

Package: framrsquared (via r-universe)

October 17, 2026

Type Package

Title FRAM Database Interface

Version 0.8.1.9000

Description 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, 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

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 2026-09-17 17:36:05 UTC

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

RemoteRef HEAD

RemoteSha 53f016112dcc4929af822588f5aa1d00b0c3cbc8

Contents

add_total_mortality	4
addstock_check	4
aeq_mortality	5
bkfram_checks_coho	6
calc_fram_scaling	6
change_run_id	8
check_demo_coverage	8
coho_mark_rates	9
cohort_abundance	10
compare_databases	10
compare_fishery_input_flags	13
compare_fishery_inputs	14
compare_inputs	14
compare_inputs_chart	15
compare_non_retention_input_flags	16
compare_non_retention_inputs	16
compare_recruits	17
compare_runs	18
compare_stock_fishery_rate_scalers	18
connect_fram_db	19
copy_fishery_scalers	20
copy_run	20
copy_tamms	21
disconnect_all_fram_connections	22
disconnect_fram_db	23
fetch_quarto_templates	24
fetch_table	25
fetch_table_bkchin	25
filter_ak	26
filter_bc	27
filter_ca	28
filter_coast	28
filter_commercial_wa_nt	29
filter_flag	30
filter_marine	30
filter_net	31
filter_or	32
filter_puget_sound	33
filter_sport	33
filter_wa	34
find_tables_by_column_	35
fishery_mortality	35
fram_clean_tables	36
fram_database_species	36
fram_database_type	37
get_fishery_ids	37

get_run_ids	38
get_stock_ids	38
initialize_project	39
input_summary_	40
label_fisheries_db	40
label_flags	41
label_stocks_db	41
list_extant_fram_connections	42
make_batch_runs	42
management_week	43
modify_table	44
mortality_scalers	45
msf_encounters	46
msf_encounters_chinook_	46
msf_encounters_coho_	47
msf_landed_catch	47
msf_landed_catch_chinook_	48
msf_landed_catch_coho_	48
msf_mortalities	49
msf_mortalities_chinook_	50
msf_mortalities_coho_	50
mzp_mortality	51
NR_flag_translate	52
plot_impacts_per_catch_heatmap	52
plot_stock_comp	53
plot_stock_mortality	54
plot_stock_mortality_time_step	55
population_statistics	56
post_season_abundance	56
provide_table_names	57
remove_run	57
run_info	58
scalers_flag_translate	58
sensitivity_custom	59
sensitivity_exact	60
sensitivity_scaled	62
statistical_week	64
stock_age_checker	65
stock_check_helper	65
stock_fate	66
stock_fate_chinook	67
stock_fate_coho	67
stock_id_comp	68
stock_mortality	69
styleguide	69
terminal_fisheries	70
terminal_info	71
terminal_stocks	71

truns_fisheries	72
truns_stocks	73
validate_fishery_ids	73
validate_fram_db	74
validate_run_id	74
validate_stock_ids	75
welcome	75

Index 76

add_total_mortality	<i>Sum separate mortality columns into new "total_mortality" column</i>
---------------------	---

Description

Convenience function for combining the separate mortality columns of the Mortality table. Note: this does *not* account for AEQ for Chinook.

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 just before \$landed_catch.

addstock_check	<i>Check FRAM database after adding new stock</i>
----------------	---

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.

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	<i>Extract AEQ mortality from Chinook FRAM database. Refactored and stripped down from the framr package written by Dan Auerbach. https://github.com/FRAMverse/framr/</i>
---------------	---

Description

Extract AEQ mortality from Chinook FRAM database. Refactored and stripped down from the framr package written by Dan Auerbach. <https://github.com/FRAMverse/framr/>

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	Logical, defaults to TRUE. Add human-readable columns for flags, fisheries, stocks?

Examples

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

## End(Not run)
```

bkfram_checks_coho	<i>Performs error checks of a backwards FRAM run Returns nested tibble with diagnostics</i>
--------------------	---

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] <i>Calculate match/replace df based on scaling</i>
-------------------	--

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 scalars 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 scalars.

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 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

```
## Not run:
## in run 31, decrease stock 1's recruit numbers by 50% and double 2's recruit numbers
library(here)
fram_db <- connect_fram_db(here("example_fram_db.mdb"))

df <- data.frame(match_RunID = c(31, 31), match_StockID = 1:2, scale_RecruitScaleFactor = c(.5, 2))

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)
```

change_run_id	<i>Changes a run's ID number in a FRAM database</i>
---------------	---

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

Examples

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

check_demo_coverage	<i>Check if exported functions are documented in a file</i>
---------------------	---

Description

Reads a package NAMESPACE file to extract exported functions and checks whether each function appears in a specified documentation file (e.g., a Quarto document or vignette). Reports coverage statistics to the console using the cli package.

Usage

```
check_demo_coverage(namespace_path, doc_path)
```

Arguments

namespace_path	Character string. Path to the NAMESPACE file, typically located at the root of an R package directory.
doc_path	Character string. Path to the documentation file to check (e.g., a .qmd, .Rmd, or .md file).

Details

The function searches for function calls in the form `function_name(` to determine if a function is present in the documentation. It prints a formatted report to the console showing which functions are documented and which are missing.

Value

Invisibly returns a list with three elements:

found Character vector of function names found in the document

missing Character vector of function names not found in the document

coverage Numeric percentage of functions documented

Examples

```
## Not run:
# Check if all exported functions appear in a vignette
check_demo_coverage("NAMESPACE", "vignettes/function_demo.qmd")

# Store results for further analysis
results <- check_demo_coverage("NAMESPACE", "README.md")
results$missing

## End(Not run)
```

coho_mark_rates	[Experimental] <i>Returns a tibble displaying predicted FRAMencounter mark rates by fishery, fishery type, and time-step.</i>
-----------------	--

Description

[Experimental] Returns a tibble displaying predicted FRAMencounter mark rates by fishery, fishery type, and time-step.

Usage

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

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)

Examples

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

## End(Not run)
```

cohort_abundance	[Experimental] <i>Calculate starting cohort abundance</i>
------------------	--

Description

The starting cohort abundance listed in the database can be wrong. This function calculates the value by multiplying 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)

Examples

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

## End(Not run)
```

compare_databases	[Experimental] <i>Compare tables in two equivalent FRAM databases</i>
-------------------	--

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 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. `$prop_err` provides this proportional error, where -0.5 means the entry in `fram_db_2` was 50% less than the corresponding value in `fram_db_1`, and a value of 2 means the entry in `fram_db_2` was 200% more than the corresponding value in `fram_db_1`. This gives error in context of the original value, and is often a good way to look for problems. However, we sometimes find very large `$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 0.00001 ($(0.002 - 0.001) / 1000$). 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	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_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	Two run ids
tolerance	Tolerance for detecting changes
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_fishery_inputs(c(55, 56))
```

compare_inputs

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

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

Details

Comparisons assume the first run provided is the baseline, and provide relative changes from that. This includes percent changes (\$percent.diff)include percent changes (infinite when)

Value

Data frame of differences. \$percentdiff = change in quota (comparing the appropriate quotas based on fishery flags), \$regulation_comparison = change in regulation (NS, MSF, NS + MSF). Columns present in the FisheriesScalers database are included, with _original and _comparison suffixes identifying entries associated with the first and second entries of run_ids, respectively.

See Also

[compare_inputs_chart\(\)](#)

Examples

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

compare_inputs_chart	<i>Generate heat map of changed values between two run inputs.</i>
----------------------	--

Description

Can be a very busy chart if not filtered down. Consider using a filter on the dataframe before piping into compare_input_chart.

Usage

```
compare_inputs_chart(.data)
```

Arguments

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

See Also

[compare_inputs\(\)](#)

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_inputs(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	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_inputs(c(55, 56))
```

compare_recruits	<i>Compares the recruit scalers of two runs</i>
------------------	---

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	Two run ids
tolerance	Tolerance for detecting changes
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_recruits()
```

compare_runs	<i>Generates a report to the console of changes to inputs between two runs</i>
--------------	--

Description

Generates a report to the console of changes to inputs between two runs

Usage

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

Arguments

fram_db	FRAM database object
run_ids	Two run ids. Run names must differ; change in FRAM if necessary.
tolerance	Tolerance of detection, 1 percent default

Examples

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

compare_stock_fishery_rate_scalers	<i>Compares the stock fishery rate scalers of two runs</i>
------------------------------------	--

Description

Compares the stock fishery rate scalers of two runs

Usage

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

Arguments

fram_db	FRAM database object
run_ids	Two run ids

See Also

[compare_runs\(\)](#)

Examples

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

connect_fram_db	<i>Connect to FRAM database</i>
-----------------	---------------------------------

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 <i>must not</i> 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_species` 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

[disconnect_fram_db\(\)](#), [disconnect_all_fram_connections\(\)](#)

Examples

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

copy_fishery_scalers	<i>Experimental copying scaler inputs from one run to another DANGEROUS</i>
----------------------	---

Description

Experimental copying scaler inputs from one run to another 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.

Examples

```
## Not run: framdb |> copy_fishery_scalers(132, 133, 87)
```

copy_run	[Experimental] <i>Copies a run a number of times</i>
----------	---

Description

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.

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.

Value

Invisibly returns the run ids of the copied runs, for use in other functions.

Examples

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

copy_tamms	[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. One TAMM file will be copied multiple times in the target_folder with suffixes that identify each of the run_ids. The "Use folder" button on the "Run Multiple Runs" screen can then use the target folder to set up large batch runs. Typically users should use make_batch_runs() instead (this first copies runs and then uses copy_tamms to create tamms that match the new runs).

Usage

```
copy_tamms(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.

Value

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

Examples

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

```
disconnect_all_fram_connections
```

Clear all connections

Description

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.

Usage

```
disconnect_all_fram_connections()
```

Value

nothing

Examples

```
## Not run:
fram_db = connect_fram_db("Chin2025.mdb")
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	<i>Safely disconnect from FRAM database</i>
--------------------	---

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

[connect_fram_db\(\)](#)

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

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

Value

Nothing.

See Also

[initialize_project\(\)](#)

fetch_table	<i>Fetch a complete table from a FRAM database.</i>
-------------	---

Description

Returns a cleaned tibble, with column labels that were camel case (e.g., TimeStep) converted to snake case (e.g., time_step). **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`.

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. If not given, a list of options will be printed.
label	Logical, defaults to TRUE. Add human-readable columns for flags, fisheries, stocks?
warn	Print a warning when fetching BackwardsFRAM table from a Chinook database? Logical, defaults to TRUE.

Examples

```
## Not run:
fram_db <- connect_fram_db("validat2024.mdb")
fram_db |> fetch_table('Mortality')
## End(Not run)
```

fetch_table_bkchin	<i>Safely fetch Chinook BackwardsFRAM table</i>
--------------------	---

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.

Examples

```
#' @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	<i>Filters a dataframe to Alaska fisheries. 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.</i>
-----------	---

Description

Filters a dataframe to Alaska fisheries. 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)
```

Arguments

```
.data          Dataframe containing fishery_id column. Commonly, output from framrsquared::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
```

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_ak()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_ak(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_ak(species = "COHO")
```

filter_bc	<i>Filters a dataframe to Canadian (BC) fisheries. 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.</i>
-----------	--

Description

Filters a dataframe to Canadian (BC) fisheries. 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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_bc()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_bc(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_bc(species = "COHO")
```

filter_ca	<i>Filters a dataframe to California fisheries. 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.</i>
-----------	---

Description

Filters a dataframe to California fisheries. 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)
```

Arguments

- .data Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_ca()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_ca(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_ca(species = "COHO")
```

filter_coast	<i>Filters a dataframe to Coastal fisheries. 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.</i>
--------------	--

Description

Filters a dataframe to Coastal fisheries. 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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_coast()

## End(Not run)
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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_commercial_wa_NT()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_commercial_wa_nt(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_commercial_wa_nt(species = "COHO")
```

filter_flag	<i>NA's all the information in the FisheryScalers that's not being used e.g Flag 1 only NS Scalers will be returned</i>
-------------	---

Description

NA's all the information in the FisheryScalers that's not being used e.g Flag 1 only NS Scalers will be returned

Usage

```
filter_flag(.data)
```

Arguments

.data	Fishery Scalers table
-------	-----------------------

Examples

```
## Not run: fishery_scalers_table |> filter_flag()
```

filter_marine	<i>Filters a dataframe to marine fisheries. 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.</i>
---------------	---

Description

Filters a dataframe to marine fisheries. 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)
```

Arguments

<code>.data</code>	Dataframe containing fishery_id column. Commonly, output from <code>framrsquared::fetch_table()</code> .
<code>species</code>	Optional argument to identify species if <code>.data</code> doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_marine()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_marine(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_marine(species = "COHO")
```

<code>filter_net</code>	<i>Filters a dataframe to net fisheries. 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.</i>
-------------------------	--

Description

Filters a dataframe to net fisheries. 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)
```

Arguments

<code>.data</code>	Dataframe containing fishery_id column. Commonly, output from <code>framrsquared::fetch_table()</code> .
<code>species</code>	Optional argument to identify species if <code>.data</code> doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_net()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_net(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_net(species = "COHO")
```

filter_or	<i>Filters a dataframe to Oregon fisheries. 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.</i>
-----------	---

Description

Filters a dataframe to Oregon fisheries. 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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_or()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_or(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_or(species = "COHO")
```

filter_puget_sound	<i>Filters a dataframe to Puget Sound fisheries. 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.</i>
--------------------	--

Description

Filters a dataframe to Puget Sound fisheries. 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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from <code>framrsquared::fetch_table()</code> .
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK" or variations thereof. Defaults to NULL

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_puget_sound()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_puget_sound(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_puget_sound(species = "COHO")
```

filter_sport	<i>Filters a dataframe to sport fisheries. 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.</i>
--------------	--

Description

Filters a dataframe to sport fisheries. 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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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_wa\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_sport()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_sport(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_sport(species = "COHO")
```

filter_wa	<i>Filters a dataframe to Washington State fisheries. 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.</i>
-----------	---

Description

Filters a dataframe to Washington State fisheries. 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)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::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

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\(\)](#)

Examples

```
## Not run:
fram_dataframe |> filter_wa()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_wa(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_wa(species = "COHO")
```

find_tables_by_column_

Finds tables that contain a specific column name

Description

Finds tables that contain a specific column name

Usage

```
find_tables_by_column_(fram_db, column_name)
```

Arguments

fram_db	FRAM database object
column_name	Name of a column

Examples

```
## Not run: fram_db |> find_tables_by_column_('RunID')
```

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, msp = TRUE)
```

Arguments

fram_db	FRAM database object
run_id	Run ID
msp	Use Model Stock Proportion? Logical, defaults to TRUE.

Examples

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

## End(Not run)
```

fram_clean_tables	<i>Cleans the names of FRAM tables and coverts to a tibble</i>
-------------------	--

Description

Cleans the names of FRAM tables and coverts to a tibble

Usage

```
fram_clean_tables(.data)
```

Arguments

.data	Dataframe
-------	-----------

Examples

```
## Not run: fram_dataframe |> fram_clean_tables()
```

fram_database_species	<i>Identifies the FRAM database species focus - Chinook or Coho</i>
-----------------------	---

Description

Identifies the FRAM database species focus - Chinook or Coho

Usage

```
fram_database_species(con)
```

Arguments

con	Connection to FRAM database
-----	-----------------------------

Examples

```
## Not run: fram_database_species(con)
```

fram_database_type	<i>Identifies the FRAM database type - Full or Transfer</i>
--------------------	---

Description

Identifies the FRAM database type - Full or Transfer

Usage

```
fram_database_type(con)
```

Arguments

con	Connection to FRAM database
-----	-----------------------------

Examples

```
## Not run: fram_database_type(con)
```

get_fishery_ids	<i>Gets all fishery_ids of FRAM database</i>
-----------------	--

Description

Gets all fishery_ids of FRAM database

Usage

```
get_fishery_ids(fram_db)
```

Arguments

fram_db	Fram database object
---------	----------------------

Examples

```
## Not run: fram_dataframe |> get_run_ids()
```

`get_run_ids`*Gets all run_ids of FRAM database*

Description

Gets all run_ids of FRAM database

Usage

```
get_run_ids(fram_db)
```

Arguments

`fram_db` Fram database object

Examples

```
## Not run: fram_dataframe |> get_run_ids()
```

`get_stock_ids`*Gets all stock_id of FRAM database*

Description

Gets all stock_id of FRAM database

Usage

```
get_stock_ids(fram_db)
```

Arguments

`fram_db` Fram database object

Examples

```
## Not run: fram_dataframe |> get_run_ids()
```

initialize_project	[Experimental] <i>Initializes a FRAM project</i>
--------------------	---

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 <code>_quarto.yml</code> and <code>style.css</code> 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 sub-folder 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()`.

See Also

[fetch_quarto_templates\(\)](#)

Examples

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

## End(Not run)
```

input_summary_	<i>Generates an input summary based on a FisheryScalers dataframe. Probably end up streamlining / revising this.</i>
----------------	--

Description

Generates an input summary based on a FisheryScalers dataframe. Probably end up streamlining / revising this.

Usage

```
input_summary_(.data, run_id)
```

Arguments

.data	FisheryFishery scalers dataframe
run_id	Run ID number

Examples

```
## Not run: fishery_scalers_dataframe |> input_summary()
```

label_fisheries_db	<i>Label fisheries based on FRAM database</i>
--------------------	---

Description

Like [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()`, useful if working with databases with unusual stock or fishery tables.

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

[label_fisheries\(\)](#), [label_stocks_db\(\)](#)

label_flags	<i>Adds a column with a text version of flags for either non-retention or fishery scalers</i>
-------------	---

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 | fetches FisheryScalers or NonRetentions |
| species | Optional, identifying species if .data doesn't. If provided, should be "CHI-NOOK" or "COHO" (or variants) |
| warn | Logical, defaults to TRUE. Warn if neither flag column is present in dataframe? |

Examples

```
## Not run: mortality_table |> add_flag_text()
```

label_stocks_db	<i>Label stocks based on FRAM database</i>
-----------------	--

Description

Like [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()`, useful if working with databases with unusual stock or fishery tables.

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

[label_stocks\(\)](#), [label_fisheries_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()
```

`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

[sensitivity_exact\(\)](#), [sensitivity_scaled\(\)](#), [sensitivity_custom\(\)](#)

management_week	<i>Vectorized approach to calculating the management week, returns an integer</i>
-----------------	---

Description

Vectorized approach to calculating the management week, returns an integer

Usage

```
management_week(date)
```

Arguments

date	A column with dates
------	---------------------

Examples

```
## Not run:
data_fram |>
  mutate(mngmt_week = management_week(date_field))

## End(Not run)
```

modify_table	[Experimental] <i>Modify FRAM database based on match/replace dataframe</i>
--------------	--

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 would probably start by using `copy_run` to create multiple duplicate runs, and then 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))`. Notably, we might 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`.

Examples

```
## Not run:
df_total <- tibble(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
fram_db |> modify_db(table_name = "StockRecruit", df = df_total)

## End(Not run)
```

mortality_scalers	<i>Quantify the proportion of fishery mortalities associated with stock(s) of interest</i>
-------------------	--

Description

Supports guestimating the impact of making changes to a fishery on a particular stock (or group of stocks) by multiplying its mortalities by the stock_mortality_ratio produced by mortality_scalers().

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	Do we use MSP expansion? Logical, defaults to FALSE. Only relevant for Chinook

See Also

```
plot\_impacts\_per\_catch\_heatmap\(\)
```

Examples

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

msf_encounters	<i>Reproduce MSF encounters screen</i>
----------------	--

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_encounters_chinook_	<i>Replicate MSF screen report encounters for Chinook</i>
-------------------------	---

Description

Returns a tibble matching the MSF screen report encounters for Chinook. This is specific for Chinook and in most cases [msf_encounters\(\)](#) is preferable.

Usage

```
msf_encounters_chinook_(fram_db)
```

Arguments

fram_db	FRAM database object
---------	----------------------

See Also

[msf_encounters\(\)](#), [msf_mortalities_chinook_\(\)](#), [msf_landed_catch_chinook_\(\)](#)

Examples

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

msf_encounters_coho_	<i>Replicate MSF screen report encounters for COHO</i>
----------------------	--

Description

Returns a tibble matching the MSF screen report encounters for Coho This is specific for Coho and in most cases `msf_encounters()` is preferable.

Usage

```
msf_encounters_coho_(fram_db)
```

Arguments

fram_db	FRAM database object
---------	----------------------

See Also

[msf_encounters\(\)](#), [msf_landed_catch_coho_\(\)](#), [msf_mortalities_coho_\(\)](#)

Examples

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

msf_landed_catch	<i>Reproduce MSF landed catch screen</i>
------------------	--

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_landed_catch_chinook_

Replicate MSF screen report landed catch for Chinook

Description

Returns a tibble matching the MSF screen report landed catch for Chinook. This is specific for Chinook and in most cases [msf_landed_catch\(\)](#) is preferable.

Usage

```
msf_landed_catch_chinook_(fram_db)
```

Arguments

fram_db FRAM database object

See Also

[msf_landed_catch\(\)](#), [msf_mortalities_chinook_\(\)](#), [msf_encounters_chinook_\(\)](#)

Examples

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

msf_landed_catch_coho_

Replicate MSF screen report landed catch for COHO

Description

Returns a tibble matching the MSF screen report landed catch for Coho This is specific for Coho and in most cases [msf_landed_catch\(\)](#) is preferable.

Usage

```
msf_landed_catch_coho_(fram_db)
```

Arguments

fram_db FRAM database object

See Also

[msf_landed_catch\(\)](#), [msf_encounters_coho\(\)](#), [msf_mortalities_coho\(\)](#)

Examples

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

msf_mortalities	<i>Reproduce MSF mortalities screen</i>
-----------------	---

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

See Also

[msf_encounters\(\)](#), [msf_landed_catch\(\)](#)

Examples

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

`msf_mortalities_chinook_`*Replicate MSF screen report mortalities for Chinook*

Description

Returns a tibble matching the MSF screen report mortalities for Chinook. This is specific for Chinook and in most cases `msf_mortalities()` is preferable.

Usage

```
msf_mortalities_chinook_(fram_db)
```

Arguments

<code>fram_db</code>	FRAM database object
----------------------	----------------------

See Also

`msf_mortalities()`, `msf_encounters_chinook_()`, `msf_landed_catch_chinook_()`

Examples

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

`msf_mortalities_coho_` *Replicate MSF screen report mortalities for COHO*

Description

Returns a tibble matching the MSF screen report mortalities for Coho. This is specific for Coho and in most cases `msf_mortalities()` is preferable.

Usage

```
msf_mortalities_coho_(fram_db)
```

Arguments

<code>fram_db</code>	FRAM database object
----------------------	----------------------

See Also

`msf_mortalities()`, `msf_encounters_coho_()`, `msf_landed_catch_coho_()`

Examples

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

msp_mortality	<i>Expand Chinook mortality table using Model-Stock Proportion</i>
---------------	--

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)
```

NR_flag_translate	<i>Provides English translation of numeric non-retention flags</i>
-------------------	--

Description

Provides English translation of numeric non-retention flags

Usage

```
NR_flag_translate(vec)
```

Arguments

vec	vector of flags
-----	-----------------

Examples

```
## Not run: NR_flag_translate(sample(1:4, 10, replace = T))
```

plot_impacts_per_catch_heatmap	<i>Make plots to show the amount of landed catch_per_impact</i>
--------------------------------	---

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.

Usage

```
plot_impacts_per_catch_heatmap(
  fram_db,
  run_id,
  stock_id,
  filters_list = list(filter_wa, filter_sport),
  msp = TRUE
)
```

Arguments

fram_db	fram database connection
run_id	run_id of interest
stock_id	stock_id of interest
filters_list	list of framrsquared filter functions to apply before plotting. Defaults to list(filter_wa, filter_sport), which filters to WA sport fisheries.
msp	Use Model Stock Proportion? Logical, defaults to TRUE.

Value

ggplot 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	<i>Plot stock composition</i>
-----------------	-------------------------------

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

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
)
```

Arguments

fram_db	fram database object, supplied through connect_fram_db
run_id	numeric, RunID
stock_id	numeric, ID of focal 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.

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

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

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
)
```

Arguments

fram_db	fram database object, supplied through connect_fram_db
run_id	numeric, RunID
stock_id	numeric, ID of focal 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.

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

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. Used for forecasting.*

Description

Generates post-season January age 3 abundances by stock from post-season databases. Used for forecasting.

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.

Examples

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

provide_table_names	<i>List names of FRAM table</i>
---------------------	---------------------------------

Description

Provides list of FRAM database names, typically useful for internal functions.

Usage

```
provide_table_names(is_full = TRUE)
```

Arguments

is_full	Logical. Provide names for a full FRAM database (TRUE) or a model transfer (FALSE)?
---------	---

Value

Character string of the names of FRAM tables

Examples

```
provide_table_names(is_full = FALSE)
```

remove_run	<i>Removes a run in a FRAM database</i>
------------	---

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

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

<code>fram_db</code>	FRAM database object
<code>run_id</code>	FRAM run ID

Examples

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

`scalars_flag_translate`*Provides English translation of numeric scalars flags*

Description

Provides English translation of numeric scalars flags

Usage

```
scalars_flag_translate(vec)
```

Arguments

<code>vec</code>	vector of flags
------------------	-----------------

Examples

```
## Not run: scalars_flag_translate(sample(c(1, 2, 7, 8, 17, 18, 27, 28), 10, replace = T))
```

sensitivity_custom	[Experimental] <i>Generate sensitivity analyses runs based on a list of match/replace dataframes</i>
--------------------	---

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.

See Also

[sensitivity_scaled\(\)](#), [sensitivity_exact\(\)](#)

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] <i>Generate sensitivity analyses runs based on exact values</i>
-------------------	---

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 <code>data.frame(StockID = c(17, 18))</code> . To modify values only for Stillaguamish age 2s, we would use <code>expand_grid(StockID = c(17, 19), Age = 2)</code> . Unlike match/replace dataframes for <code>modify_table()</code> , 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 <code>table_name</code> that should be rescaled.
tamm_template	Optional; character string of filepath of a TAMM to be used as a template. If provided (and <code>tamm_target_folder</code> provided), <code>sensitivity_scaled</code> 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.

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] <i>Generate sensitivity analyses runs scaled by template values</i>
--------------------	---

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 specified 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 <code>data.frame(StockID = c(17, 18))</code> . To modify values only for Stillaguamish age 2s, we would use <code>expand_grid(StockID = c(17, 19), Age = 2)</code> . Unlike match/replace dataframes for <code>modify_table()</code> , 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 used, one per sensitivity analysis run. Defines the number of runs generated. For example, <code>scale_values = 2:10</code> 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 <code>table_name</code> that should be rescaled.
tamm_template	Optional; character string of filepath of a TAMM to be used as a template. If provided (and <code>tamm_target_folder</code> provided), <code>sensitivity_scaled</code> 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

[sensitivity_exact\(\)](#), [sensitivity_custom\(\)](#)

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	<i>Vectorized approach to calculating the statistical week, returns an integer</i>
------------------	--

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
------	-------------------

Examples

```
## Not run:
data_fram |>
  mutate(mngmt_week = statistical_week(date_field))

## End(Not run)
```

stock_age_checker	<i>Helper function to check that all stock x age combinations are present</i>
-------------------	---

Description

#' Intended for internal use, makes some assumptions about inputs.

Usage

```
stock_age_checker(table_name, NumStk, old_stockcount, df, min_age, max_age)
```

Arguments

table_name	Character of table name, for informative messages.
NumStk	Maximum number of stock, pulled from BaseID table
old_stockcount	Number of stock in previous FRAM baseperiod. Only looks for problems for StockID > this number.
df	Dataframe to check. Must have columns "stock_id" and "age" (which are the names for relevant columns of framrsquared::fetch_table).
min_age	Minimum age modeled. Should be the min_age from the baseid_df.
max_age	Maximum age modeled. Should be the max_age from the baseid_df.

Value

numeri; 0 if no warning, 1 if warning.

stock_check_helper	<i>Helper function to check that stock id make sense</i>
--------------------	--

Description

More thorough checking than stock_id_comp. Checks that the number of stock IDs makes sense given NumStk, that Stock IDs are sequential (in the sense that if NumStk = n, every integer up to n is represented). Optionally, can check that each stock ID is unique.

Usage

```
stock_check_helper(table_name, NumStk, stock_vec, uniques_only = FALSE)
```

Arguments

table_name	Character of table name, for informative messages.
NumStk	Maximum number of stock, pulled from BaseID table
stock_vec	vector of stock ids to check. Presumably column of fetched table.
uniques_only	Do we want warnings if there are duplicats of StockIDs? Useful for tables like Stock and Growth that should have only one entry per stock. Logical, default = FALSE.

Value

Numeric, returning number of warnings detected.

See Also

[addstock_check\(\)](#)

stock_fate	[Experimental] Summarizes the three true outcomes of a stocks abundance, where it dies to fishery related mortality, natural mortality, or reaches some sort of escapement. When run against the coho database spawning escapement will be displayed, when run against the Chinook database escapement to the river will be displayed along with recruits to the next year 'age_up'
------------	--

Description

[Experimental] Summarizes the three true outcomes of a stocks abundance, where it dies to fishery related mortality, natural mortality, or reaches some sort of escapement. When run against the coho database spawning escapement will be displayed, when run against the Chinook database escapement to the river will be displayed along with recruits to the next year 'age_up'

Usage

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

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)
units	'fish' or 'percentage'. Percentage is proportion of starting adundance

Examples

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

## End(Not run)
```

stock_fate_chinook	[Experimental] <i>Chinook flavor of the stock fate function</i>
--------------------	--

Description

[Experimental] Chinook flavor of the stock fate function

Usage

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

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)
units	'fish' or 'percentage'. Percentage is proportion of starting abundance

See Also

[stock_fate\(\)](#)

Examples

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

stock_fate_coho	[Experimental] <i>Coho flavor of the stock fate function</i>
-----------------	---

Description

[Experimental] Coho flavor of the stock fate function

Usage

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

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)
units	'fish' or 'percentage'. Percentage is proportion of starting abundance

See Also

[stock_fate\(\)](#)

Examples

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

## End(Not run)
```

stock_id_comp	<i>Helper function to check that stock id exist in the Stock database</i>
---------------	---

Description

Intended for internal use, makes some assumptions about inputs.

Usage

```
stock_id_comp(table_name, df, stock_ref)
```

Arguments

- table_name Character of table name, for informative messages
- df Dataframe
- stock_ref numeric vector of all stock IDs. Should be stock_df\$stock_id.

Value

numeric; 0 if no warning, 1 if warning.

See Also

[addstock_check\(\)](#)

stock_mortality	<i>Replicate Stock Mortality screen</i>
-----------------	---

Description

Returns a tibble matching the Stock Mortality screen.

Usage

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

Arguments

fram_db	FRAM database object
run_id	Run ID

Examples

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

styleguide	<i>Framrsquared style guide</i>
------------	---------------------------------

Description

For easy readability, we want to use consistent coding style when developing code for framrsquared. Presently this includes (a) using <- for assignment, and (b) using snakecase for variable and function names. The functions here streamline checking R code for consistency with this style.

Usage

```
frs_stylecheck_assignment(filepath, n = Inf)
```

```
frs_stylecheck_snakecase(filepath, n = Inf)
```

Arguments

filepath	Path to R file to be checked
n	Number of rows to print. Default is to print all rows, but set to smaller values if output is overwhelming.

Details

`frs_stylecheck_assignment()` takes the path to an R file, and prints (and returns) any rows that may be mis-using the = for assignment. Note that it will give false positives for arguments defined in function calls if the call spans multiple lines, as well as = signs included in character strings.

`frs_stylecheck_snakecase` takes the path to an R file and prints (and returns) the names of any variables assigned using `<-` that do not include underscores. This will identify variables that do not use snakecase, but will also give false positive matches for variables that are single words and thus do not need snakecase.

Examples

```
## Not run:
frs_stylecheck_assignment("R/copy.R")
frs_stylecheck_snakecase("R/copy.R")

## End(Not run)
```

terminal_fisheries	[Experimental] <i>List terminal stock information</i>
--------------------	--

Description

For each TAA, lists the associated fisheries

Usage

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

Arguments

<code>fram_db</code>	Fram database object
<code>species</code>	"COHO" or "CHINOOK". Optional, defaults to the database species. Provide this only if <code>fram_db</code> 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] <i>Parse TAAETRS table</i>
---------------	--

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.

See Also

`terminal_stocks()`, `terminal_fisheries()`

Examples

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

terminal_stocks	[Experimental] <i>List terminal stock information</i>
-----------------	--

Description

For each TAA, lists the associated FRAM stocks and timesteps.

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

taa of taa stocks and timesteps

Examples

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

truns_fisheries	<i>Returns a dataframe with fisheries defined by the TRuns report driver</i>
-----------------	--

Description

Returns a dataframe with fisheries defined by the TRuns report driver

Usage

```
truns_fisheries(fram_db)
```

Arguments

fram_db	FRAM database object
---------	----------------------

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
---------	----------------------

See Also

[truns_fisheries\(\)](#)

Examples

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

validate_fishery_ids	Convenience function to check fishery input
----------------------	---

Description

No error checking for transfer databases

Usage

```
validate_fishery_ids(fram_db, fishery_id, call = rlang::caller_env())
```

Arguments

fram_db	FRAM database object
fishery_id	one or more fishery_ids
call	internal use: identify name of function that called this function (for informative error message)

validate_fram_db	<i>Convenience function to check fram_db input</i>
------------------	--

Description

Convenience function to check fram_db input

Usage

```
validate_fram_db(  
  fram_db,  
  db_type = NULL,  
  db_species = NULL,  
  call = rlang::caller_env()  
)
```

Arguments

fram_db	FRAM database object
db_type	Enforcement of a database type 'full' or 'transfer'
db_species	Enforcement of a species 'COHO' or 'CHINOOK'
call	internal use: identify name of function that called this function (for informative error message)

validate_run_id	<i>Convenience function to check run_id input</i>
-----------------	---

Description

Convenience function to check run_id input

Usage

```
validate_run_id(fram_db, run_id, call = rlang::caller_env())
```

Arguments

fram_db	FRAM database object
run_id	one or more run_ids
call	internal use: identify name of function that called this function (for informative error message)

validate_stock_ids	<i>Convenience function to check fishery input</i>
--------------------	--

Description

No error checking for transfer databases

Usage

```
validate_stock_ids(fram_db, stock_id, call = rlang::caller_env())
```

Arguments

fram_db	FRAM database object
stock_id	one or more stock_ids
call	internal use: identify name of function that called this function (for informative error message)

welcome	<i>Welcome message, summarizing database information</i>
---------	--

Description

Welcome message, summarizing database information

Usage

```
welcome(con)
```

Arguments

con	FRAM database connection
-----	--------------------------

Examples

```
## Not run: welcome(con)
```

Index

- * **fishery_filters**
 - filter_ak, 26
 - filter_bc, 27
 - filter_ca, 28
 - filter_coast, 28
 - filter_commercial_wa_nt, 29
 - filter_marine, 30
 - filter_net, 31
 - filter_or, 32
 - filter_puget_sound, 33
 - filter_sport, 33
 - filter_wa, 34
- add_total_mortality, 4
- addstock_check, 4
- addstock_check(), 66, 68
- aeq_mortality, 5
- aeq_mortality(), 4, 51
- bkfram_checks_coho, 6
- calc_fram_scaling, 6
- change_run_id, 8
- check_demo_coverage, 8
- coho_mark_rates, 9
- cohort_abundance, 10
- compare_databases, 10
- compare_fishery_input_flags, 13
- compare_fishery_inputs, 14
- compare_inputs, 14
- compare_inputs(), 15
- compare_inputs_chart, 15
- compare_inputs_chart(), 15
- compare_non_retention_input_flags, 16
- compare_non_retention_inputs, 16
- compare_recruits, 17
- compare_runs, 18
- compare_runs(), 13, 14, 16–18
- compare_stock_fishery_rate_scalers, 18
- connect_fram_db, 19
- connect_fram_db(), 23
- copy_fishery_scalers, 20
- copy_run, 20
- copy_tamms, 21
- disconnect_all_fram_connections, 22
- disconnect_all_fram_connections(), 19
- disconnect_fram_db, 23
- disconnect_fram_db(), 19
- fetch_quarto_templates, 24
- fetch_quarto_templates(), 39
- fetch_table, 25
- fetch_table_(fetch_table), 25
- fetch_table_bkchin, 25
- fetch_table_bkchin(), 25
- filter_ak, 26, 27–29, 31–35
- filter_bc, 27, 27–29, 31–35
- filter_ca, 27, 28, 29, 31–35
- filter_coast, 27, 28, 28, 29, 31–35
- filter_commercial_wa_nt, 27, 28, 29, 29, 31–35
- filter_flag, 30
- filter_marine, 27–29, 30, 31–35
- filter_net, 27–29, 31, 31–35
- filter_or, 27–29, 31, 32, 33–35
- filter_puget_sound, 27–29, 31, 32, 33, 34, 35
- filter_sport, 27–29, 31, 32, 33, 33, 35
- filter_wa, 27–29, 31–33, 34, 34
- find_tables_by_column_, 35
- fishery_mortality, 35
- fram_clean_tables, 36
- fram_database_species, 36
- fram_database_type, 37
- frs_stylecheck_assignment (styleguide), 69
- frs_stylecheck_snakecase (styleguide), 69

get_fishery_ids, 37
get_run_ids, 38
get_stock_ids, 38

initialize_project, 39
initialize_project(), 24
input_summary_, 40

label_fisheries(), 40, 41
label_fisheries_db, 40
label_fisheries_db(), 42
label_flags, 41
label_stocks(), 41, 42
label_stocks_db, 41
label_stocks_db(), 41
list_extant_fram_connections, 42

make_batch_runs, 42
management_week, 43
modify_table, 44
modify_table(), 61, 63
mortality_scalers, 45
msf_encounters, 46
msf_encounters(), 46–49
msf_encounters_chinook_, 46
msf_encounters_chinook_(), 48, 50
msf_encounters_coho_, 47
msf_encounters_coho_(), 49, 50
msf_landed_catch, 47
msf_landed_catch(), 46, 48, 49
msf_landed_catch_chinook_, 48
msf_landed_catch_chinook_(), 46, 50
msf_landed_catch_coho_, 48
msf_landed_catch_coho_(), 47, 50
msf_mortalities, 49
msf_mortalities(), 46, 48, 50
msf_mortalities_chinook_, 50
msf_mortalities_chinook_(), 46, 48
msf_mortalities_coho_, 50
msf_mortalities_coho_(), 47, 49
msp_mortality, 51

NR_flag_translate, 52

plot_impacts_per_catch_heatmap, 52
plot_impacts_per_catch_heatmap(), 45, 54, 55
plot_stock_comp, 53
plot_stock_mortality, 54
plot_stock_mortality(), 53, 55
plot_stock_mortality_time_step, 55
plot_stock_mortality_time_step(), 54
population_statistics, 56
post_season_abundance, 56
provide_table_names, 57

remove_run, 57
run_info, 58

scalers_flag_translate, 58
sensitivity_custom, 59
sensitivity_custom(), 43, 63
sensitivity_exact, 60
sensitivity_exact(), 43, 60, 63
sensitivity_scaled, 62
sensitivity_scaled(), 43, 60
statistical_week, 64
stock_age_checker, 65
stock_check_helper, 65
stock_fate, 66
stock_fate(), 67, 68
stock_fate_chinook, 67
stock_fate_coho, 67
stock_id_comp, 68
stock_mortality, 69
styleguide, 69

terminal_fisheries, 70
terminal_fisheries(), 71
terminal_info, 71
terminal_stocks, 71
terminal_stocks(), 71
truns_fisheries, 72
truns_fisheries(), 73
truns_stocks, 73
truns_stocks(), 72

validate_fishery_ids, 73
validate_fram_db, 74
validate_run_id, 74
validate_stock_ids, 75

welcome, 75