# fidelioDiagnostic

## config

### default.yml

empty

### project.yml

paths:

gdx\_dir: "data-raw/gdx/eta\_cpi\_025\_beta\_infl\_1"

outputs: "outputs"

scenarios: ["baseline","ff55"]

groups:

EU28: ["AUT","BEL","BGR","HRV","CYP","CZE","DNK","EST","FIN","FRA","DEU","GRC",

"HUN","IRL","ITA","LVA","LTU","LUX","MLT","NLD","POL","PRT","ROU","SVK",

"SVN","ESP","SWE","GBR"]

extract:

include: ["GDPr\_t","HSAVR\_t","HDY\_VAL\_t","GSUR\_VAL\_t","GINV\_VAL\_t",

"Q\_t","I\_PP\_t","P\_I\_t","K\_t","L\_t","U\_t","P\_CPI\_t","ir\_t",

"P\_INPUT\_t","P\_Q\_t","USE\_PP\_t","M\_TOT\_t","P\_USE\_t","P\_Mcif\_t",

"BITRADE\_t","GHG\_t"]

aggregations:

additive\_symbols: ["GDPr\_t","HDY\_VAL\_t","GSUR\_VAL\_t","GINV\_VAL\_t",

"Q\_t","I\_PP\_t","K\_t","L\_t","U\_t",

"USE\_PP\_t","M\_TOT\_t","BITRADE\_t","GHG\_t",

"P\_USE\_t","P\_Mcif\_t","P\_I\_t","P\_Q\_t","P\_INPUT\_t"]

save:

formats: ["parquet","feather"] # add "csv","fst","rds" if you like

# ---- selective bundles belong UNDER save: ----

bundles:

diagnostic\_app:

include: ["GDPr\_t","HDY\_VAL\_t","HDYr\_t","I\_TOT\_PP\_t","DS\_t","FS\_t","GSUR\_VAL\_t",

"GINV\_VAL\_t","TBr\_t","TB\_GDP\_t","HSAVR\_t","U\_t","KLratio\_country\_t",

"ir\_t","P\_HH\_CPI\_t","I\_PP\_t","K\_t","L\_t","GHG\_t","KLratio\_t","P\_Q\_t",

"P\_KL\_t","I\_PP\_SECT6\_t","OUT\_COMP6\_SHARE\_REAL\_t","BITRADE\_REG\_t"]

results\_app:

include: ["GDPr\_t","TBr\_t","TB\_GDP\_t","I\_PP\_SECT6\_t","OUT\_COMP6\_SHARE\_REAL\_t","BITRADE\_REG\_t"]

filters:

BITRADE\_REG\_t:

keep:

c: ["TOT"] # keep only commodity total

csv\_combine: true

csv\_basename: "results\_bundle"

csv\_shape: "wide"

derive:

# list base symbols to promote to derived if not present

include\_from\_base: ["GDPr\_t"]

# optional: tell the promoter which keys to use when coercing long->wide

# keys:

# GDPr\_t: ["n","t"]

## Inst

### App/diagnostic/app.R

library(shiny)

library(data.table)

library(ggplot2)

library(DT)

# Bind internal helpers from the package namespace

load\_symbol <- fidelioDiagnostics:::load\_symbol

load\_bundle <- fidelioDiagnostics:::load\_bundle

load\_manifest <- fidelioDiagnostics:::load\_manifest

resolve\_outputs\_dir <- fidelioDiagnostics:::resolve\_outputs\_dir

outputs\_dir\_from\_config <- fidelioDiagnostics:::outputs\_dir\_from\_config

cat("[APP] getwd(): ", getwd(), "\n")

cat("[APP] outputs/derived resolved to: ",

normalizePath(file.path("outputs","derived"), winslash="/", mustWork = FALSE), "\n")

# ---- Helpers to read base and promote to derived-shape ----

read\_base\_symbol <- function(name, dir\_base = file.path("outputs","base")) {

f\_parq <- file.path(dir\_base, paste0(name, ".parquet"))

f\_feat <- file.path(dir\_base, paste0(name, ".feather"))

if (file.exists(f\_parq)) {

if (!requireNamespace("arrow", quietly = TRUE)) return(NULL)

return(data.table::as.data.table(arrow::read\_parquet(f\_parq)))

}

if (file.exists(f\_feat)) {

if (requireNamespace("arrow", quietly = TRUE)) {

return(data.table::as.data.table(arrow::read\_feather(f\_feat)))

} else if (requireNamespace("feather", quietly = TRUE)) {

return(data.table::as.data.table(feather::read\_feather(f\_feat)))

} else return(NULL)

}

NULL

}

promote\_base\_to\_derived <- function(tbl, key\_cols) {

need <- c(key\_cols, "scenario", "value")

if (!all(need %in% names(tbl))) return(NULL)

W <- data.table::dcast(tbl, as.formula(paste(paste(key\_cols, collapse = " + "), "~ scenario")),

value.var = "value")

if (!all(c("baseline","ff55") %in% names(W))) return(NULL)

W <- W[!is.na(baseline) & !is.na(ff55)]

W[, delta := ff55 - baseline]

W[, pct := data.table::fifelse(abs(baseline) > .Machine$double.eps, delta / baseline, NA\_real\_)]

W[]

}

# ---- Load bundle first, then fallback to manifest ----

get\_results <- function() {

dir <- file.path("outputs","derived")

b <- try(load\_bundle("diagnostic\_app", dir = dir), silent = TRUE)

if (!inherits(b, "try-error") && !is.null(b)) return(b)

# Fallback: read per symbol via manifest, using the list in YAML/docs

wanted <- c("GDPr\_t","HDY\_VAL\_t","HDYr\_t","I\_TOT\_PP\_t","DS\_t","FS\_t","GSUR\_VAL\_t","GINV\_VAL\_t",

"TBr\_t","TB\_GDP\_t","HSAVR\_t","U\_t","KLratio\_country\_t","ir\_t","P\_HH\_CPI\_t",

"I\_PP\_t","K\_t","L\_t","GHG\_t","KLratio\_t","P\_Q\_t","P\_KL\_t",

"I\_PP\_SECT6\_t","OUT\_COMP6\_SHARE\_REAL\_t","BITRADE\_REG\_t")

setNames(lapply(wanted, function(s) load\_symbol(s, dir = dir)), wanted)

}

results\_by\_symbol <- get\_results()

# Keep only successfully loaded (non-NULL) tables

results\_by\_symbol <- Filter(function(x) !is.null(x) && nrow(x) > 0, results\_by\_symbol)

available\_syms <- names(results\_by\_symbol)

# ---- Your existing meta (trimmed to what exists) ----

meta <- data.table::rbindlist(list(

# ---------- Nation level (n,t)

data.table(symbol = c("GDPr\_t","HDY\_VAL\_t","HDYr\_t","I\_TOT\_PP\_t","DS\_t","FS\_t","GSUR\_VAL\_t",

"GINV\_VAL\_t","TBr\_t","TB\_GDP\_t","HSAVR\_t","U\_t","KLratio\_country\_t","ir\_t","P\_HH\_CPI\_t"),

label = c("Real GDP","Household disposable income","Household real disposable income","Total investment",

"Domestic savings","Foreign savings","Government surplus","Government investment",

"Trade balance (real)","Trade balance to GDP ratio",

"Household saving rate","Unemployment","Capital–labor ratio (country)","Interest rate","HH CPI"),

group = "Nation level", keep = TRUE),

# ---------- Industry level (n,i,t)

data.table(symbol = c("I\_PP\_t","K\_t","L\_t","GHG\_t","KLratio\_t","P\_Q\_t","P\_KL\_t",

"I\_PP\_SECT6\_t","OUT\_COMP6\_SHARE\_REAL\_t"),

label = c("Investment by sector","Capital demand","Labor demand","Emissions",

"Capital–labor ratio (sector)","Output price","K/L price",

"Investment by 6-sector group","Output shares by 6-sector groups (real)"),

group = "Industry level", keep = TRUE),

# ---------- Bilateral trade

data.table(symbol = "BITRADE\_REG\_t",

label = "Bilateral trade flows (macro regions)",

group = "Bilateral trade", keep = TRUE)

))

# keep only those actually available & marked keep

meta <- meta[symbol %in% available\_syms & keep == TRUE]

meta[, group := factor(group, levels = c("Nation level","Industry level","Bilateral trade"))]

# ---- The rest of your UI/server exactly as in your script ----

# (Keep dynamic filters, bindCache, BITRADE special plot, etc.)

# Replace only the initial results loader above and small label tweaks if needed.

# ...

# Grouped choices: user sees labels, value is the symbol

choices\_grouped <- lapply(split(meta, meta$group), function(d)

stats::setNames(d$symbol, d$label)

)

# Quick lookup helpers

label\_of <- function(sym) meta[symbol == sym, label][1]

group\_of <- function(sym) meta[symbol == sym, group][1]

# =========================

# Helpers

# =========================

# Show macro regions first in facets/legends

macro\_first <- function(vals) {

vals <- as.character(vals)

known <- c("EU28","NonEU28","WORLD")

rest <- setdiff(vals, known)

factor(vals, levels = c(known[known %in% vals], sort(rest)))

}

# Convert t -> calendar year (base 2014 -> t=1 = 2015). No-op if no 't'.

add\_year <- function(DT) {

if ("t" %in% names(DT)) DT[, year := 2014 + as.numeric(t)]

DT

}

# Long-format helper for level plots

to\_long <- function(DT, scenarios = c("baseline","ff55")) {

if (is.null(DT)) return(NULL)

id\_cols <- setdiff(names(DT), c(scenarios, "delta", "pct"))

melt(DT, id.vars = id\_cols, measure.vars = scenarios,

variable.name = "scenario", value.name = "value")

}

shallow\_copy <- function(x) {

# Use shallow() if available, otherwise fall back to copy()

if ("shallow" %in% getNamespaceExports("data.table")) {

data.table::shallow(x)

} else {

data.table::copy(x)

}

}

# =========================

# UI

# =========================

ui <- fluidPage(

tags$head(tags$style(HTML("

.container-fluid { padding-top: 6px; }

.selectize-dropdown .optgroup-header { font-weight: 600; }

"))),

titlePanel("FIDELIO diagnostics"),

sidebarLayout(

sidebarPanel(

selectizeInput(

"sym", "Variable:",

choices = choices\_grouped, selected = meta$symbol[1]

),

uiOutput("dynamic\_filters"),

radioButtons(

"view", "Show:",

c("Table" = "table", "Plot levels" = "plot\_lvl", "Plot Δ / %" = "plot\_var"),

selected = "plot\_lvl"

),

checkboxInput("asPercent", "Show pct as %", TRUE)

),

mainPanel(

conditionalPanel("input.view == 'table'", DTOutput("tbl")),

conditionalPanel("input.view != 'table'", plotOutput("plt", height = "650px"))

)

)

)

# =========================

# Server

# =========================

server <- function(input, output, session) {

# Selected table for current symbol

symDT <- reactive({

results\_by\_symbol[[input$sym]]

})

# ---------- Dynamic filters (built from keys present; skip scenarios/time) ----------

output$dynamic\_filters <- renderUI({

DT <- symDT()

if (is.null(DT)) return(NULL)

key\_cols <- setdiff(names(DT), c("baseline","ff55","delta","pct","t","year"))

show\_keys <- intersect(c("n","n1","i","c","au","oc"), key\_cols)

if (length(show\_keys) == 0) return(NULL)

# We create the inputs; choices are set in the observer below

pickers <- lapply(show\_keys, function(k) {

selectizeInput(

inputId = paste0("key\_", k),

label = paste0("Filter ", k, ":"),

choices = c("(all)" = ""), selected = "",

options = list(plugins = list("remove\_button")),

multiple = TRUE

)

})

do.call(tagList, pickers)

})

# Populate the choices (after UI exists)

observe({

DT <- symDT(); if (is.null(DT)) return()

key\_cols <- setdiff(names(DT), c("baseline","ff55","delta","pct","t","year"))

show\_keys <- intersect(c("n","n1","i","c","au","oc"), key\_cols)

for (k in show\_keys) {

vals <- sort(unique(as.character(DT[[k]])))

if (k == "n") vals <- levels(macro\_first(vals)) # macro regions first

updateSelectizeInput(session, paste0("key\_", k),

choices = c("(all)" = "", vals), server = TRUE)

}

})

# ---------- Apply filters ----------

# helper stays as before

shallow\_copy <- function(x) {

if ("shallow" %in% getNamespaceExports("data.table")) data.table::shallow(x) else data.table::copy(x)

}

# ---- filtered slice (define raw, then cache once) ----

filteredDT\_raw <- reactive({

DT <- results\_by\_symbol[[input$sym]]

if (is.null(DT)) return(NULL)

DT <- shallow\_copy(DT)

if ("n" %in% names(DT)) DT[, n := macro\_first(n)]

if (!"year" %in% names(DT) && "t" %in% names(DT)) DT[, year := 2014L + as.integer(t)]

for (k in intersect(c("n","n1","i","c","au","oc"), names(DT))) {

pick <- input[[paste0("key\_", k)]]

if (!is.null(pick) && length(pick) > 0 && any(nchar(pick) > 0)) {

DT <- DT[get(k) %in% pick]

}

}

if ("pct" %in% names(DT)) {

DT[, pct\_plot := if (isTRUE(input$asPercent)) 100 \* pct else pct]

}

DT[]

})

# cache based on the true dependencies of the slice

filteredDT <- shiny::bindCache(

filteredDT\_raw,

input$sym, input$asPercent,

input$key\_n, input$key\_i, input$key\_c, input$key\_n1, input$key\_au, input$key\_oc

)

# ---------- Table view ----------

output$tbl <- renderDT({

req(input$view == "table")

datatable(filteredDT(), options = list(pageLength = 20, scrollX = TRUE))

})

# ---------- Plot view ----------

output$plt <- renderPlot({

# ---- SPECIAL PLOT for BITRADE\_REG\_t -----------------------------------------

if (input$sym == "BITRADE\_REG\_t") {

BT <- filteredDT()

ord <- c("EEU","NWEU","SEU","USA","CHN","IND","OECD","NonOECD","ROW","TOT")

if ("n" %in% names(BT)) BT[, n := factor(as.character(n), levels = ord)]

if ("n1" %in% names(BT)) BT[, n1 := factor(as.character(n1), levels = ord)]

req(!is.null(BT), NROW(BT) > 0)

# Use precomputed columns from gdx2R: year, delta, pct (and pct\_plot from filteredDT)

xcol <- if ("year" %in% names(BT)) "year" else "t"

yvar <- if (input$view == "plot\_var") {

if ("pct\_plot" %in% names(BT)) "pct\_plot" else "pct"

} else "delta"

ylab <- if (yvar %in% c("pct","pct\_plot")) "Deviation wrt baseline (%)" else "Change (level)"

p <- ggplot(BT, aes\_string(x = xcol, y = yvar, color = "c", group = "c")) +

geom\_hline(yintercept = 0, color = "grey30", linewidth = 0.3) +

geom\_line(linewidth = 0.8) +

facet\_grid(n1 ~ n, scales = "free\_y") +

labs(x = "Year", y = ylab, color = "Commodity",

title = "Bilateral trade (macro regions): ff55 vs baseline") +

theme\_minimal(base\_size = 11) +

theme(

legend.position = "bottom",

panel.border = element\_rect(color = "black", fill = NA, linewidth = 0.4),

panel.background = element\_rect(fill = "grey97", color = NA)

)

return(p)

}

# ---- END SPECIAL PLOT for BITRADE\_REG\_t -------------------------------------

# ---- SPECIAL PLOT for OUT\_COMP6\_SHARE\_REAL\_t — Δ-share scatter --------------

if (input$sym == "OUT\_COMP6\_SHARE\_REAL\_t") {

DT <- filteredDT()

req(!is.null(DT), NROW(DT) > 0)

# Ensure year column

if (!"year" %in% names(DT) && "t" %in% names(DT)) DT[, year := 2014L + as.integer(t)]

req("year" %in% names(DT))

# Long format over scenarios

L <- to\_long(DT, c("baseline","ff55"))

req(!is.null(L), NROW(L) > 0)

# Endpoint years

yr0 <- L[, min(year, na.rm = TRUE)]

yrT <- L[, max(year, na.rm = TRUE)]

# Keep only endpoints

End <- L[year %in% c(yr0, yrT)]

# Compute change (final - initial) by (n,i,scenario)

# Cast years wide to ensure correct subtraction order

End\_w <- data.table::dcast(

End, n + i + scenario ~ year, value.var = "value"

)

# Drop rows missing either endpoint

End\_w <- End\_w[!is.na(get(as.character(yr0))) & !is.na(get(as.character(yrT)))]

# Δ share

End\_w[, delta := get(as.character(yrT)) - get(as.character(yr0))]

# Now cast scenarios wide: columns 'baseline' and 'ff55' contain Δ shares

SC <- data.table::dcast(

End\_w[, .(n, i, scenario, delta)],

n + i ~ scenario, value.var = "delta"

)

# If shares are fractions (0–1), convert to percentage points

frac\_like <- SC[, max(abs(c(baseline, ff55)), na.rm = TRUE)] <= 1.001

if (isTRUE(frac\_like)) {

SC[, `:=`(baseline = 100 \* baseline, ff55 = 100 \* ff55)]

ylab\_pp <- "Δ share (pp, 2015→2050)"

} else {

ylab\_pp <- "Δ share (pp or % units, 2015→2050)"

}

# Order regions (optional macro-first)

if ("n" %in% names(SC)) SC[, n := macro\_first(n)]

# Clean NAs (e.g., if a sector is absent in one scenario)

SC <- SC[is.finite(baseline) & is.finite(ff55)]

req(NROW(SC) > 0)

# Scatter: x = baseline Δ, y = ff55 Δ

p <- ggplot(SC, aes(x = baseline, y = ff55, color = i)) +

geom\_abline(slope = 1, intercept = 0, linewidth = 0.6, linetype = 2, alpha = 0.8) +

geom\_hline(yintercept = 0, linewidth = 0.4, linetype = 3, alpha = 0.7) +

geom\_vline(xintercept = 0, linewidth = 0.4, linetype = 3, alpha = 0.7) +

geom\_point(size = 2, alpha = 0.9) +

coord\_equal() +

facet\_wrap(~ n) +

labs(

x = paste0("Baseline ", yr0, "→", yrT),

y = paste0("ff55 ", yr0, "→", yrT),

color = "Sector group (6)",

title = paste0("Policy vs baseline change in output shares (", yr0, "→", yrT, ")"),

subtitle = "Points above the diagonal: policy increases the share vs baseline; below: reduces it"

) +

theme\_minimal(base\_size = 11) +

theme(

legend.position = "bottom",

panel.border = element\_rect(color = "black", fill = NA, linewidth = 0.4),

panel.background = element\_rect(fill = "grey97", color = NA)

)

return(p)

}

# ---- END SPECIAL PLOT for OUT\_COMP6\_SHARE\_REAL\_t -----------------------------

# Generic plots

req(input$view != "table")

DT <- filteredDT()

req(!is.null(DT), NROW(DT) > 0)

if (!data.table::is.data.table(DT)) data.table::setDT(DT)

# Available keys (skip scenario/time cols)

keys <- setdiff(names(DT), c("baseline","ff55","delta","pct","t","year"))

# x-axis: prefer 'year', then 't', else first key

xcol <- if ("year" %in% names(DT)) "year" else if ("t" %in% names(DT)) "t" else keys[1]

if (input$view == "plot\_lvl") {

L <- to\_long(DT, c("baseline","ff55"))

req(!is.null(L), NROW(L) > 0)

if (!data.table::is.data.table(L)) data.table::setDT(L)

# group lines by all keys except x and scenario

gcols <- setdiff(setdiff(names(L), c("scenario","value")), xcol)

if (length(gcols) == 0) {

L[, grp := "all"]

} else {

L[, grp := do.call(paste, c(.SD, list(sep = "|"))), .SDcols = gcols]

}

L[, grp2 := paste(grp, scenario, sep = "|")]

p <- ggplot(L, aes\_string(x = xcol, y = "value",

group = "grp2", color = "scenario",

linetype = "scenario")) +

geom\_line() +

geom\_point(size = 0.8) +

labs(x = xcol, y = label\_of(input$sym),

color = "Scenario", linetype = "Scenario",

title = paste("Levels —", label\_of(input$sym))) +

theme\_minimal()

# facet by the “largest” key among n > i > c > n1 > au > oc

facet\_key <- NULL

for (cand in c("n","i","c","n1","au","oc")) if (cand %in% gcols) { facet\_key <- cand; break }

if (!is.null(facet\_key)) p <- p + facet\_wrap(as.formula(paste("~", facet\_key)), scales = "free\_y")

p

} else {

# Δ or % plot (use the pre-scaled pct\_plot when available)

yvar <- if (isTRUE(input$asPercent)) {

if ("pct\_plot" %in% names(DT)) "pct\_plot" else "pct"

} else "delta"

ylab <- if (yvar %in% c("pct","pct\_plot")) "Change (%)" else "Change (level)"

# choose a facet variable automatically: n > i > c > n1 > au > oc

facet\_key <- NULL

for (cand in c("n","i","c","n1","au","oc")) if (cand %in% keys) { facet\_key <- cand; break }

# lines grouped by all keys except x & chosen facet

group\_cols <- setdiff(keys, c(xcol, facet\_key))

if (length(group\_cols) == 0L) {

DT[, grp := "all"]; color\_col <- NULL

} else {

DT[, grp := do.call(paste, c(.SD, list(sep = "|"))), .SDcols = group\_cols]

color\_col <- group\_cols[1]

}

p <- ggplot(DT, aes\_string(x = xcol, y = yvar, group = "grp")) +

geom\_hline(yintercept = 0, linetype = 2) +

geom\_line(alpha = 0.9) +

geom\_point(size = 0.7, alpha = 0.9) +

labs(x = xcol, y = ylab,

title = paste("ff55 vs baseline —", label\_of(input$sym))) +

theme\_minimal()

if (!is.null(color\_col)) p <- p + aes\_string(color = color\_col) + labs(color = color\_col)

if (!is.null(facet\_key)) p <- p + facet\_wrap(as.formula(paste("~", facet\_key)), scales = "free\_y")

p

}

})%>% shiny::bindCache(

input$sym, input$view, input$asPercent,

input$key\_n, input$key\_i, input$key\_c, input$key\_n1, input$key\_au, input$key\_oc

)

}

shinyApp(ui, server)

### App/results/app.R

library(shiny)

library(data.table)

library(ggplot2)

library(DT)

# ---- Package helpers ----

load\_symbol <- fidelioDiagnostics:::load\_symbol

load\_bundle <- fidelioDiagnostics:::load\_bundle

load\_manifest <- fidelioDiagnostics:::load\_manifest

resolve\_outputs\_dir <- fidelioDiagnostics:::resolve\_outputs\_dir

outputs\_dir\_from\_config <- fidelioDiagnostics:::outputs\_dir\_from\_config

cat("[APP] getwd(): ", getwd(), "\n")

cat("[APP] outputs/derived resolved to: ",

normalizePath(file.path("outputs","derived"), winslash="/", mustWork = FALSE), "\n")

# =========================

# Data loading

# =========================

get\_results <- function() {

dir <- file.path("outputs","derived") # same place as the diagnostic app

wanted <- c("GDPr\_t","TBr\_t","TB\_GDP\_t","I\_PP\_SECT6\_t","OUT\_COMP6\_SHARE\_REAL\_t","BITRADE\_REG\_t")

cat("\n[results\_app] Looking in:", normalizePath(dir, winslash = "/"), "\n")

# 1) Try bundle

b <- try(load\_bundle("results\_app", dir = dir), silent = TRUE)

if (!inherits(b, "try-error") && !is.null(b)) {

cat("[results\_app] Bundle found. Symbols inside:\n -", paste(names(b), collapse = "\n - "), "\n")

} else {

cat("[results\_app] No bundle found (or load error).\n")

b <- list()

}

# 2) Backfill per-symbol files for anything missing from bundle

missing <- setdiff(wanted, names(b))

if (length(missing)) {

cat("[results\_app] Backfilling per-symbol files for:", paste(missing, collapse = ", "), "\n")

for (s in missing) {

# show if a file exists (any supported extension)

files <- list.files(dir, pattern = paste0("^", s, "\\.(parquet|feather|fst|rds|csv)$"), ignore.case = FALSE)

cat(" ·", s, "files in dir:", if (length(files)) paste(files, collapse = ", ") else "(none)", "\n")

b[[s]] <- try(load\_symbol(s, dir = dir), silent = TRUE)

if (inherits(b[[s]], "try-error")) b[[s]] <- NULL

}

}

# 3) Report row counts before filtering empties

for (s in intersect(names(b), wanted)) {

n <- try(nrow(b[[s]]), silent = TRUE)

cat(" · rows(", s, ") = ", if (inherits(n, "try-error") || is.null(n)) "ERR/NULL" else n, "\n", sep = "")

}

# 4) Drop NULL/empty

b <- Filter(function(x) !is.null(x) && is.data.frame(x) && nrow(x) > 0, b)

# 5) Final report

have <- intersect(names(b), wanted)

miss <- setdiff(wanted, have)

cat("[results\_app] Loaded symbols:", paste(have, collapse = ", "), "\n")

if (length(miss)) cat("[results\_app] Missing or empty:", paste(miss, collapse = ", "), "\n")

b

}

results\_by\_symbol <- get\_results()

results\_by\_symbol <- Filter(function(x) !is.null(x) && nrow(x) > 0, results\_by\_symbol)

available\_syms <- names(results\_by\_symbol)

# =========================

# Meta (labels & groups)

# =========================

meta <- data.table::rbindlist(list(

# Nation level

data.table(

symbol = c("GDPr\_t","TBr\_t","TB\_GDP\_t"),

label = c("Real GDP","Trade balance (real)","Trade balance to GDP ratio"),

group = "Nation level", keep = TRUE

),

# Industry level (6 groups)

data.table(

symbol = c("I\_PP\_SECT6\_t","OUT\_COMP6\_SHARE\_REAL\_t"),

label = c("Investment by 6-sector group","Output shares by 6-sector groups (real)"),

group = "Industry level", keep = TRUE

),

# Bilateral trade (macro regions)

data.table(

symbol = "BITRADE\_REG\_t",

label = "Bilateral trade flows (macro regions)",

group = "Bilateral trade", keep = TRUE

)

))

meta <- meta[symbol %in% available\_syms & keep == TRUE]

meta[, group := factor(group, levels = c("Nation level","Industry level","Bilateral trade"))]

choices\_grouped <- lapply(split(meta, meta$group), function(d)

stats::setNames(d$symbol, d$label)

)

label\_of <- function(sym) meta[symbol == sym, label][1]

group\_of <- function(sym) meta[symbol == sym, group][1]

# =========================

# Helpers

# =========================

macro\_first <- function(vals) {

vals <- as.character(vals)

known <- c("EU28","NonEU28","WORLD")

rest <- setdiff(vals, known)

factor(vals, levels = c(known[known %in% vals], sort(rest)))

}

add\_year <- function(DT) {

if ("t" %in% names(DT)) DT[, year := 2014L + as.integer(t)]

DT

}

to\_long <- function(DT, scenarios = c("baseline","ff55")) {

if (is.null(DT)) return(NULL)

id\_cols <- setdiff(names(DT), c(scenarios, "delta", "pct"))

melt(DT, id.vars = id\_cols, measure.vars = scenarios,

variable.name = "scenario", value.name = "value")

}

shallow\_copy <- function(x) {

if ("shallow" %in% getNamespaceExports("data.table")) data.table::shallow(x) else data.table::copy(x)

}

# =========================

# UI

# =========================

ui <- fluidPage(

tags$head(tags$style(HTML("

.container-fluid { padding-top: 6px; }

.selectize-dropdown .optgroup-header { font-weight: 600; }

"))),

titlePanel("FIDELIO results"),

sidebarLayout(

sidebarPanel(

selectizeInput("sym", "Variable:", choices = choices\_grouped, selected = meta$symbol[1]),

uiOutput("dynamic\_filters"),

radioButtons("view", "Show:",

c("Table" = "table", "Plot levels" = "plot\_lvl", "Plot Δ / %" = "plot\_var"),

selected = "plot\_lvl"),

checkboxInput("asPercent", "Show pct as %", TRUE)

),

mainPanel(

conditionalPanel("input.view == 'table'", DTOutput("tbl")),

conditionalPanel("input.view != 'table'", plotOutput("plt", height = "650px"))

)

)

)

# =========================

# Server

# =========================

server <- function(input, output, session) {

symDT <- reactive({ results\_by\_symbol[[input$sym]] })

# Dynamic filters built from present keys (skip scenario/time)

output$dynamic\_filters <- renderUI({

DT <- symDT()

if (is.null(DT)) return(NULL)

key\_cols <- setdiff(names(DT), c("baseline","ff55","delta","pct","t","year"))

show\_keys <- intersect(c("n","n1","i","c","au","oc"), key\_cols)

if (length(show\_keys) == 0) return(NULL)

# Only create pickers with >1 unique value

pickers <- list()

for (k in show\_keys) {

vals <- sort(unique(as.character(DT[[k]])))

# skip if only one unique value or all NA/empty

if (length(vals) <= 1 || all(!nzchar(vals))) next

pickers[[k]] <- selectizeInput(

inputId = paste0("key\_", k),

label = paste0("Filter ", k, ":"),

choices = c("(all)" = "", vals), selected = "",

options = list(plugins = list("remove\_button")),

multiple = TRUE

)

}

if (length(pickers) == 0) return(NULL)

do.call(tagList, pickers)

})

observe({

DT <- symDT(); if (is.null(DT)) return()

key\_cols <- setdiff(names(DT), c("baseline","ff55","delta","pct","t","year"))

show\_keys <- intersect(c("n","n1","i","c","au","oc"), key\_cols)

for (k in show\_keys) {

vals <- sort(unique(as.character(DT[[k]])))

if (length(vals) <= 1 || all(!nzchar(vals))) next # nothing to update/show

if (k == "n") vals <- levels(macro\_first(vals))

updateSelectizeInput(session, paste0("key\_", k),

choices = c("(all)" = "", vals), server = TRUE)

}

})

# Filtered slice

filteredDT\_raw <- reactive({

DT <- results\_by\_symbol[[input$sym]]

if (is.null(DT)) return(NULL)

DT <- shallow\_copy(DT)

if ("n" %in% names(DT)) DT[, n := macro\_first(n)]

if (!"year" %in% names(DT) && "t" %in% names(DT)) DT[, year := 2014L + as.integer(t)]

for (k in intersect(c("n","n1","i","c","au","oc"), names(DT))) {

pick <- input[[paste0("key\_", k)]]

if (!is.null(pick) && length(pick) > 0 && any(nchar(pick) > 0)) {

DT <- DT[get(k) %in% pick]

}

}

if ("pct" %in% names(DT)) DT[, pct\_plot := if (isTRUE(input$asPercent)) 100 \* pct else pct]

DT[]

})

filteredDT <- shiny::bindCache(

filteredDT\_raw, input$sym, input$asPercent,

input$key\_n, input$key\_i, input$key\_c, input$key\_n1, input$key\_au, input$key\_oc

)

# Table

output$tbl <- renderDT({

req(input$view == "table")

datatable(filteredDT(), options = list(pageLength = 20, scrollX = TRUE))

})

# Plot

output$plt <- renderPlot({

# ---- SPECIAL: BITRADE\_REG\_t (macro-regions, c already filtered to TOT in bundle) ----

if (input$sym == "BITRADE\_REG\_t") {

BT <- filteredDT()

ord <- c("EEU","NWEU","SEU","USA","CHN","IND","OECD","NonOECD","ROW","TOT")

if ("n" %in% names(BT)) BT[, n := factor(as.character(n), levels = ord)]

if ("n1" %in% names(BT)) BT[, n1 := factor(as.character(n1), levels = ord)]

req(!is.null(BT), NROW(BT) > 0)

xcol <- if ("year" %in% names(BT)) "year" else "t"

yvar <- if (input$view == "plot\_var") {

if ("pct\_plot" %in% names(BT)) "pct\_plot" else "pct"

} else "delta"

ylab <- if (yvar %in% c("pct","pct\_plot")) "Deviation wrt baseline (%)" else "Change (level)"

p <- ggplot(BT, aes\_string(x = xcol, y = yvar)) +

geom\_hline(yintercept = 0, color = "grey30", linewidth = 0.3) +

geom\_line(linewidth = 0.8) +

facet\_grid(n1 ~ n, scales = "free\_y") +

labs(x = "Year", y = ylab, title = "Bilateral trade (macro regions): ff55 vs baseline") +

theme\_minimal(base\_size = 11) +

theme(

legend.position = "none",

panel.border = element\_rect(color = "black", fill = NA, linewidth = 0.4),

panel.background = element\_rect(fill = "grey97", color = NA)

)

return(p)

}

# ---- SPECIAL: OUT\_COMP6\_SHARE\_REAL\_t — Δ-share scatter (policy vs baseline) ----

if (input$sym == "OUT\_COMP6\_SHARE\_REAL\_t") {

DT <- filteredDT(); req(!is.null(DT), NROW(DT) > 0)

if (!"year" %in% names(DT) && "t" %in% names(DT)) DT[, year := 2014L + as.integer(t)]

L <- to\_long(DT, c("baseline","ff55")); req(!is.null(L), NROW(L) > 0)

yr0 <- L[, min(year, na.rm = TRUE)]; yrT <- L[, max(year, na.rm = TRUE)]

End <- L[year %in% c(yr0, yrT)]

End\_w <- data.table::dcast(End, n + i + scenario ~ year, value.var = "value")

End\_w <- End\_w[!is.na(get(as.character(yr0))) & !is.na(get(as.character(yrT)))]

End\_w[, delta := get(as.character(yrT)) - get(as.character(yr0))]

SC <- data.table::dcast(End\_w[, .(n, i, scenario, delta)], n + i ~ scenario, value.var = "delta")

frac\_like <- SC[, max(abs(c(baseline, ff55)), na.rm = TRUE)] <= 1.001

if (isTRUE(frac\_like)) SC[, `:=`(baseline = 100 \* baseline, ff55 = 100 \* ff55)]

if ("n" %in% names(SC)) SC[, n := macro\_first(n)]

SC <- SC[is.finite(baseline) & is.finite(ff55)]; req(NROW(SC) > 0)

p <- ggplot(SC, aes(x = baseline, y = ff55, color = i)) +

geom\_abline(slope = 1, intercept = 0, linewidth = 0.6, linetype = 2, alpha = 0.8) +

geom\_hline(yintercept = 0, linewidth = 0.4, linetype = 3, alpha = 0.7) +

geom\_vline(xintercept = 0, linewidth = 0.4, linetype = 3, alpha = 0.7) +

geom\_point(size = 2, alpha = 0.9) +

coord\_equal() +

facet\_wrap(~ n) +

labs(

x = paste0("Baseline ", yr0, "→", yrT),

y = paste0("ff55 ", yr0, "→", yrT),

color = "Sector group (6)",

title = paste0("Policy vs baseline change in output shares (", yr0, "→", yrT, ")")

) +

theme\_minimal(base\_size = 11) +

theme(

legend.position = "bottom",

panel.border = element\_rect(color = "black", fill = NA, linewidth = 0.4),

panel.background = element\_rect(fill = "grey97", color = NA)

)

return(p)

}

# ---- Generic plots (for GDPr\_t, TBr\_t, TB\_GDP\_t, I\_PP\_SECT6\_t) ----

req(input$view != "table")

DT <- filteredDT(); req(!is.null(DT), NROW(DT) > 0)

if (!data.table::is.data.table(DT)) data.table::setDT(DT)

keys <- setdiff(names(DT), c("baseline","ff55","delta","pct","t","year"))

xcol <- if ("year" %in% names(DT)) "year" else if ("t" %in% names(DT)) "t" else keys[1]

if (input$view == "plot\_lvl") {

L <- to\_long(DT, c("baseline","ff55")); req(!is.null(L), NROW(L) > 0)

if (!data.table::is.data.table(L)) data.table::setDT(L)

gcols <- setdiff(setdiff(names(L), c("scenario","value")), xcol)

if (length(gcols) == 0) { L[, grp := "all"] } else {

L[, grp := do.call(paste, c(.SD, list(sep = "|"))), .SDcols = gcols]

}

L[, grp2 := paste(grp, scenario, sep = "|")]

p <- ggplot(L, aes\_string(x = xcol, y = "value", group = "grp2", color = "scenario", linetype = "scenario")) +

geom\_line() + geom\_point(size = 0.8) +

labs(x = xcol, y = label\_of(input$sym), color = "Scenario", linetype = "Scenario",

title = paste("Levels —", label\_of(input$sym))) +

theme\_minimal()

facet\_key <- NULL

for (cand in c("n","i","c","n1","au","oc")) if (cand %in% gcols) { facet\_key <- cand; break }

if (!is.null(facet\_key)) p <- p + facet\_wrap(as.formula(paste("~", facet\_key)), scales = "free\_y")

p

} else {

yvar <- if (isTRUE(input$asPercent)) { if ("pct\_plot" %in% names(DT)) "pct\_plot" else "pct" } else "delta"

ylab <- if (yvar %in% c("pct","pct\_plot")) "Change (%)" else "Change (level)"

facet\_key <- NULL

for (cand in c("n","i","c","n1","au","oc")) if (cand %in% keys) { facet\_key <- cand; break }

group\_cols <- setdiff(keys, c(xcol, facet\_key))

if (length(group\_cols) == 0L) { DT[, grp := "all"]; color\_col <- NULL } else {

DT[, grp := do.call(paste, c(.SD, list(sep = "|"))), .SDcols = group\_cols]

color\_col <- group\_cols[1]

}

p <- ggplot(DT, aes\_string(x = xcol, y = yvar, group = "grp")) +

geom\_hline(yintercept = 0, linetype = 2) +

geom\_line(alpha = 0.9) + geom\_point(size = 0.7, alpha = 0.9) +

labs(x = xcol, y = ylab, title = paste("ff55 vs baseline —", label\_of(input$sym))) +

theme\_minimal()

if (!is.null(color\_col)) p <- p + aes\_string(color = color\_col) + labs(color = color\_col)

if (!is.null(facet\_key)) p <- p + facet\_wrap(as.formula(paste("~", facet\_key)), scales = "free\_y")

p

}

}) %>% shiny::bindCache(

input$sym, input$view, input$asPercent,

input$key\_n, input$key\_i, input$key\_c, input$key\_n1, input$key\_au, input$key\_oc

)

}

shinyApp(ui, server)

## R

### Aaa\_import.R

# ==============================================================================

#'==== aaa\_imports.R ========

# ==============================================================================

# -------------- Internal imports and global variables--------------------------

#' This file:

#' - declares package-level imports (e.g., {data.table})

#' - registers NSE symbols/column names used across the package so

#' `R CMD check` doesn’t warn about “no visible binding”.

# ---------------- declares package-level imports ----------------------------

#' @details This file is only used for package-level settings.

#' It does not export any user-facing functions.

#' @keywords internal

#' @import data.table

#' @importFrom utils globalVariables

#' @noRd

NULL

# ----data.table NSE symbols and common column names created on the fly --------

if (getRversion() >= "2.15.1") {

utils::globalVariables(c(

".SD", ".N", ".I", ".GRP",

"value", "delta", "pct",

"n", "n1", "au", "oc", "i", "c", "t", "year",

"scenario", "baseline", "ff55"

))

}

### App\_loaders

#' @keywords internal

resolve\_project\_root <- function(root = NULL) {

# Priority: explicit arg > option > env var > current getwd()

cand <- c(root, getOption("fidelio.project", ""), Sys.getenv("FIDELIO\_PROJECT", ""), getwd())

cand <- cand[nzchar(cand)]

for (start in cand) {

cur <- normalizePath(start, winslash = "/", mustWork = TRUE)

repeat {

# Heuristics for project root

has\_cfg <- file.exists(file.path(cur, "config", "project.yml"))

has\_rproj <- length(Sys.glob(file.path(cur, "\*.Rproj"))) > 0

if (has\_cfg || has\_rproj) return(cur)

parent <- dirname(cur)

if (identical(parent, cur)) break

cur <- parent

}

}

stop("Cannot resolve project root. Set options(fidelio.project='...') or FIDELIO\_PROJECT.")

}

#' @keywords internal

outputs\_dir\_from\_config <- function(root = NULL, config\_path = NULL) {

cfg\_file <- if (!is.null(config\_path)) {

normalizePath(config\_path, winslash = "/", mustWork = TRUE)

} else {

root <- resolve\_project\_root(root)

file.path(root, "config", "project.yml")

}

if (!file.exists(cfg\_file)) stop("Config file not found: ", cfg\_file)

cfg <- load\_config(cfg\_file) # your existing reader (returns normalized paths)

out <- file.path(cfg$paths$outputs, "derived") # cfg$paths$outputs is already absolute in your setup

normalizePath(out, winslash = "/", mustWork = TRUE)

}

#' @keywords internal

resolve\_outputs\_dir <- function(dir = NULL) {

# Priority: explicit arg > option > env var > default relative folder

cand <- c(

dir,

getOption("fidelio.outputs", ""),

Sys.getenv("FIDELIO\_OUTPUTS", ""),

file.path(getwd(), "outputs", "derived")

)

cand <- normalizePath(cand[nzchar(cand)], winslash = "/", mustWork = FALSE)

hit <- cand[file.exists(cand)]

if (!length(hit)) stop(

"Cannot find outputs directory. Set options(fidelio.outputs='...') ",

"or env var FIDELIO\_OUTPUTS, or pass 'outputs\_dir' to launch\_app()."

)

hit[[1]]

}

#' @keywords internal

load\_manifest <- function(dir = NULL) {

dir <- resolve\_outputs\_dir(dir)

man\_r <- file.path(dir, "manifest.rds")

man\_c <- file.path(dir, "manifest.csv")

if (file.exists(man\_r)) return(readRDS(man\_r))

if (file.exists(man\_c)) return(data.table::fread(man\_c))

stop("No manifest.rds or manifest.csv found in: ", dir)

}

#' @keywords internal

load\_symbol <- function(symbol, dir = NULL,

prefer = c("parquet","feather","fst","rds","csv")) {

dir <- resolve\_outputs\_dir(dir)

man <- load\_manifest(dir)

row <- man[symbol == !!symbol][match(prefer, format, nomatch = 0L) > 0][1]

if (!nrow(row)) stop("Symbol not found in manifest: ", symbol)

p <- file.path(dir, row$filename %||% row$path) # support either column name

if (grepl("\\.parquet$", p)) return(arrow::read\_parquet(p))

if (grepl("\\.feather$", p)) return(arrow::read\_feather(p))

if (grepl("\\.fst$", p)) return(fst::read\_fst(p, as.data.table = TRUE))

if (grepl("\\.rds$", p)) return(readRDS(p))

if (grepl("\\.csv$", p)) return(data.table::fread(p))

stop("Unknown format: ", p)

}

#' @keywords internal

load\_bundle <- function(name, dir = NULL) {

dir <- resolve\_outputs\_dir(dir)

p <- file.path(dir, paste0("bundle\_", name, ".rds"))

if (file.exists(p)) return(readRDS(p))

NULL

}

### Config\_load.R

# ==============================================================================

# ==== Configuration ========

# ==============================================================================

# ---------------------------- Best-effort read--------------------------------

safe\_read\_yaml <- function(file) {

if (!file.exists(file)) return(list())

y <- tryCatch(yaml::read\_yaml(file), error = function(e) NULL)

if (is.null(y) || !is.list(y)) return(list())

y

}

# ---------------------------- Load configuration -----------------------------

#' Load and validate configuration (defaults + overrides)

#' @export

load\_config <- function(path = "config/project.yml", verbose = TRUE) {

# -- helpers ---------------------------------------------------------------

# Resolve paths relative to the project root used by proj\_path()

# (proj\_path() is already available in your package; we reuse it)

.is\_abs <- function(p) {

# Windows drive (C:\ or C:/), UNC (\\server\share), or Unix root (/)

grepl("^[A-Za-z]:[/\\\\]|^/|^\\\\\\\\", p)

}

.to\_abs <- function(p) {

p <- path.expand(p)

if (.is\_abs(p)) {

normalizePath(p, winslash = "/", mustWork = FALSE)

} else {

normalizePath(proj\_path(p), winslash = "/", mustWork = FALSE)

}

}

safe\_read\_yaml <- function(f) {

if (is.null(f) || !nzchar(f) || !file.exists(f)) return(list())

yaml::read\_yaml(f)

}

# -- load default + project ------------------------------------------------

def\_file <- proj\_path("config", "default.yml")

override\_file <- if (is.character(path) && length(path) == 1L) proj\_path(path) else NULL

default <- safe\_read\_yaml(def\_file)

override <- safe\_read\_yaml(override\_file)

# merge (project overrides default)

cfg <- utils::modifyList(default, override, keep.null = TRUE)

# -- ensure sections exist -------------------------------------------------

if (is.null(cfg$paths)) cfg$paths <- list()

if (is.null(cfg$extract)) cfg$extract <- list(include = character())

if (is.null(cfg$derived)) cfg$derived <- list(include = character())

if (is.null(cfg$validate)) cfg$validate <- list(rules = character())

# -- defaults --------------------------------------------------------------

if (is.null(cfg$paths$gdx\_dir) || !nzchar(cfg$paths$gdx\_dir)) cfg$paths$gdx\_dir <- "data-raw/gdx"

if (is.null(cfg$paths$outputs) || !nzchar(cfg$paths$outputs)) cfg$paths$outputs <- "outputs"

if (is.null(cfg$paths$cache) || !nzchar(cfg$paths$cache)) cfg$paths$cache <- "outputs/cache"

if (is.null(cfg$scenarios) || !length(cfg$scenarios)) cfg$scenarios <- "baseline"

# -- optional session overrides -------------------------------------------

# Let advanced users override these in the current R session

cfg$paths$gdx\_dir <- getOption("fidelio.gdx\_dir", cfg$paths$gdx\_dir)

cfg$paths$outputs <- getOption("fidelio.outputs", cfg$paths$outputs)

cfg$paths$cache <- getOption("fidelio.cache", cfg$paths$cache)

# -- normalize to absolute paths ------------------------------------------

cfg$paths$gdx\_dir <- .to\_abs(cfg$paths$gdx\_dir)

cfg$paths$outputs <- .to\_abs(cfg$paths$outputs)

cfg$paths$cache <- .to\_abs(cfg$paths$cache)

# -- ensure dirs exist -----------------------------------------------------

dir.create(cfg$paths$outputs, showWarnings = FALSE, recursive = TRUE)

dir.create(cfg$paths$cache, showWarnings = FALSE, recursive = TRUE)

# -- threads default -------------------------------------------------------

if (is.null(cfg$threads)) cfg$threads <- data.table::getDTthreads()

# -- breadcrumbs -----------------------------------------------------------

if (isTRUE(verbose)) {

message("[CONFIG] Using gdx\_dir = ", cfg$paths$gdx\_dir)

message("[CONFIG] Using outputs = ", cfg$paths$outputs)

}

cfg

}

#...............................................................................

### Derive\_indicators.R

# ==============================================================================

# ==== derive\_indicators ========

# ==============================================================================

# ---- Sector group mapping: generalized by leading letter ---------------------

# high\_energy\_manufacturing stays explicit; everything else is pattern-based.

# - primary = codes starting with A or B

# - low\_energy\_manufacturing = codes starting with C EXCEPT the high-energy list

# - utilities\_construction = codes starting with D, E, or F

# - market\_services = codes starting with G..N (G,H,I,J,K,L,M,N)

# - public\_personal\_services = codes starting with O..U (O,P,Q,R,S,T,U)

# Any leftover codes -> "other".

.sector\_group6\_vec <- function(ic,

high\_energy = c("C17","C19","C20","C23","C24")) {

ic <- as.character(ic)

# match "C24", "C24.", "C24\_xx", etc.

he <- if (length(high\_energy)) {

Reduce(`|`, lapply(high\_energy, function(h) startsWith(ic, h)))

} else {

rep(FALSE, length(ic))

}

primary <- grepl("^[AB]", ic)

low\_man <- grepl("^C", ic) & !he

util\_cons <- grepl("^[DEF]", ic)

market <- grepl("^[G-N]", ic) # ASCII range G..N

pub\_pers <- grepl("^[O-U]", ic) # ASCII range O..U

data.table::fcase(

he, "high\_energy\_manufacturing",

low\_man, "low\_energy\_manufacturing",

primary, "primary",

util\_cons, "utilities\_construction",

market, "market\_services",

pub\_pers, "public\_personal\_services",

default = "other"

)

}

# Add 6-group sector aggregates to an (n,i,t, <scenarios...>) wide table.

# If `append\_original = TRUE` it keeps original industries and ADDs grouped rows.

add\_sector\_groups\_additive <- function(DT, scenarios, append\_original = TRUE) {

stopifnot(is.data.table(DT), all(scenarios %in% names(DT)))

if (!"i" %in% names(DT)) return(DT)

DT <- data.table::copy(DT)

# drop any pre-aggregated industry rows you might carry (e.g., "TOT")

if ("i" %in% names(DT))

DT <- DT[i != "TOT"]

# map to 6 groups and aggregate

G <- DT[, .(i = .sector\_group6\_vec(i), n, t, across = 1L)]

G <- cbind(G[, .(i, n, t)], DT[, ..scenarios]) # rebind scenario cols

G <- G[, lapply(.SD, sum, na.rm = TRUE), .SDcols = scenarios, by = .(n, i, t)]

# optional total across groups (nice for composition checks)

TOT <- G[, lapply(.SD, sum, na.rm = TRUE), .SDcols = scenarios, by = .(n, t)]

TOT[, i := "TOT\_G6"]

data.table::setcolorder(TOT, c("n","i","t", scenarios))

out <- if (isTRUE(append\_original)) {

data.table::rbindlist(list(DT, G, TOT), use.names = TRUE)

} else {

data.table::rbindlist(list(G, TOT), use.names = TRUE)

}

out[]

}

# --- Derived indicators -----------------------------------

#' Compute all derived indicators requested in config

#' @export

derive\_all <- function(raw, cfg) {

# you can drive which ones to compute from YAML (cfg$derived$include)

wanted <- cfg$derived$include

if (is.null(wanted) || !length(wanted)) {

# default: compute a few common ones

wanted <- c("TB\_GDP\_t")

}

out <- list()

if ("TB\_GDP\_t" %in% wanted) out$TB\_GDP\_t <- derive\_TB\_GDP(raw)

# return only non-null results

Filter(Negate(is.null), out)

}

# ---- helpers used by many derived indicators ----

# select base/policy columns from a wide table and rename with a prefix

.pick\_base\_pol <- function(DT, keys = c("n","t"), base\_scn, pol\_scn, prefix) {

if (is.null(DT) || !all(c(base\_scn, pol\_scn) %in% names(DT))) return(NULL)

cols <- c(intersect(keys, names(DT)), base\_scn, pol\_scn)

out <- DT[, ..cols]

data.table::setnames(out, c(base\_scn, pol\_scn), paste0(prefix, c("\_base","\_pol")))

out

}

# region map for BITRADE regionalization (you can later move these lists to YAML)

.region\_map\_vec <- function(cc) {

seu <- c("CYP","ESP","GRC","ITA","MLT","PRT")

eeu <- c("BGR","CZE","EST","HRV","HUN","LTU","LVA","POL","ROU","SVN","SVK")

nweu <- c("AUT","BEL","DEU","DNK","FIN","FRA","GBR","IRL","LUX","NLD","SWE")

OECD\_NO\_USA <- c("CAN","JPN","KOR","AUS","CHE","NOR","TUR","MEX") # USA separate

nonOECD <- c("RUS","BRA","ARG","IDN","ZAF","SAU")

data.table::fcase(

cc %chin% eeu, "EEU",

cc %chin% nweu, "NWEU",

cc %chin% seu, "SEU",

cc == "USA", "USA",

cc == "CHN", "CHN",

cc == "IND", "IND",

cc %chin% OECD\_NO\_USA, "OECD",

cc %chin% nonOECD, "NonOECD",

default = "ROW"

)

}

# ---- derive indicators from \*wide\* base tables (fixed .SD sums) ----

#' Compute all derived indicators that were previously built from wide tables

#' @param rs named list of wide data.tables (results\_by\_symbol)

#' @param cfg config (for scenarios + EU28)

#' @return named list of wide data.tables

#' @export

derive\_from\_wide <- function(rs, cfg) {

if (is.null(rs) || !length(rs)) return(list())

base\_scn <- cfg$scenarios[1]

pol\_scn <- cfg$scenarios[length(cfg$scenarios)]

EU28 <- cfg$groups$EU28

out <- list()

# --- I\_TOT\_PP\_t = sum\_i I\_PP\_t ----------------------------------------------

I\_PP <- rs[["I\_PP\_t"]]

if (!is.null(I\_PP)) {

I\_PP <- norm\_key\_types(I\_PP)

I\_TOT <- I\_PP[, lapply(.SD, sum, na.rm = TRUE),

.SDcols = c(base\_scn, pol\_scn), by = .(n, t)]

I\_TOT <- add\_macroregions\_additive(I\_TOT, EU28, scenarios = cfg$scenarios)

I\_TOT <- add\_var\_cols(I\_TOT, base = base\_scn, pol = pol\_scn)

out[["I\_TOT\_PP\_t"]] <- I\_TOT

}

# --- TB\_t = X - M using value sums -----------------------------------------

TB\_X <- sum\_value\_by(

priceDT = rs[["P\_USE\_t"]],

qtyDT = rs[["USE\_PP\_t"]],

by = c("n","t"),

filter = list(au = "X"),

scenarios = cfg$scenarios

)

TB\_M <- sum\_value\_by(

priceDT = rs[["P\_Mcif\_t"]],

qtyDT = rs[["M\_TOT\_t"]],

by = c("n","t"),

scenarios = cfg$scenarios

)

if (!is.null(TB\_X) && !is.null(TB\_M)) {

TB <- merge(TB\_X, TB\_M, by = c("n","t"), all = TRUE, suffixes = c("\_X","\_M"))

for (s in cfg$scenarios) {

x <- paste0(s,"\_X"); m <- paste0(s,"\_M")

if (all(c(x, m) %in% names(TB))) TB[, (s) := get(x) - get(m)]

}

TB <- TB[, c("n","t", base\_scn, pol\_scn), with = FALSE]

TB <- add\_macroregions\_additive(TB, EU28, scenarios = cfg$scenarios)

TB <- add\_var\_cols(TB, base = base\_scn, pol = pol\_scn)

out[["TB\_t"]] <- TB

}

# --- Real TB from BITRADE (no pre-agg rows) → TBr\_t -------------------------

BT <- rs[["BITRADE\_t"]]

if (!is.null(BT) && is.data.table(BT)) {

BT <- norm\_key\_types(BT)

if ("n" %in% names(BT)) BT <- BT[!(n %in% c("EU28","NonEU28","WORLD"))]

if ("n1" %in% names(BT)) BT <- BT[!(n1 %in% c("EU28","NonEU28","WORLD"))]

if ("c" %in% names(BT)) BT <- BT[c != "TOT"]

meas\_cols <- intersect(names(BT), cfg$scenarios)

if (length(meas\_cols) >= 1) {

# exporter totals

X\_tot <- BT[, lapply(.SD, sum), .SDcols = meas\_cols, by = .(n1, t)]

X\_tot[, n := n1][, n1 := NULL]

data.table::setcolorder(X\_tot, c("n","t",meas\_cols))

# importer totals

M\_tot <- BT[, lapply(.SD, sum), .SDcols = meas\_cols, by = .(n, t)]

data.table::setcolorder(M\_tot, c("n","t",meas\_cols))

# TB = X - M

TB\_cty <- merge(X\_tot, M\_tot, by = c("n","t"), all = TRUE, suffixes = c("\_X","\_M"))

for (m in meas\_cols) {

x <- paste0(m,"\_X"); mm <- paste0(m,"\_M")

TB\_cty[is.na(get(x)), (x) := 0]

TB\_cty[is.na(get(mm)), (mm) := 0]

TB\_cty[, (m) := get(x) - get(mm)]

}

TB\_cty <- TB\_cty[, c("n","t",meas\_cols), with = FALSE]

TBr <- add\_macroregions\_additive(TB\_cty, EU28, scenarios = cfg$scenarios)

TBr <- add\_var\_cols(TBr, base = base\_scn, pol = pol\_scn)

out[["TBr\_t"]] <- TBr

}

}

# --- TB/GDP (from aggregated wide TB & GDP) ---------------------------------

TB <- out[["TB\_t"]]

GDP <- rs[["GDPr\_t"]]

if (!is.null(TB) && !is.null(GDP)) {

TB <- norm\_key\_types(TB); GDP <- norm\_key\_types(GDP)

TB\_GDP <- merge(TB[, c("n","t", base\_scn, pol\_scn), with = FALSE],

GDP[, c("n","t", base\_scn, pol\_scn), with = FALSE],

by = c("n","t"), suffixes = c("\_TB","\_GDP"))

for (s in c(base\_scn, pol\_scn)) {

TB\_GDP[, (s) := fifelse(get(paste0(s,"\_GDP")) == 0 | is.na(get(paste0(s,"\_GDP"))),

NA\_real\_, get(paste0(s,"\_TB")) / get(paste0(s,"\_GDP")))]

}

TB\_GDP <- TB\_GDP[, c("n","t", base\_scn, pol\_scn), with = FALSE]

TB\_GDP <- add\_var\_cols(TB\_GDP, base = base\_scn, pol = pol\_scn)

out[["TB\_GDP\_t"]] <- TB\_GDP

}

# --- Foreign Savings FS\_t ---------------------------------------------------

FS\_INV <- sum\_value\_by(

priceDT = rs[["P\_I\_t"]],

qtyDT = rs[["I\_PP\_t"]],

by = c("n","t"),

scenarios = cfg$scenarios

)

HSAV <- rs[["HSAVR\_t"]]; HDY <- rs[["HDY\_VAL\_t"]]

GSUR <- rs[["GSUR\_VAL\_t"]]; GINV <- rs[["GINV\_VAL\_t"]]

objs <- list(FS\_INV = FS\_INV, TB = out[["TB\_t"]], HSAV = HSAV, HDY = HDY, GSUR = GSUR, GINV = GINV)

if (all(!vapply(objs, is.null, TRUE))) {

for (nm in names(objs)) objs[[nm]] <- norm\_key\_types(objs[[nm]])

keys <- unique(data.table::rbindlist(lapply(objs, function(D) D[, .(n,t)]), use.names = TRUE))

data.table::setkey(keys, n, t)

m2 <- function(L, R) merge(L, R, by = c("n","t"), all = TRUE)

.pick <- function(D, prefix) {

if (is.null(D)) return(keys[, .(n,t)][0])

out <- D[, c("n","t", base\_scn, pol\_scn), with = FALSE]

data.table::setnames(out, c(base\_scn, pol\_scn), paste0(prefix, c("\_base","\_pol")))

out

}

X <- m2(keys, .pick(objs$FS\_INV, "INV"))

X <- m2(X, .pick(objs$TB, "TB"))

X <- m2(X, .pick(objs$HSAV, "HSAV"))

X <- m2(X, .pick(objs$HDY, "HDY"))

X <- m2(X, .pick(objs$GSUR, "GSUR"))

X <- m2(X, .pick(objs$GINV, "GINV"))

for (s in c("base","pol")) {

X[, (paste0("FS\_", s)) :=

get(paste0("INV\_", s)) -

get(paste0("TB\_", s)) -

(get(paste0("HSAV\_", s)) \* get(paste0("HDY\_", s))) -

get(paste0("GSUR\_", s)) -

get(paste0("GINV\_", s))]

}

FS\_out <- X[, .(n, t, base = FS\_base, pol = FS\_pol)]

data.table::setnames(FS\_out, c("base","pol"), c(base\_scn, pol\_scn))

FS\_out <- add\_macroregions\_additive(FS\_out, EU28, scenarios = cfg$scenarios)

FS\_out <- add\_var\_cols(FS\_out, base = base\_scn, pol = pol\_scn)

out[["FS\_t"]] <- FS\_out

}

# --- Domestic Savings DS\_t --------------------------------------------------

HSAV <- rs[["HSAVR\_t"]]; HDY <- rs[["HDY\_VAL\_t"]]

GSUR <- rs[["GSUR\_VAL\_t"]]; GINV <- rs[["GINV\_VAL\_t"]]

objs2 <- list(HSAV=HSAV, HDY=HDY, GSUR=GSUR, GINV=GINV)

if (all(!vapply(objs2, is.null, TRUE))) {

for (nm in names(objs2)) objs2[[nm]] <- norm\_key\_types(objs2[[nm]])

keys <- unique(data.table::rbindlist(lapply(objs2, function(D) D[, .(n,t)]), use.names = TRUE))

data.table::setkey(keys, n, t)

m2 <- function(L, R) merge(L, R, by = c("n","t"), all = TRUE)

.pick <- function(D, prefix) {

out <- D[, c("n","t", base\_scn, pol\_scn), with = FALSE]

data.table::setnames(out, c(base\_scn, pol\_scn), paste0(prefix, c("\_base","\_pol")))

out

}

DS <- m2(keys, .pick(HSAV,"HSAV"))

DS <- m2(DS, .pick(HDY, "HDY"))

DS <- m2(DS, .pick(GSUR,"GSUR"))

DS <- m2(DS, .pick(GINV,"GINV"))

for (s in c("base","pol")) {

DS[, (paste0("DS\_", s)) :=

get(paste0("GSUR\_", s)) +

get(paste0("GINV\_", s)) +

(get(paste0("HSAV\_", s)) \* get(paste0("HDY\_", s)))]

}

DS\_out <- DS[, .(n, t, base = DS\_base, pol = DS\_pol)]

data.table::setnames(DS\_out, c("base","pol"), c(base\_scn, pol\_scn))

DS\_out <- add\_macroregions\_additive(DS\_out, EU28, scenarios = cfg$scenarios)

DS\_out <- add\_var\_cols(DS\_out, base = base\_scn, pol = pol\_scn)

out[["DS\_t"]] <- DS\_out

}

# --- Household CPI: P\_HH\_CPI\_t ---------------------------------------------

P\_CPI <- rs[["P\_CPI\_t"]]

if (!is.null(P\_CPI) && "au" %in% names(P\_CPI)) {

P\_HH\_CPI <- P\_CPI[au == "CP", c("n","t", base\_scn, pol\_scn), with = FALSE]

P\_HH\_CPI <- add\_var\_cols(P\_HH\_CPI, base = base\_scn, pol = pol\_scn)

out[["P\_HH\_CPI\_t"]] <- P\_HH\_CPI

}

# --- K/L ratios (country & sector) ------------------------------------------

KDT <- rs[["K\_t"]]; LDT <- rs[["L\_t"]]

if (!is.null(KDT) && !is.null(LDT)) {

KDT <- norm\_key\_types(KDT); LDT <- norm\_key\_types(LDT)

keys\_KL <- intersect(intersect(names(KDT), names(LDT)), c("n","i","t"))

KL <- merge(KDT, LDT, by = keys\_KL, all = FALSE, suffixes = c("\_K","\_L"))

for (s in c(base\_scn, pol\_scn)) {

num <- paste0(s, "\_K"); den <- paste0(s, "\_L")

if (all(c(num, den) %in% names(KL))) {

KL[, (s) := fifelse(get(den) == 0 | is.na(get(den)), NA\_real\_, get(num) / get(den))]

}

}

KL <- KL[, c("n", "i", "t", base\_scn, pol\_scn), with = FALSE]

KL <- add\_var\_cols(KL, base = base\_scn, pol = pol\_scn)

out[["KLratio\_t"]] <- KL

# ΣK & ΣL by (n,t), then ratio

K\_cty <- KDT[, lapply(.SD, sum, na.rm=TRUE), .SDcols = c(base\_scn, pol\_scn), by = .(n,t)]

L\_cty <- LDT[, lapply(.SD, sum, na.rm=TRUE), .SDcols = c(base\_scn, pol\_scn), by = .(n,t)]

KL\_cty <- merge(K\_cty, L\_cty, by = c("n","t"), suffixes = c("\_K","\_L"))

KL\_cty[, (base\_scn) := fifelse(get(paste0(base\_scn,"\_L")) == 0, NA\_real\_,

get(paste0(base\_scn,"\_K")) / get(paste0(base\_scn,"\_L")))]

KL\_cty[, (pol\_scn) := fifelse(get(paste0(pol\_scn,"\_L")) == 0, NA\_real\_,

get(paste0(pol\_scn,"\_K")) / get(paste0(pol\_scn,"\_L")))]

KL\_cty[, c(paste0(base\_scn,"\_K"), paste0(pol\_scn,"\_K"),

paste0(base\_scn,"\_L"), paste0(pol\_scn,"\_L")) := NULL]

KL\_cty <- add\_var\_cols(KL\_cty, base = base\_scn, pol = pol\_scn)

out[["KLratio\_country\_t"]] <- KL\_cty

}

# --- Relative price K/L by (n,i,t): P\_KL\_t ---------------------------------

Pinput <- rs[["P\_INPUT\_t"]]

if (!is.null(Pinput) && is.data.table(Pinput) && "oc" %in% names(Pinput)) {

Pinput <- norm\_key\_types(Pinput)

P\_K <- Pinput[oc == "K", c("n","i","t", base\_scn, pol\_scn), with = FALSE]

P\_L <- Pinput[oc == "L", c("n","i","t", base\_scn, pol\_scn), with = FALSE]

PKL <- merge(P\_K, P\_L, by = c("n","i","t"), suffixes = c("\_K","\_L"))

for (s in c(base\_scn, pol\_scn)) {

num <- paste0(s,"\_K"); den <- paste0(s,"\_L")

PKL[, (s) := fifelse(get(den) == 0 | is.na(get(den)), NA\_real\_, get(num)/get(den))]

}

PKL <- PKL[, c("n","i","t", base\_scn, pol\_scn), with = FALSE]

PKL <- add\_var\_cols(PKL, base = base\_scn, pol = pol\_scn)

out[["P\_KL\_t"]] <- PKL

}

# --- Investment by industry: 6-group aggregates ----------------------------

I\_PP <- rs[["I\_PP\_t"]]

if (!is.null(I\_PP) && is.data.table(I\_PP)) {

# ensure wide

if ("scenario" %in% names(I\_PP) && "value" %in% names(I\_PP)) {

I\_PP <- wide\_by\_scenario(I\_PP, scenarios = c(base\_scn, pol\_scn))

}

# select columns by \*names stored in variables\* (data.table way)

sel <- c("n","i","t", base\_scn, pol\_scn)

I\_PP\_sub <- I\_PP[, sel, with = FALSE] # <- this keeps the real column names

# sanity check (optional)

miss <- setdiff(sel, names(I\_PP\_sub))

if (length(miss)) stop("Missing columns: ", paste(miss, collapse = ", "))

I\_PP\_G6 <- add\_sector\_groups\_additive(

I\_PP\_sub,

scenarios = c(base\_scn, pol\_scn),

append\_original = FALSE

)

# add EU macroregions on the country dimension if you want symmetry with others

I\_PP\_G6 <- add\_macroregions\_additive(I\_PP\_G6, EU28, scenarios = c(base\_scn, pol\_scn))

I\_PP\_G6 <- add\_var\_cols(I\_PP\_G6, base = base\_scn, pol = pol\_scn)

out[["I\_PP\_SECT6\_t"]] <- I\_PP\_G6

}

# --- Output composition by 6 groups (REAL terms; shares sum to 100) ----------

# Build gross output in \*quantities\* by industry:

GOq\_by\_i <- NULL

# Preferred: direct quantity table

if (!is.null(rs[["Q\_t"]])) {

GOq\_by\_i <- rs[["Q\_t"]][, c("n","i","t", base\_scn, pol\_scn), with = FALSE]

# Fallback: deflate value by price (GO\_VAL / P\_Q) to get "real" output

} else if (!is.null(rs[["VA\_VAL\_t"]]) && !is.null(rs[["P\_Q\_t"]])) {

V <- rs[["VA\_VAL\_t"]][, c("n","i","t", base\_scn, pol\_scn), with = FALSE]

P <- rs[["P\_Q\_t"]][, c("n","i","t", base\_scn, pol\_scn), with = FALSE]

GOq\_by\_i <- merge(V, P, by = c("n","i","t"), suffixes = c("\_V","\_P"))

for (s in c(base\_scn, pol\_scn)) {

GOq\_by\_i[, (s) := fifelse(get(paste0(s,"\_P")) == 0 | is.na(get(paste0(s,"\_P"))),

NA\_real\_,

get(paste0(s,"\_V")) / get(paste0(s,"\_P")))]

}

GOq\_by\_i <- GOq\_by\_i[, c("n","i","t", base\_scn, pol\_scn), with = FALSE]

}

if (!is.null(GOq\_by\_i)) {

GOq\_by\_i <- norm\_key\_types(GOq\_by\_i)

# 1) Aggregate industries to the 6 groups in \*real\* terms

GO6q <- add\_sector\_groups\_additive(GOq\_by\_i,

scenarios = c(base\_scn, pol\_scn),

append\_original = FALSE)

# 2) Add EU macroregions \*\*before\*\* computing shares (shares are not additive)

GO6q <- add\_macroregions\_additive(GO6q, EU28, scenarios = c(base\_scn, pol\_scn))

# 3) Drop total row; compute shares (×100) within each (n,t)

GO6q <- GO6q[i != "TOT\_G6"]

for (s in c(base\_scn, pol\_scn)) {

GO6q[, (s) := 100 \* get(s) / sum(get(s), na.rm = TRUE), by = .(n, t)]

}

# Optional: consistent group order for plotting

grp\_order <- c("primary","high\_energy\_manufacturing","low\_energy\_manufacturing",

"utilities\_construction","market\_services","public\_personal\_services")

GO6q[, i := factor(as.character(i), levels = grp\_order)]

# 4) Add deltas/% changes on the \*shares\* (base vs policy)

GO6q <- add\_var\_cols(GO6q, base = base\_scn, pol = pol\_scn)

out[["OUT\_COMP6\_SHARE\_REAL\_t"]] <- GO6q[]

} else {

message("• Skipping OUT\_COMP6\_SHARE\_REAL\_t: need either Q\_t or (GO\_VAL\_t & P\_Q\_t).")

}

# --- BITRADE by macro regions (USA split) -----------------------------------

BT\_src <- rs[["BITRADE\_t"]]

if (!is.null(BT\_src) && is.data.table(BT\_src)) {

BT\_src <- norm\_key\_types(data.table::copy(BT\_src))

if ("n" %in% names(BT\_src)) BT\_src <- BT\_src[!(n %chin% c("EU28","NonEU28","WORLD"))]

if ("n1" %in% names(BT\_src)) BT\_src <- BT\_src[!(n1 %chin% c("EU28","NonEU28","WORLD"))]

BT\_src[, n := .region\_map\_vec(n)]

BT\_src[, n1 := .region\_map\_vec(n1)]

BT\_reg <- BT\_src[, lapply(.SD, sum, na.rm = TRUE),

.SDcols = c(base\_scn, pol\_scn), by = .(n, n1, c, t)]

imp\_tot <- BT\_reg[, lapply(.SD, sum), .SDcols = c(base\_scn, pol\_scn), by = .(n, c, t)]

imp\_tot[, n1 := "TOT"]

data.table::setcolorder(imp\_tot, c("n","n1","c","t", base\_scn, pol\_scn))

exp\_tot <- BT\_reg[, lapply(.SD, sum), .SDcols = c(base\_scn, pol\_scn), by = .(n1, c, t)]

exp\_tot[, n := "TOT"]

data.table::setcolorder(exp\_tot, c("n","n1","c","t", base\_scn, pol\_scn))

BT\_reg2 <- data.table::rbindlist(list(BT\_reg, imp\_tot, exp\_tot), use.names = TRUE)

prod\_tot <- BT\_reg2[, lapply(.SD, sum), .SDcols = c(base\_scn, pol\_scn), by = .(n, n1, t)]

prod\_tot[, c := "TOT"]

data.table::setcolorder(prod\_tot, c("n","n1","c","t", base\_scn, pol\_scn))

BT\_reg3 <- data.table::rbindlist(list(BT\_reg2, prod\_tot), use.names = TRUE)

BT\_reg3[, year := 2014L + as.integer(t)]

BT\_reg3 <- add\_var\_cols(BT\_reg3, base = base\_scn, pol = pol\_scn)

order\_levels <- c("EEU","NWEU","SEU","USA","CHN","IND","OECD","NonOECD","ROW","TOT")

BT\_reg3[, n := factor(as.character(n), levels = order\_levels)]

BT\_reg3[, n1 := factor(as.character(n1), levels = order\_levels)]

out[["BITRADE\_REG\_t"]] <- BT\_reg3[]

}

Filter(Negate(is.null), out)

}

### Extract\_registry.R

# ==============================================================================

# ==== extract\_registry ========

# ==============================================================================

# ---------------------------- Helpers --------------------------------

#' Symbols to extract and their key dimensions

#' @export

plan\_extractions <- function(cfg) {

# Build registry directly as a data.table with a list-column for dims

reg <- data.table::data.table(

symbol = c(

# macro (n,t)

"GDPr\_t","HSAVR\_t","HDY\_VAL\_t","GSUR\_VAL\_t","GINV\_VAL\_t","U\_t","ir\_t",

"TB\_t",

# inv & prices (n,i,t)

"Q\_t","I\_PP\_t","P\_I\_t","K\_t","L\_t","P\_Q\_t",

# input prices (n,i,oc,t)

"P\_INPUT\_t",

# CPI (n,au,t)

"P\_CPI\_t",

# trade (quantities + prices)

"USE\_PP\_t","M\_TOT\_t","P\_USE\_t","P\_Mcif\_t","BITRADE\_t",

# emissions

"GHG\_t"

),

dims = list(

# macro

c("n","t"), c("n","t"), c("n","t"), c("n","t"), c("n","t"), c("n","t"),

c("n","t"), c("n","t"),

# inv & prices

c("n","i","t"), c("n","i","t"), c("n","i","t"), c("n","i","t"), c("n","i","t"),

c("n","i","t"),

# input prices

c("n","i","oc","t"),

# CPI

c("n","au","t"),

# trade

c("n","c","au","t"), c("n","c","t"), c("n","c","au","t"), c("n","c","t"),

c("n","n1","c","t"),

# emissions

c("n","i","t")

),

label = c(

"Real GDP","Household saving rate","HH disposable income (val)",

"Gov surplus (val)","Gov investment (val)","Unemployment","Interest rate",

"Trade balance","Output","Investment (PP)","Investment price","Capital","Labor",

"Output price", "Input price (oc)","Consumer price index",

"Use at purchasers' prices","Total imports","Use price",

"Import price (cif)","Bilateral trade", "Emissions"

)

)

# filter by YAML list if provided (use base indexing to avoid scoping issues)

keep <- cfg$extract$include

if (!is.null(keep) && length(keep) > 0L) {

reg <- reg[reg[["symbol"]] %in% keep, ]

}

if (nrow(reg) == 0L) stop("No symbols selected in extract registry (after filtering).")

reg

}

### Helpers\_dt.R

# ==============================================================================

# ==== helpers\_dt.R =====

# ==============================================================================

# data.table-centric helpers: type normalization, pivots, sums, aggregates.

# >>> ensure the numeric measure column is named 'value' -----------------------

norm\_value\_col <- function(DT) {

if (is.null(DT) || nrow(DT) == 0L) return(DT)

vc <- intersect(names(DT), c("value","val","VAL","Val"))

if (length(vc) == 1L && vc != "value") data.table::setnames(DT, vc, "value")

DT

}

# >>> coerce common key columns to stable types (chars) and t to numeric -------

norm\_key\_types <- function(DT) {

if (is.null(DT)) return(DT)

if (!data.table::is.data.table(DT)) DT <- data.table::as.data.table(DT)

for (k in intersect(names(DT), c("n","i","c","em","au"))) DT[, (k) := as.character(get(k))]

if ("t" %in% names(DT)) DT[, t := as.numeric(t)]

DT

}

# >>> add delta (pol - base) and pct ((pol/base)-1) and reorder columns --------

add\_var\_cols <- function(DT, base = "baseline", pol = "ff55") {

if (is.null(DT)) return(NULL)

DT <- norm\_key\_types(DT)

if (!all(c(base, pol) %in% names(DT))) return(DT)

DT[, `:=`(

delta = get(pol) - get(base),

pct = data.table::fifelse(is.na(get(base)) | get(base) == 0, NA\_real\_, (get(pol)/get(base)) - 1)

)]

key\_cols <- setdiff(names(DT), c(base, pol, "delta", "pct"))

data.table::setcolorder(DT, c(key\_cols, base, pol, "delta", "pct"))

DT[]

}

# >>> compute value = price \* quantity, then sum by keys for given scenarios----

sum\_value\_by <- function(priceDT, qtyDT, by = c("n","t"), filter = NULL,

scenarios = c("baseline","ff55")) {

if (is.null(priceDT) || is.null(qtyDT)) return(NULL)

P <- norm\_key\_types(data.table::copy(priceDT))

Q <- norm\_key\_types(data.table::copy(qtyDT))

if (!is.null(filter)) {

for (nm in names(filter)) {

if (nm %in% names(P)) P <- P[get(nm) %in% filter[[nm]]]

if (nm %in% names(Q)) Q <- Q[get(nm) %in% filter[[nm]]]

}

}

scn\_cols <- c("baseline","ff55","delta","pct")

join\_keys <- intersect(setdiff(names(P), scn\_cols), setdiff(names(Q), scn\_cols))

if (!all(by %in% join\_keys)) by <- intersect(by, join\_keys)

if (length(join\_keys) == 0L) return(NULL)

scenarios <- intersect(scenarios, intersect(names(P), names(Q)))

if (length(scenarios) == 0L) return(NULL)

data.table::setnames(P, scenarios, paste0("P\_", scenarios))

data.table::setnames(Q, scenarios, paste0("Q\_", scenarios))

data.table::setkeyv(P, join\_keys); data.table::setkeyv(Q, join\_keys)

M <- merge(P, Q, by = join\_keys, allow.cartesian = TRUE)

if (nrow(M) == 0L) return(NULL)

if ("t" %in% names(M)) M[, t := as.numeric(t)]

for (s in scenarios) M[, (s) := get(paste0("P\_", s)) \* get(paste0("Q\_", s))]

out <- M[, lapply(.SD, sum, na.rm = TRUE), .SDcols = scenarios, by = by]

out[]

}

# >>> add EU28, NonEU28,and WORLD rows by summing additive variables over'n'----

add\_macroregions\_additive <- function(DT, eu\_members,

scenarios = c("baseline","ff55")) {

if (is.null(DT) || !"n" %in% names(DT)) return(DT)

DT <- norm\_key\_types(DT)

base\_rows <- DT[!(n %in% c("EU28","NonEU28","WORLD"))]

key\_cols <- setdiff(names(base\_rows), c(scenarios, "delta","pct"))

by\_no\_n <- setdiff(key\_cols, "n")

if (length(by\_no\_n) == length(key\_cols)) by\_no\_n <- key\_cols # safety fallback

agg\_subset <- function(sub, tag) {

if (nrow(sub) == 0L) return(NULL)

A <- sub[, lapply(.SD, sum, na.rm=TRUE), .SDcols = scenarios, by = by\_no\_n]

A[, n := tag]

data.table::setcolorder(A, c("n", by\_no\_n, scenarios))

A[]

}

EU <- agg\_subset(base\_rows[n %chin% eu\_members], "EU28")

NonEU <- agg\_subset(base\_rows[!n %chin% eu\_members], "NonEU28")

WLD <- data.table::rbindlist(list(EU, NonEU), use.names = TRUE, fill = TRUE)

if (!is.null(WLD)) {

WLD <- WLD[, lapply(.SD, sum, na.rm=TRUE), .SDcols = scenarios, by = by\_no\_n]

WLD[, n := "WORLD"]

data.table::setcolorder(WLD, c("n", by\_no\_n, scenarios))

}

out <- data.table::rbindlist(list(base\_rows, EU, NonEU, WLD), use.names = TRUE, fill = TRUE)

out[]

}

# >>> pivot a long table with 'scenario' into a wide table with one col per scn----

wide\_by\_scenario <- function(DT, scenarios) {

if (is.null(DT) || !nrow(DT)) return(DT)

DT <- norm\_key\_types(DT)

dims <- setdiff(names(DT), c("scenario","value"))

if (!length(dims)) stop("No key dimensions found to pivot.")

w <- data.table::dcast(DT, as.formula(paste(paste(dims, collapse = "+"), "~ scenario")),

value.var = "value", fill = NA\_real\_)

for (s in scenarios) if (!s %in% names(w)) w[, (s) := NA\_real\_]

data.table::setcolorder(w, c(dims, scenarios[scenarios %in% names(w)]))

w[]

}

### Helpers\_utils.R

# ==============================================================================

# ==== helpers\_utils.R ======

# ==============================================================================

# General-purpose helpers: logging, project paths, and safe lookups.

# >>> print a timestamped message to the console--------------------------------

log\_time <- function(msg) {

message(sprintf("[%s] %s", format(Sys.time(), "%Y-%m-%d %H:%M:%S"), msg))

}

# >>> find the project root ----------------------------------------------------

project\_root <- function(start = getwd()) {

cur <- normalizePath(start, winslash = "/", mustWork = FALSE)

repeat {

if (file.exists(file.path(cur, "DESCRIPTION")) ||

length(list.files(cur, pattern = "\\.Rproj$", all.files = TRUE, no.. = TRUE)) > 0) {

return(cur)

}

parent <- dirname(cur)

if (identical(parent, cur)) break

cur <- parent

}

normalizePath(start, winslash = "/", mustWork = FALSE)

}

# >>> resolve a relative path against the project root -------------------------

proj\_path <- function(...) {

root <- project\_root()

normalizePath(file.path(root, ...), winslash = "/", mustWork = FALSE)

}

# >>> print basic runtime info -------------------------------------------------

print\_runtime\_info <- function(cfg) {

root <- project\_root()

log\_time(paste("Project root:", root))

log\_time(paste("GDX dir :", cfg$paths$gdx\_dir))

log\_time(paste("Outputs dir :", cfg$paths$outputs))

}

# >>> fetch a symbol (table) from a list or return NULL ------------------------

require\_symbol <- function(raw, name, require\_cols = NULL, min\_rows = 1L, quiet = FALSE) {

if (!is.list(raw) || is.null(raw[[name]])) {

if (!quiet) message("• Skipping derived: missing base symbol '", name, "'.")

return(NULL)

}

DT <- raw[[name]]

if (is.null(DT) || nrow(DT) < min\_rows) {

if (!quiet) message("• Skipping derived: symbol '", name, "' is empty.")

return(NULL)

}

if (!is.null(require\_cols)) {

miss <- setdiff(require\_cols, names(DT))

if (length(miss)) {

if (!quiet) message("• Skipping '", name, "': missing cols {", paste(miss, collapse = ", "), "}.")

return(NULL)

}

}

DT

}

### IO\_gdx.R

# ==============================================================================

# ==== io\_gdx ========

# ==============================================================================

# ---- Paths & openers ----

# Build the path to a scenario's GDX file

gdx\_path\_for <- function(cfg, scenario) {

file.path(cfg$paths$gdx\_dir, paste0("results\_all\_", scenario, ".gdx"))

}

# Open a GDX using gdxtools (object used by extract\_param)

open\_gdx <- function(cfg, scenario) {

fp <- gdx\_path\_for(cfg, scenario)

if (!file.exists(fp)) stop("GDX not found: ", fp)

if (!requireNamespace("gdxtools", quietly = TRUE)) {

stop("Package 'gdxtools' is required. Install with:\n",

"remotes::install\_github('lolow/gdxtools')")

}

gdxtools::gdx(fp)

}

# ---- utility: check if a symbol exists in the GDX ----

symbol\_exists <- function(gdx\_obj, name) {

nm <- unique(c(gdx\_obj$variables$name, gdx\_obj$parameters$name))

isTRUE(name %in% nm)

}

# ---- low-level extraction (unchanged if gdxtools is your backend) ----

extract\_param <- function(gdx\_obj, name) {

name <- as.character(name)[1L] # <<< ensure scalar

dt <- data.table::as.data.table(gdxtools::extract(gdx\_obj, name))

dt <- norm\_value\_col(dt)

dt <- norm\_key\_types(dt)

dt

}

# ---- one symbol across ALL scenarios (make sym scalar + existence check) ----

extract\_symbol\_all <- function(cfg, reg\_row) {

sym <- as.character(reg\_row[["symbol"]][1L]) # <<< force single name

dims <- reg\_row[["dims"]][[1L]]

lst <- lapply(cfg$scenarios, function(scn) {

gdx\_obj <- open\_gdx(cfg, scn)

if (!symbol\_exists(gdx\_obj, sym)) {

message("Skipping '", sym, "' in scenario '", scn, "' (not found in GDX).")

return(NULL)

}

dt <- extract\_param(gdx\_obj, sym)

if (!all(dims %in% names(dt))) {

stop("Symbol ", sym, " missing expected dims: ",

paste(setdiff(dims, names(dt)), collapse = ", "))

}

data.table::setDT(dt)

data.table::setkeyv(dt, dims)

dt[, scenario := scn]

dt

})

lst <- Filter(Negate(is.null), lst)

if (!length(lst)) return(data.table::data.table()) # empty if missing in all scenarios

out <- data.table::rbindlist(lst, use.names = TRUE, fill = TRUE)

data.table::setkeyv(out, c(dims, "scenario"))

out

}

#' Extract all selected symbols for all scenarios (long format)

#' @export

extract\_all <- function(cfg, reg) {

res <- lapply(seq\_len(nrow(reg)), function(i) extract\_symbol\_all(cfg, reg[i]))

names(res) <- reg$symbol

res

}

list\_gdx\_names <- function(cfg, scenario) {

g <- open\_gdx(cfg, scenario)

sort(unique(c(g$parameters$name, g$variables$name)))

}

### Launch\_app.R

#' Launch one of the packaged Shiny apps

#' @param app "diagnostic" or "results"

#' @param outputs\_dir Optional absolute path to outputs/derived.

#' @param project\_root Optional project root; if set we read config/project.yml there.

#' @param config\_path Optional explicit path to config/project.yml.

#' @export

launch\_app <- function(app = c("diagnostic","results"),

outputs\_dir = NULL,

project\_root = NULL,

config\_path = NULL) {

app <- match.arg(app)

app\_dir <- system.file("app", app, package = utils::packageName())

if (!nzchar(app\_dir)) stop("App not found in this package: ", app)

# Resolve outputs path (no hard-coding needed)

if (is.null(outputs\_dir)) {

outputs\_dir <- tryCatch(

outputs\_dir\_from\_config(root = project\_root, config\_path = config\_path),

error = function(e) {

# Fallback to option/env var if config discovery fails

getOption("fidelio.outputs", Sys.getenv("FIDELIO\_OUTPUTS", ""))

}

)

}

if (!nzchar(outputs\_dir) || !dir.exists(outputs\_dir)) {

stop(

"Could not resolve outputs/derived.\n",

"Try one of:\n",

" - launch\_app(app, project\_root = '...') # use config/project.yml\n",

" - launch\_app(app, config\_path = '.../config/project.yml')\n",

" - launch\_app(app, outputs\_dir = '.../outputs/derived')\n",

" - options(fidelio.project = '...') or options(fidelio.outputs = '...')\n",

" - set FIDELIO\_PROJECT or FIDELIO\_OUTPUTS environment variables."

)

}

# Pass to the app via an option the loaders already read

options(fidelio.outputs = normalizePath(outputs\_dir, winslash = "/", mustWork = TRUE))

on.exit(options(fidelio.outputs = NULL), add = TRUE)

shiny::runApp(app\_dir, display.mode = "normal")

}

### packages.R

# ==============================================================================

# ==== packages ========

# ==============================================================================

# Package-wide options (safe and lightweight)

.onLoad <- function(libname, pkgname) {

# Respect user env var; otherwise let data.table decide

if (nzchar(Sys.getenv("FIDELIO\_DT\_THREADS"))) {

data.table::setDTthreads(as.integer(Sys.getenv("FIDELIO\_DT\_THREADS")))

}

# Consistent printing (feel free to tweak)

options(

datatable.print.nrows = 200L,

datatable.print.topn = 5L,

datatable.print.class = TRUE

)

}

# Optional: soft check for suggested packages in interactive Shiny sessions

check\_suggested <- function(pkgs) {

missing <- pkgs[!vapply(pkgs, requireNamespace, logical(1), quietly = TRUE)]

if (length(missing) && interactive()) {

msg <- paste0(

"Missing suggested packages: ", paste(missing, collapse = ", "),

". Install with install.packages() (or dev/install\_deps.R)."

)

message(msg)

}

invisible(missing)

}

### pipeline.R

# ==============================================================================

# ==== pipeline ========

# ==============================================================================

#' Run the end-to-end pipeline: extract → wide → aggregates → deltas → derived → save

#' @export

run\_pipeline <- function(config = "config/project.yml") {

cfg <- if (is.list(config)) config else load\_config()

print\_runtime\_info(cfg)

# 1) what to extract

reg <- plan\_extractions(cfg)

# 2) extract long (dims + scenario + value)

raw <- extract\_all(cfg, reg)

# 3) wide with aggregates + Δ/% (this reproduces your results\_by\_symbol)

additive\_syms <- tryCatch(cfg$aggregations$additive\_symbols, error = function(e) NULL)

if (is.null(additive\_syms)) {

additive\_syms <- c(

"GDPr\_t","HDY\_VAL\_t","GSUR\_VAL\_t","GINV\_VAL\_t",

"Q\_t","I\_PP\_t","K\_t","L\_t","U\_t",

"USE\_PP\_t","M\_TOT\_t","BITRADE\_t","GHG\_t",

"P\_USE\_t","P\_Mcif\_t","P\_I\_t","P\_Q\_t","P\_INPUT\_t"

)

}

base\_scn <- cfg$scenarios[1]; pol\_scn <- cfg$scenarios[length(cfg$scenarios)]

postproc <- function(dt, name) {

if (is.null(dt) || !nrow(dt)) return(NULL)

w <- wide\_by\_scenario(dt, cfg$scenarios)

if ("n" %in% names(w) && name %in% additive\_syms && !is.null(cfg$groups$EU28)) {

w <- add\_macroregions\_additive(w, eu\_members = cfg$groups$EU28, scenarios = cfg$scenarios)

}

add\_var\_cols(w, base = base\_scn, pol = pol\_scn)

}

results\_by\_symbol <- lapply(names(raw), function(sym) postproc(raw[[sym]], sym))

names(results\_by\_symbol) <- names(raw)

# 4) your “derived from wide” block

derived <- derive\_from\_wide(results\_by\_symbol, cfg)

# ---------- Generic promoter: base -> derived (config driven) ----------

# Coerce to derived (wide) shape when needed

ensure\_derived\_shape <- function(DT, key\_cols = NULL) {

DT <- data.table::as.data.table(DT)

# auto-detect keys if not given

if (is.null(key\_cols)) {

key\_cols <- intersect(names(DT), c("n","n1","i","c","au","oc","t"))

if (!length(key\_cols)) key\_cols <- character(0)

}

# A) already wide (baseline/ff55 present)

if (all(c("baseline","ff55") %in% names(DT))) {

if (!"delta" %in% names(DT)) DT[, delta := ff55 - baseline]

if (!"pct" %in% names(DT)) DT[, pct := data.table::fifelse(abs(baseline) > .Machine$double.eps,

delta / baseline, NA\_real\_)]

return(DT[])

}

# B) long (scenario/value) -> promote to wide

need <- c(key\_cols, "scenario", "value")

if (all(need %in% names(DT))) {

W <- data.table::dcast(DT,

as.formula(paste(paste(key\_cols, collapse = " + "), "~ scenario")),

value.var = "value")

# Normalize scenario names if necessary

sc\_other <- setdiff(names(W), c(key\_cols, "delta","pct"))

if (!all(c("baseline","ff55") %in% names(W)) && length(sc\_other) >= 2)

data.table::setnames(W, sc\_other[1:2], c("baseline","ff55"))

W <- W[!is.na(baseline) & !is.na(ff55)]

W[, delta := ff55 - baseline]

W[, pct := data.table::fifelse(abs(baseline) > .Machine$double.eps,

delta / baseline, NA\_real\_)]

return(W[])

}

# otherwise leave as-is

DT[]

}

# Read config lists (with safe defaults)

incl <- try(cfg$derive$include\_from\_base, silent = TRUE)

if (inherits(incl, "try-error") || is.null(incl)) incl <- character(0)

key\_hints <- try(cfg$derive$keys, silent = TRUE)

if (inherits(key\_hints, "try-error") || is.null(key\_hints)) key\_hints <- list()

# Promote each requested base symbol into 'derived' if not already there

for (nm in incl) {

if (nm %in% names(derived)) next

if (!(nm %in% names(results\_by\_symbol))) next # not in base

D <- data.table::copy(results\_by\_symbol[[nm]])

# add calendar year if useful

if (!"year" %in% names(D) && "t" %in% names(D)) D[, year := 2014L + as.integer(t)]

keys <- key\_hints[[nm]] %||% NULL

Dd <- ensure\_derived\_shape(D, key\_cols = keys)

if (!is.null(Dd) && nrow(Dd)) derived[[nm]] <- Dd

}

# 5) save

save\_artifacts(results\_by\_symbol, cfg, subdir = "base")

if (length(derived)) save\_artifacts(derived, cfg, subdir = "derived")

message("Built results\_by\_symbol: ", paste(names(results\_by\_symbol), collapse = ", "))

if (length(derived)) message("Built derived: ", paste(names(derived), collapse = ", "))

invisible(list(cfg = cfg, raw = raw, results\_by\_symbol = results\_by\_symbol, derived = derived))

}

### R2csv.R

# ==============================================================================

# ==== R2csv.R =====

# ==============================================================================

# Convert the 'results\_app' bundle into a long CSV with a PyPSA-like

# variable column and an extra pct\_change column.

#' Export bundled results to a template CSV (PyPSA-like)

#'

#' @param cfg list from load\_config() or a path to config YAML. If omitted,

#' the function will call load\_config() for you (robust to empty default.yml).

#' @param bundle\_name name of the saved bundle to read (default "results\_app").

#' @param out\_basename file stem for the CSV (default "results\_bundle\_template").

#' @param model\_name value for the `model` column (default "FIDELIO").

#' @param pct\_as\_percent if TRUE, multiplies pct by 100.

#' @param include\_dim\_names if TRUE, encode dims as `i=C24|n1=USA`; if FALSE, just `C24|USA`.

#' @param default\_unit default unit string for all symbols except special cases.

#' @param unit\_overrides named list mapping symbol -> unit string (wins over defaults).

#' @param csv\_sep field separator passed to fwrite (use ";" for EU Excel).

#' @export

export\_results\_csv <- function(cfg = NULL,

bundle\_name = "results\_app",

out\_basename = "results\_bundle\_template",

model\_name = "FIDELIO",

pct\_as\_percent = TRUE,

include\_dim\_names = TRUE,

default\_unit = "million EUR",

unit\_overrides = list(),

csv\_sep = ",") {

# ---- helpers from the package namespace ----

outputs\_dir\_from\_config <- fidelioDiagnostics:::outputs\_dir\_from\_config

load\_config <- fidelioDiagnostics:::load\_config

`%||%` <- function(x, y) if (is.null(x)) y else x

# ---- config (robust) ----

if (is.null(cfg) || is.character(cfg)) {

cfg <- tryCatch(load\_config(), error = function(e) NULL)

if (is.null(cfg)) {

cfg <- list(paths = list(outputs = normalizePath("outputs", mustWork = FALSE)),

scenarios = c("baseline","ff55"))

}

}

# Prefer explicit path from cfg; fall back to helper if needed

outdir <- tryCatch(file.path(cfg$paths$outputs, "derived"),

error = function(e) NA\_character\_)

if (is.na(outdir) || is.null(outdir) || !nzchar(outdir)) {

outdir <- outputs\_dir\_from\_config()

}

if (!dir.exists(outdir)) dir.create(outdir, recursive = TRUE, showWarnings = FALSE)

# ---- load bundle ----

bundle\_path <- file.path(outdir, paste0("bundle\_", bundle\_name, ".rds"))

if (!file.exists(bundle\_path)) {

stop("Bundle not found: ", bundle\_path,

"\nMake sure your pipeline built save/bundles$", bundle\_name, " first.")

}

b <- readRDS(bundle\_path)

if (!length(b)) stop("Bundle '", bundle\_name, "' is empty.")

# ---- units (override > special > default; never blank) ----

special\_units <- c(

"TB\_GDP\_t" = "ratio",

"OUT\_COMP6\_SHARE\_REAL\_t" = "share"

)

unit\_of <- function(symbol) {

u <- unit\_overrides[[symbol]]

if (!is.null(u) && nzchar(u)) return(u)

if (!is.null(special\_units[[symbol]])) return(special\_units[[symbol]])

default\_unit

}

# ---- dims and scenarios ----

dim\_priority <- c("i","c","n1","au","oc") # 'n' becomes region

scn\_cfg <- cfg$scenarios %||% c("baseline","ff55")

# ---- normalize wide -> long (scenario) ----

to\_long\_scn <- function(DT) {

data.table::setDT(DT)

meas\_wide <- intersect(names(DT), c(scn\_cfg, "baseline", "ff55"))

if (length(meas\_wide) >= 1L) {

id\_cols <- setdiff(names(DT), c(meas\_wide, "delta", "pct"))

L <- data.table::melt(DT, id.vars = id\_cols,

measure.vars = meas\_wide,

variable.name = "scenario",

value.name = "value")

} else if (all(c("scenario","value") %in% names(DT))) {

L <- data.table::copy(DT)

} else {

stop("Cannot normalize table; expected either wide (scenario columns) or long (scenario/value).")

}

if ("t" %in% names(L) && !"year" %in% names(L)) {

L[, year := 2014L + as.integer(t)]

}

L[]

}

# ---- attach pct\_change from wide 'pct' if present ----

attach\_pct <- function(W, L) {

if (!("pct" %in% names(W))) {

L[, pct\_change := NA\_real\_]

return(L[])

}

key\_cols <- intersect(names(W), c("n","n1","i","c","au","oc","t"))

X <- data.table::copy(W[, c(key\_cols, "pct"), with = FALSE])

if ("t" %in% names(X) && !"year" %in% names(X)) X[, year := 2014L + as.integer(t)]

L2 <- merge(L, X, by = intersect(names(L), names(X)), all.x = TRUE)

base\_name <- scn\_cfg[1] %||% "baseline"

if (isTRUE(pct\_as\_percent) && "pct" %in% names(L2)) L2[, pct := 100 \* pct]

L2[, pct\_change := ifelse(scenario == base\_name, NA\_real\_, pct)]

L2[, pct := NULL]

L2[]

}

# ---- build PyPSA-like variable string ----

make\_var <- function(symbol, DT) {

base <- sub("\_t$", "", symbol)

if (!nzchar(base)) base <- as.character(symbol)

parts <- character(0)

for (d in dim\_priority) if (d %in% names(DT)) {

vals <- as.character(DT[[d]])

lbl <- if (isTRUE(include\_dim\_names)) paste0(d, "=", vals) else vals

parts <- if (length(parts)) paste(parts, lbl, sep = "|") else lbl

}

if (!length(parts)) base else paste(base, parts, sep = "|")

}

# ---- walk bundle, normalize and stack ----

rows <- list()

for (sym in names(b)) {

W <- data.table::as.data.table(b[[sym]])

if (!nrow(W)) next

L <- to\_long\_scn(W)

if (!"n" %in% names(L)) L[, n := NA\_character\_]

L[, region := as.character(n)]

L <- attach\_pct(W, L)

# variable and unit

L[, variable := make\_var(sym, L)]

u <- unit\_of(sym)

if (is.null(u) || is.na(u) || !nzchar(u)) u <- default\_unit

L[, unit := u]

keep <- c("scenario","region","variable","unit","year","value","pct\_change")

for (k in keep) if (!k %in% names(L)) L[, (k) := NA]

L <- L[, keep, with = FALSE]

L[, model := model\_name]

data.table::setcolorder(L, c("model", keep))

rows[[sym]] <- L[]

}

ALL <- data.table::rbindlist(rows, use.names = TRUE, fill = TRUE)

data.table::setorder(ALL, variable, region, scenario, year)

# ---- write CSV ----

out\_file <- file.path(outdir, paste0(out\_basename, ".csv"))

data.table::fwrite(ALL, out\_file, sep = csv\_sep)

message("Wrote template CSV: ", out\_file)

invisible(out\_file)

}

### Save\_export.R

# ==============================================================================

# ==== save\_export ========

# ==============================================================================

#' Save a named list of data.tables according to config

#' @export

save\_artifacts <- function(objs, cfg, subdir = NULL) {

outdir <- cfg$paths$outputs

if (!is.null(subdir)) outdir <- file.path(outdir, subdir)

dir.create(outdir, recursive = TRUE, showWarnings = FALSE)

fmts <- cfg$save$formats

if (is.null(fmts) || !length(fmts)) fmts <- "csv"

for (nm in names(objs)) {

dt <- objs[[nm]]

if (is.null(dt) || !nrow(dt)) next

base <- file.path(outdir, nm)

if ("feather" %in% fmts) {

arrow::write\_feather(dt, paste0(base, ".feather"))

}

if ("parquet" %in% fmts) {

arrow::write\_parquet(dt, paste0(base, ".parquet"))

}

if ("fst" %in% fmts) {

fst::write\_fst(dt, paste0(base, ".fst"))

}

if ("csv" %in% fmts) {

data.table::fwrite(dt, paste0(base, ".csv"))

}

if ("rds" %in% fmts) {

saveRDS(dt, paste0(base, ".rds"))

}

}

man <- make\_manifest(objs, outdir) # index for lazy loading in Shiny

make\_bundles(objs, outdir, cfg$save$bundles) # selective bundles + combined CSVs

invisible(outdir)

}

# --- build a simple manifest (symbol, format, path, nrows) --------------------

make\_manifest <- function(objs, outdir) {

rows <- lapply(names(objs), function(nm) {

cand <- c("parquet","feather","fst","rds","csv")

files <- file.path(outdir, paste0(nm, ".", cand))

keep <- file.exists(files)

if (!any(keep)) return(NULL)

data.table::data.table(

symbol = nm,

format = cand[keep],

path = files[keep],

nrows = nrow(objs[[nm]])

)

})

man <- data.table::rbindlist(rows, use.names = TRUE, fill = TRUE)

if (!is.null(man) && nrow(man)) {

data.table::fwrite(man, file.path(outdir, "manifest.csv"))

saveRDS(man, file.path(outdir, "manifest.rds"))

}

man

}

# --- apply symbol-specific filters (keep/drop lists by column) ----------------

apply\_filters <- function(DT, filt) {

if (is.null(filt) || !length(filt)) return(DT)

D <- data.table::copy(DT)

if (!is.null(filt$keep)) {

for (nm in names(filt$keep)) if (nm %in% names(D))

D <- D[get(nm) %chin% filt$keep[[nm]]]

}

if (!is.null(filt$drop)) {

for (nm in names(filt$drop)) if (nm %in% names(D))

D <- D[!(get(nm) %chin% filt$drop[[nm]])]

}

D[]

}

# --- normalize any table for a combined CSV (pad missing dims; add 'symbol') --

norm\_for\_csv <- function(DT, symbol, prefer = c("baseline","ff55","delta","pct")) {

dims\_all <- c("n","n1","i","c","au","oc","t")

D <- data.table::as.data.table(DT)

# make sure these columns exist (else add as NA)

for (k in dims\_all) if (!k %in% names(D)) D[, (k) := NA]

# ensure preferred measure columns exist if present in any DTs

meas <- intersect(prefer, names(D))

D[, symbol := symbol]

# order columns: symbol, dims, then measures present

data.table::setcolorder(D, c("symbol", dims\_all, meas))

D[]

}

# --- write selective bundles (and optional combined CSV) ----------------------

make\_bundles <- function(objs, outdir, bundles\_spec) {

if (is.null(bundles\_spec) || !length(bundles\_spec)) return(invisible(NULL))

for (bn in names(bundles\_spec)) {

spec <- bundles\_spec[[bn]]

inc <- spec$include %||% character(0) # small infix helper below

sel <- intersect(inc, names(objs))

if (!length(sel)) next

# apply symbol-specific filters (if any)

filtered <- lapply(sel, function(nm) {

DT <- objs[[nm]]

filt <- spec$filters[[nm]]

apply\_filters(DT, filt)

})

names(filtered) <- sel

# save RDS bundle

saveRDS(filtered, file.path(outdir, paste0("bundle\_", bn, ".rds")))

# optional combined CSV (wide or long)

if (isTRUE(spec$csv\_combine)) {

shape <- tolower(spec$csv\_shape %||% "wide")

if (shape == "long" && all(vapply(filtered, function(x) all(c("scenario","value") %in% names(x)), TRUE))) {

# already in long; just pad + stack

pieces <- mapply(norm\_for\_csv, filtered, names(filtered), SIMPLIFY = FALSE)

} else {

# wide (baseline/ff55/delta/pct); pad dims + stack

pieces <- mapply(norm\_for\_csv, filtered, names(filtered), SIMPLIFY = FALSE)

}

all\_dt <- data.table::rbindlist(pieces, use.names = TRUE, fill = TRUE)

data.table::fwrite(all\_dt, file.path(outdir, paste0(spec$csv\_basename %||% bn, ".csv")))

}

}

invisible(NULL)

}

`%||%` <- function(x, y) if (is.null(x)) y else x