
label	Label added to each of the generated run names to identify this sequence of sensitivity analyses. String, defaults to "sensitivity"
save_log	Should a log .csv of the specifics used (row ids, match criterion, scaling or replacement values) be saved in the same folder as the FRAM database? Logical, defaults to TRUE.

Value

Invisibly returns a list of dataframes. $values_by_run contains a row for each sensitivity run and maps the values used to run ids. $full_df is the full match/scale factor used by calc_fram_scaling, and shows the match conditions and scaling used for each run. If cols_to_vary has length 1, the two dataframes will contain the same information.

See Also

Other Sensitivity: `sensitivity_scaled()`

Examples

```
## Not run:
fram_db <- connect_fram_db(here("Valid2024_sens_test.mdb"))

tamm_template <- here("Coho2513NOF-165.xlsx")
tamm_target_folder <- here("sens_test_exact/")
fram_db |>
  sensitivity_exact(
    template_run = 28,
    table_name = "StockRecruit",
    match_df = data.frame(match_StockID = 1:2),
    exact_values = seq(0.5, 5, by = 0.5),
    cols_to_vary = c("RecruitScaleFactor"),
    tamm_template = tamm_template,
    tamm_target_folder = tamm_target_folder,
    label = "Stilly sensitivity exact"
  )
disconnect_fram_db(fram_db)

## End(Not run)
```

---

sensitivity_scaled **[Experimental]** *Generate sensitivity analyses runs scaled by template values*

Description

From a template FRAM run, for a single vector of scaling factors (e.g., `c(0.5, 2)` would test halving and doubling), generate sensitivity analyses which rescale the columns specied in (`cols_to_vary`) for rows which match the conditions specified in `match_df`. Optionally creates corresponding tamms from a template TAMM, labeled to work with folder loading option in the FRAM multi-run fork.

Usage

```
sensitivity_scaled(
  fram_db,
  template_run,
  table_name,
  match_df,
  scale_values,
  cols_to_vary,
  tamm_template = NULL,
  tamm_target_folder = NULL,
```

```
  label = "sensitivity",
  save_log = TRUE
)
```

Arguments

fram_db	Fram database
template_run	Run ID of the run that should be used as a template for the sensitivity analyses.
table_name	Name of FRAM table that will be modified for the sensitivity analyses
match_df	dataframe that defines which rows should be modified during sensitivity analyses. To modify some values for marked and unmarked Stillaguamish stocks, we would use data.frame(StockID = c(17, 18)). To modify values only for Stillaguamish age 2s, we would use expand_grid(StockID = c(17, 19), Age = 2). Unlike match/replace dataframes for modify_table(), column names do not need to start with "match_" (but this function will still work if they do).
scale_values	Numeric vector of the scaling factors to be be used, one per sensitivity analysis run. Defines the number of runs generated. For example, scale_values = 2:10 would generate 9 runs. The first would multiply the values of interest by 2, the second by 3, etc.
cols_to_vary	Character or character vector of column names of FRAM table table_name that should be rescaled.
tamm_template	Optional; character string of filepath of a TAMM to be used as a template. If provided (and tamm_target_folder provided), sensitivity_scaled will make a tamm for each sensitivity analysis run, using names that work with the FRAM multirun fork "Use folder" option.
tamm_target_folder	Folder to copy TAMMs into. Will create if it does not exist.
label	Label added to each of the generated run names to identify this sequence of sensitivity analyses. String, defaults to "sensitivity"
save_log	Should a log .csv of the specifics used (row ids, match criterion, scaling or replacement values) be saved in the same folder as the FRAM database? Logical, defaults to TRUE.

Details

Dev note: update to allow match_df to NOT start with "match_" – it's implied.

Value

Invisibly returns a list of dataframes. $scales_by_runs contains a row for each sensitivity run and maps the scaling factors to run ids. $full_df is the full match/scale factor used by calc_fram_scaling, and shows the match conditions and scaling used for each run.

See Also

Other Sensitivity: sensitivity_exact()

Examples

```
## Not run:
# Testing sensitivity_scaled
library(here)
fram_db <- connect_fram_db(here("Valid2024_sens_test.mdb"))

tamm_template <- "ChinValidrunTest.xlsx"
tamm_target_folder <- here("sens_test/")
fram_db |>
  sensitivity_scaled(
    template_run = 28,
    table_name = "StockRecruit",
    match_df = data.frame(match_StockID = c(17, 19)),
    scale_values = seq(0.1, 2, length = 10),
    cols_to_vary = c("RecruitScaleFactor", "RecruitCohortSize"),
    tamm_template = tamm_template,
    tamm_target_folder = tamm_target_folder
  )
disconnect_fram_db(fram_db)

## End(Not run)
```

---

statistical_week *Vectorized approach to calculating the statistical week, returns an integer*

Description

Statistical weeks start on mondays, so the first statistical week of the year starts on the first Monday of the year. (Contrast with management weeks which start on Sundays).

Usage

```
statistical_week(date)
```

Arguments

`date`	A vector of dates

Value

a numeric vector with same length as argument `date`

Examples

```
## Not run:
statistical_week(as.Date(Sys.Date()))
data_fram |>
  mutate(mngmt_week = statistical_week(date_field))
```

```
## End(Not run)
```

`stock_fate` **[Experimental]**

Description

Summarize outcomes for stock

Usage

```
stock_fate(fram_db, run_id = NULL, units = c("fish", "percentage"))
```

Arguments

`fram_db`	FRAM database object
`run_id`	Run ID (optional)
`units`	Should fates be presented in 'fish' or 'percentage'? Percentage is proportion of starting abundance (so not actually a percent, but percent/100).

Details

Summarizes the three true outcomes of a stocks abundance, where it either (a) dies to fishery related mortality, (b) dies to natural mortality, or (c) reaches some sort of escapement. When run against the coho database, spawning escapement will be displayed. When run against theChinook database escapement to the river will be displayed along with recruits to the next year 'age_up'

Value

dataframe identifying run_id, stock, and age, and providing `$natural_mortality`, `$fishery_mortality`, and either (a) `$age_up` and `$escapment_to_river` (Chinook database) or (b) `$escapement_spawning`

Examples

```
## Not run:
fram_db |> stock_fate(run_id = 145)

## End(Not run)
```

stock_mortality *Replicate Stock Mortality screen*

Description

Returns a tibble matching the Stock Mortality screen.

Usage

```
stock_mortality(fram_db, run_id = NULL, stock_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	atomic or vector of run_ids to filter to. Can improve speed. Optional, defaults to NULL.
stock_id	atomic or vector of stock_id to filter to. Can improve speed. Optional, defaults to NULL.

Value

Tibble matching the "Stock Mortality" screen of the FRAM interface.

Examples

```
## Not run:
fram_db |>
 stock_mortality(run_id=132) |>
 filter(stock_id == 17, fishery_id == 36)


## End(Not run)
```

terminal_fisheries **[Experimental]** *List terminal stock information*

Description

For each TAA, lists the associated fisheries. Intended to support working with bios for QAQC.

Usage

```
terminal_fisheries(fram_db, species = NULL)
```

Arguments

fram_db	Fram database object
species	"COHO" or "CHINOOK". Optional, defaults to the database species. Provide this only if fram_db connects to a database with both Chinook and Coho information. And try to avoid that – those databases are sketchy to work with.

Value

Tibble of taa fisheries

Examples

```
## Not run: fram_db |> terminal_fisheries()
```

---

terminal_info **[Experimental]** *Parse TAAETRS table*

Description

Terminal run information used by FRAM is stored in the TAAETRSList and (soon) the TAAETRSListChinook tables, but stored in a way that is not very human readable. `parse_terminal_info()` translates this to human-readable form, primarily to then be used by `terminal_stocks()` and `terminal_fisheries()`.

Usage

```
terminal_info(fram_db, old_table_name = TRUE, species = NULL)
```

Arguments

fram_db	Fram database object
old_table_name	Logical, defaults to TRUE. We intend to change the FRAM table from TAAETRSList to TAAETRSListChinook to avoid confusion. When working with a database where that hasn't been done, leave this argument to TRUE.
species	"COHO" or "CHINOOK". Optional, defaults to the database species. Provide this only if fram_db connects to a database with both Chinook and Coho information. And try to avoid that – those databases are sketchy to work with.

Value

tibble of TAAETRSList or TAAETRSListChinook tables translated to long form. `$taa_name` and `taa_num` identify the "TAA" group, `$stock_label` and `$stock_id` identify the FRAM stock, `$terminal_time_steps` and `$terminal_months` give the time periods that this stock is terminal, and `$fishery_label` and `$fishery_id` identify the fishery for which the stock is terminal.

See Also

`terminal_stocks()`, `terminal_fisheries()`

Examples

```
## Not run: fram_db |> parse_terminal_info()
```

`terminal_stocks` **[Experimental]** *List terminal stock information*

Description

For each TAA group, lists the associated FRAM stocks and timesteps. Intended to support working with bios for QAQC.

Usage

```
terminal_stocks(fram_db, species = NULL)
```

Arguments

`fram_db`	Fram database object
`species`	"COHO" or "CHINOOK". Optional, defaults to the database species. Provide this only if fram_db connects to a database with both Chinook and Coho information. And try to avoid that – those databases are sketchy to work with.

Value

Tibble of taa stocks and timesteps

Examples

```
## Not run: fram_db |> terminal_stocks()
```

`translate_nr_flag` *Provides English translation of numeric non-retention flags*

Description

Assumes the flags are for a Chinook run, as Coho only have one type of non-retention (dead fish).

Usage

```
translate_nr_flag(vec)
```

Arguments

`vec`	numeric vector of non-retention flags (possible values: 0 through 4)

Value

Character vector of same length as argument vec.

Examples

```
data <- data.frame(nr_flag = sample(1:4, size = 10, replace = TRUE))
data$translation = translate_nr_flag(data$nr_flag)
data
```

---

translate_scalers_flag

Provides English translation of numeric scalers flags

Description

Works for both Chinook and Coho (they use the same flagging for scalers).

Usage

```
translate_scalers_flag(vec)
```

Arguments

vec	vector of scaler flags (possible values: 1, 2, 7, 8, 17, 18, 27, 28).

Value

Character vector of same length as argument vec.

Examples

```
data <- data.frame(scalers_flag = sample(c(1, 2, 7, 8, 17, 18, 27, 28), 10, replace = TRUE))
data$translation = translate_scalers_flag(data$scalers_flag)
data
```

truns_fisheries *Returns a dataframe with fisheries defined by the TRuns report driver*

Description

Returns a dataframe with fisheries defined by the TRuns report driver

Usage

```
truns_fisheries(fram_db)
```

Arguments

fram_db	FRAM database object

Value

Tibble with fishery ID and TRUN stock name (stock_name).

See Also

truns_stocks()

Examples

```
## Not run: truns <- truns_fisheries(fram_db)
```

truns_stocks *Returns a dataframe with stocks defined by the TRuns report driver*

Description

Returns a dataframe with stocks defined by the TRuns report driver

Usage

```
truns_stocks(fram_db)
```

Arguments

fram_db	FRAM database object

Value

Tibble with stock ID and TRUN stock name (stocck_name).

See Also

truns_fisheries()

Examples

```
## Not run: truns <- truns_stocks(fram_db)
```

Index

∗ **Sensitivity**
  sensitivity_exact, 65
  sensitivity_scaled, 67
∗ **comparisons**
  compare_fishery_input_flags, 15
  compare_fishery_inputs, 15
  compare_inputs, 16
  compare_inputs_chart, 17
  compare_non_retention_inputs, 18
  compare_recruits, 19
  compare_runs, 20
  compare_sl_ratio, 21
  compare_stock_fishery_rate_scalers, 21
∗ **connections**
  connect_fram_db, 22
  disconnect_all_fram_connections, 27
  disconnect_fram_db, 28
  list_extant_fram_connections, 48
∗ **fishery_filters**
  filter_ak, 31
  filter_bc, 32
  filter_ca, 32
  filter_coast, 33
  filter_commercial_wa_nt, 34
  filter_marine, 36
  filter_net, 37
  filter_or, 38
  filter_puget_sound, 39
  filter_sport, 39
  filter_stt, 40
  filter_stt_nt, 41
  filter_wa, 42
∗ **labelers**
  label_fisheries_db, 45
  label_flags, 46
  label_stocks_db, 46
  label_timesteps_db, 47
∗ **sensitivity**
  sensitivity_custom, 64
∗ **soncc**
  create_soncc_pasteable, 26
∗ **stock_filters**
  filter_hatchery, 35
  filter_mixed, 36
  filter_wild, 42

add_total_mortality, 4
addstock_check, 4
aeq_mortality, 5
aeq_mortality(), *4*, *54*

bkfram_checks_coho, 6

calc_fram_scaling, 7
calc_fram_scaling(), *51*
calculate_soncc, *26*
calculate_stock_comp, 8
calculate_stock_comp(), *58*
change_run_id, 9
check_bp_coverage, 10
coho_mark_rates, 10
cohort_abundance, 11
compare_databases, 12
compare_fishery_input_flags, 15, *16, 17*, *19–22*
compare_fishery_inputs, 15, *15*, *17*, *19–22*
compare_inputs, *15*, 16, *16, 17*, *19–22*
compare_inputs_chart, *15, 16*, 17, *17*, *19–22*
compare_non_retention_input_flags, 18
compare_non_retention_inputs, *15–17*, 18, *19–22*
compare_recruits, *15–17*, 19, *19–22*
compare_runs, *15–17*, *19*, 20, *21, 22*
compare_runs(), *18*
compare_sl_ratio, *15–17*, *19, 20*, 21, *22*

compare_stock_fishery_rate_scalers, *15–17*, *19*, *20*, 21, *21*
connect_fram_db, 22, *27*, *28*, *48*
copy_fishery_scalers, 23
copy_run, 24
copy_run(), *26*, *49*
copy_tamm, 25
copy_tamm(), *25*, *49*
create_soncc_pasteable, 26

disconnect_all_fram_connections, *23*, 27, *28*, *48*
disconnect_fram_db, *23*, *27*, 28, *48*

fetch_quarto_templates, 28
fetch_quarto_templates(), *45*
fetch_table, 29
fetch_table(), *31*
fetch_table_*(*fetch_table*)*, 29
fetch_table_bkchin, 30
fetch_table_bkchin(), *30*
filter_ak, 31, *32–34*, *36*, *38–42*
filter_bc, *31*, 32, *33*, *34*, *36*, *38–42*
filter_ca, *31*, 32, *32*, *34*, *36*, *38–42*
filter_coast, *31*, *32*, 33, *33*, *34*, *36*, *38–42*
filter_commercial_wa_nt, *31–33*, 34, *34*, *36*, *38–42*
filter_hatchery, 35, *37*, *43*
filter_marine, *31–34*, 36, *38–42*
filter_mixed, *35*, 36, *43*
filter_net, *31–34*, *36*, 37, *38–42*
filter_or, *31–34*, *36*, 38, *38–42*
filter_puget_sound, *31–34*, *36*, *38*, 39, *40–42*
filter_sport, *31–34*, *36*, *38*, 39, *39*, *41*, *42*
filter_stt, *31–34*, *36*, *38*, *39*, 40, *40–42*
filter_stt_nt, *31–34*, *36*, *38–40*, 41, *41*, *42*
filter_wa, *31–34*, *36*, *38–41*, 42
filter_wild, *35*, *37*, 42
fishery_mortality, 43
format_soncc_pasteable, *26*
framrosetta::label_fisheries(), *45*
framrosetta::label_stocks(), *46*

initialize_project, 44
initialize_project(), *29*

label_fisheries_db, 45, *46*, *47*
label_flags, *45*, 46, *47*
label_stocks_db, *45*, 46, *46*, *47*
label_timesteps_db, *45*, *46*, 47, *47*
list_extant_fram_connections, *23*, *27*, *28*, 48

make_batch_runs, 48
make_batch_runs(), *24–26*
management_week, 49
modify_table, 50
modify_table(), *66*, *68*
mortality_scalers, 51
msf_encounters, 52
msf_encounters(), *53*, *54*
msf_landed_catch, 53
msf_landed_catch(), *52*, *54*
msf_mortalities, 53
msf_mortalities(), *52*, *53*
msp_mortality, 54

na_non_retention_from_flag, 55
na_scalers_from_flag, 55

plot_impacts_per_catch_heatmap, 56
plot_impacts_per_catch_heatmap(), *52*, *60*, *61*
plot_stock_comp, 58
plot_stock_comp(), *9*
plot_stock_mortality, 59
plot_stock_mortality(), *57*, *61*
plot_stock_mortality_time_step, 60
plot_stock_mortality_time_step(), *60*
population_statistics, 61
post_season_abundance, 62

remove_run, 63
run_info, 63

sensitivity_custom, 64
sensitivity_exact, 65, *68*
sensitivity_scaled, 67, *67*
statistical_week, 69
stock_fate, 70
stock_mortality, 71

terminal_fisheries, 71
terminal_fisheries(), *72*
terminal_info, 72
terminal_stocks, 73
terminal_stocks(), *72*
translate_nr_flag, 73

translate_scalers_flag, 74
truns_fisheries, 75
truns_fisheries(), *76*
truns_stocks, 75
truns_stocks(), *75*