# fidelioDiagnostic

## config

### default.yml

paths:

gdx\_dir: "data-raw/gdx"

outputs: "outputs"

cache: "cache"

scenarios: ["baseline","ff55"]

save:

bundles:

results\_app: true

### project.yml

project:

id: "ff55\_3scn" # short, file-system-friendly

paths:

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

outputs: "outputs"

scenarios:

- baseline

- ff55

- ff55\_2

# add more later

# - cbam

# - nze

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:

# optional extra per-table formats (keep empty if you don't want them)

formats: [] # e.g., ["csv","fst","rds"]

# Write ONE consolidated list RDS with all variables (results\_by\_symbol):

list\_rds:

enabled: true

basename: "results\_by\_symbol" # versioned + \_latest will be written

# Selective bundles (each becomes bundle\_<name>\_\*.rds and \_latest.rds)

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

derive:

# list base symbols to promote to 'derived' (wide) if not already present there

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

# keys:

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

export\_csv:

enabled: true

bundle: "results\_app" # which bundle to export to CSV

out\_basename: "results\_bundle\_template" # outputs/derived/<name>.csv

model\_name: "FIDELIO"

pct\_as\_percent: true

include\_dim\_names: false

unit\_overrides:

GDPr\_t: "real (base=2014)"

TB\_GDP\_t: "ratio"

I\_PP\_SECT6\_t: "PP units"

## Inst

### App/diagnostic/app.R

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

# fidelioDiagnostics — Diagnostic app (multi-scenario ready)

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

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

# scenario helpers

base\_scn <- fidelioDiagnostics:::base\_scn

policy\_scns <- fidelioDiagnostics:::policy\_scns

scenario\_cols <- fidelioDiagnostics:::scenario\_cols

load\_config <- fidelioDiagnostics:::load\_config

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

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

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

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

# Config (project id in title)

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

.cfg <- try(load\_config(), silent = TRUE)

.project\_id <- tryCatch({

if (!inherits(.cfg, "try-error") && !is.null(.cfg$project$id) && nzchar(.cfg$project$id)) .cfg$project$id else NULL

}, error = function(e) NULL)

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

# App constants (set per app file)

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

APP\_KIND <- "diagnostic\_app" # or "results\_app"

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

# Bundle picker (by project id)

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

.pick\_bundle\_for\_project <- function(dir, project\_id, app\_kind = APP\_KIND) {

p1 <- sprintf("^bundle\_%s\_%s\_.\*\\.(rds|qs)$", app\_kind, project\_id)

p2 <- sprintf("^bundle\_.\*\_%s\_.\*\\.(rds|qs)$", project\_id)

p3 <- sprintf("^results\_bundle\_%s\_.\*\\.(rds|qs)$", project\_id)

fs <- list.files(dir, full.names = TRUE)

cand <- fs[grepl(p1, basename(fs), ignore.case = TRUE)]

if (!length(cand)) cand <- fs[grepl(p2, basename(fs), ignore.case = TRUE)]

if (!length(cand)) cand <- fs[grepl(p3, basename(fs), ignore.case = TRUE)]

if (!length(cand)) return(NULL)

cand <- cand[order(file.info(cand)$mtime, decreasing = TRUE)]

tools::file\_path\_sans\_ext(basename(cand[1]))

}

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

# Data loading — auto-detect bundle by project id, backfill per-symbol

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

get\_results <- function() {

cfg <- load\_config()

dir <- tryCatch(outputs\_dir\_from\_config(cfg), error = function(e) NULL)

if (is.null(dir) || !dir.exists(dir)) dir <- file.path("outputs", "derived")

# Symbols used in diagnostic views (nation, industry, bilateral)

wanted <- c(

# Nation level

"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",

# Industry level

"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",

# Bilateral

"BITRADE\_REG\_t"

)

cat("\n[", APP\_KIND, "] outputs dir:", normalizePath(dir, winslash = "/"), "\n", sep = "")

# 1) Pick bundle for this project id

project\_id <- cfg$project$id %||% ""

bundle\_name <- if (nzchar(project\_id)) .pick\_bundle\_for\_project(dir, project\_id) else NULL

if (!is.null(bundle\_name)) cat("[", APP\_KIND, "] Using bundle: ", bundle\_name, "\n", sep = "")

# 2) Try bundles in order (picked one, then APP\_KIND)

try\_names <- unique(na.omit(c(bundle\_name, APP\_KIND)))

b <- list()

for (nm in try\_names) {

tmp <- try(load\_bundle(nm, dir = dir), silent = TRUE)

if (!inherits(tmp, "try-error") && length(tmp)) { b <- tmp; break }

}

if (!length(b)) {

cat("[", APP\_KIND, "] No bundle loaded. Will backfill per-symbol files.\n", sep = "")

b <- list()

}

# 3) Backfill per-symbol for anything missing

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

if (length(missing)) {

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

for (s in missing) {

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

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

}

}

# 4) Drop NULL/empty & report

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

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

miss <- setdiff(wanted, have)

cat("[", APP\_KIND, "] Loaded symbols: ", paste(have, collapse = ", "), "\n", sep = "")

if (length(miss)) cat("[", APP\_KIND, "] Missing or empty: ", paste(miss, collapse = ", "), "\n", sep = "")

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, descriptions)

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

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

# Nation level

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","HH disposable income (nominal)","HH disposable income (real)","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","CPI (households)"),

group = "Nation level",

desc = c(

"Real GDP at constant prices.",

"Nominal disposable income of households.",

"Real disposable income of households.",

"Economy-wide gross investment.",

"Domestic savings (national).",

"Foreign savings (net capital inflow).",

"Government budget surplus (nominal).",

"Public investment expenditure.",

"Real trade balance (exports − imports).",

"Trade balance divided by GDP (dimensionless).",

"Household saving rate (share of income).",

"Unemployment rate.",

"Country-level capital to labor ratio.",

"Short-term policy interest rate (model).",

"Household consumer price index."

),

keep = TRUE

),

# Industry level

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",

desc = c(

"Gross investment by sector.",

"Sectoral demand for capital services.",

"Sectoral labor demand.",

"Greenhouse gas emissions by sector.",

"Capital to labor ratio at sector level.",

"Sectoral output price index.",

"Price of the composite K–L nest.",

"Gross investment aggregated into six sector groups.",

"Share of real output by six sector groups (sums to 1)."

),

keep = TRUE

),

# Bilateral trade

data.table(

symbol = "BITRADE\_REG\_t",

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

group = "Bilateral trade",

desc = "Bilateral trade matrix across macro regions (origin n → destination n1).",

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]

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

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

# Helpers

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

`%||%` <- function(x, y) if (is.null(x)) y else x

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\_levels <- function(DT, scenarios = NULL) {

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

if (is.null(scenarios)) scenarios <- scenario\_cols(DT)

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

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

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

}

# Build long Δ/% using delta\_<pol> / pct\_<pol> columns

to\_long\_var <- function(DT, use\_pct = TRUE, only\_policies = NULL) {

cfg <- .cfg

b <- base\_scn(cfg)

pols <- policy\_scns(cfg)

if (!is.null(only\_policies)) pols <- intersect(pols, only\_policies)

if (length(pols) == 0L) return(NULL)

suffix <- if (isTRUE(use\_pct)) "pct\_" else "delta\_"

cols <- paste0(suffix, pols)

cols <- intersect(cols, names(DT))

if (length(cols) == 0L) return(NULL)

id\_cols <- setdiff(names(DT), c(cols, "pct","delta")) # ignore legacy

L <- melt(DT, id.vars = id\_cols, measure.vars = cols,

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

L[, scenario := sub("^.\*?\_", "", measure)]

L[, measure := NULL]

if (isTRUE(use\_pct)) {

mx <- suppressWarnings(max(abs(L$value), na.rm = TRUE))

frac\_like <- is.finite(mx) && mx <= 1.001

if (isTRUE(frac\_like)) L[, value := 100 \* value]

}

L[]

}

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; }

.var-help { background: #f6f8fa; border: 1px solid #e1e4e8; padding: 10px 12px; border-radius: 6px; margin-top: 10px; }

"))),

titlePanel(

if (is.null(.project\_id)) "FIDELIO diagnostics" else paste0("FIDELIO diagnostics — ", .project\_id)

),

sidebarLayout(

sidebarPanel(

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

uiOutput("scenario\_filter"),

uiOutput("dynamic\_filters"),

radioButtons("view", "Show:",

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

selected = "plot\_lvl"),

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

hr(),

div(class = "sticky-help", uiOutput("var\_help\_box"))

),

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]] })

# ---- Scenario filter (union of config + present columns)

output$scenario\_filter <- renderUI({

DT <- symDT()

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

cfg <- .cfg

b <- base\_scn(cfg)

pols <- policy\_scns(cfg)

wide\_scns <- scenario\_cols(DT)

pol\_from\_var <- gsub("^(delta|pct)\_", "", grep("^(delta|pct)\_", names(DT), value = TRUE))

pol\_from\_var <- unique(pol\_from\_var)

all\_scns <- unique(c(b, pols, wide\_scns, pol\_from\_var))

all\_scns <- all\_scns[nzchar(all\_scns)]

selectizeInput(

"scn\_pick", "Scenarios:",

choices = setNames(all\_scns, all\_scns),

selected = all\_scns,

multiple = TRUE,

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

)

})

# ---- Dynamic filters for keys (skip scenario/time/derived)

output$dynamic\_filters <- renderUI({

DT <- symDT()

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

scns <- scenario\_cols(DT)

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

grep("^delta\_", names(DT), value = TRUE),

grep("^pct\_", names(DT), value = TRUE)))

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

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

pickers <- vector("list", length(show\_keys))

for (ix in seq\_along(show\_keys)) {

k <- show\_keys[ix]

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

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

pickers[[ix]] <- selectizeInput(

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

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

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

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

multiple = TRUE

)

}

do.call(tagList, pickers)

})

observe({

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

scns <- scenario\_cols(DT)

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

grep("^delta\_", names(DT), value = TRUE),

grep("^pct\_", names(DT), value = TRUE)))

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

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

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

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

}

})

# ---- Filtered slice (keys only; scenario selection handled at plotting stage)

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

)

# ---- Variable help / context box

# helpers (add once, near label\_of/desc\_of)

note\_of <- function(sym) {

# put your custom per-variable guidance here (HTML allowed)

switch(sym,

"OUT\_COMP6\_SHARE\_REAL\_t" = paste(

"Interpretation:",

"Each point shows the sector share changes between base year and final year.",

"The x (y) axis shows the variation in the baseline (policy) scenario.",

"Points on the diagonal: same variation in baseline and policy scenario",

"Above: policy increases the sector's share vs baseline; below: decreases."

),

"BITRADE\_REG\_t" = paste(

"Interpretation:",

"Facets form a matrix of bilateral flows (rows = exporters, columns = importers)."

),

"TB\_GDP\_t" = "Interpretation: ratio of trade balance to GDP; positive = surplus/GDP, negative = deficit/GDP.",

"I\_PP\_SECT6\_t" = "Interpretation: gross investment aggregated into 6 sector groups.",

NULL # default = no extra note

)

}

output$var\_help\_box <- renderUI({

if (is.null(input$sym)) return(NULL)

HTML(sprintf(

'<div class="var-help">

<b>%s</b><br/>

<i>%s</i>%s

</div>',

label\_of(input$sym),

desc\_of(input$sym),

{

nt <- note\_of(input$sym)

if (is.null(nt) || !nzchar(nt)) "" else paste0("<br/>", nt)

}

))

})

# output$var\_help\_box <- renderUI({

# DT <- symDT()

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

# b <- base\_scn(.cfg)

# pols\_avail <- policy\_scns(.cfg)

# scn\_shown <- input$scn\_pick

# scn\_txt <- if (length(scn\_shown) == 0) "(no scenario selected)"

# else paste(scn\_shown, collapse = ", ")

#

# view\_txt <- switch(input$view,

# plot\_lvl = "Levels by scenario (wide → long).",

# plot\_var = "Deviation vs baseline (Δ or %), one line per policy.",

# table = "Raw table (filtered by keys)."

# )

# HTML(sprintf(

# '<div class="var-help"><b>%s</b><br/><i>%s</i><br/>View: %s<br/>Baseline: <code>%s</code>. Policies available: %s.</div>',

# label\_of(input$sym),

# desc\_of(input$sym),

# view\_txt,

# ifelse(length(b), b, "baseline"),

# ifelse(length(pols\_avail), paste(pols\_avail, collapse = ", "), "—")

# ))

# })

# ---- Table

output$tbl <- renderDT({

req(input$view == "table")

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

})

# ---- Plot

output$plt <- renderPlot({

cfg <- .cfg

b <- base\_scn(cfg)

pols\_cfg <- policy\_scns(cfg)

# Which scenarios to show?

scn\_pick <- input$scn\_pick

pol\_pick <- intersect(scn\_pick, pols\_cfg)

pair\_title <- function() {

if (length(pol\_pick) == 0) return("Policy vs baseline")

if (length(pol\_pick) == 1) return(paste(pol\_pick, "vs", b))

paste(paste(pol\_pick, collapse = " + "), "vs", b)

}

# ---- SPECIAL: BITRADE\_REG\_t (faceted matrix)

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

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

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

L <- to\_long\_var(BT, use\_pct = isTRUE(input$asPercent), only\_policies = pol\_pick)

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

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

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

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

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

ylab <- if (isTRUE(input$asPercent)) "Deviation wrt baseline (%)" else "Change (level)"

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

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

geom\_line(linewidth = 0.8) +

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

labs(x = "Year", y = ylab, title = paste("Bilateral trade (macro regions):", pair\_title()),

color = "Policy", linetype = "Policy") +

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)

} else {

Scns <- scenario\_cols(BT)

use\_scns <- if (length(scn\_pick)) intersect(Scns, scn\_pick) else Scns

L <- to\_long\_levels(BT, scenarios = use\_scns); req(!is.null(L), NROW(L) > 0)

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

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

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

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

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

geom\_line() + geom\_point(size = 0.6) +

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

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

title = paste("Bilateral trade (macro regions) — Levels"),

color = "Scenario", linetype = "Scenario") +

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)

}

}

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

if (input$sym == "OUT\_COMP6\_SHARE\_REAL\_t" && length(pols\_cfg) >= 1) {

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

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

p1 <- if (length(pol\_pick) >= 1) pol\_pick[1] else pols\_cfg[1]

L <- to\_long\_levels(DT, scenarios = intersect(scenario\_cols(DT), c(b, p1))); req(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")

if (all(c(b, p1) %in% names(SC))) {

mx <- SC[, max(abs(c(get(b), get(p1))), na.rm = TRUE)]

if (is.finite(mx) && mx <= 1.001) {

SC[, (b) := 100 \* get(b)]

SC[, (p1) := 100 \* get(p1)]

}

} else return(NULL)

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

SC <- SC[is.finite(get(b)) & is.finite(get(p1))]

req(NROW(SC) > 0)

p <- ggplot(SC, aes(x = .data[[b]], y = .data[[p1]], 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(b, " ", yr0, "→", yrT),

y = paste0(p1, " ", yr0, "→", yrT),

color = "Sector group (6)",

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

subtitle = "Above diagonal: policy share increases 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)

}

# ---- Generic plots

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

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

xcol <- if ("year" %in% names(DT)) "year" else if ("t" %in% names(DT)) "t" else setdiff(names(DT), c("delta","pct"))[1]

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

Scns <- scenario\_cols(DT)

use\_scns <- intersect(Scns, input$scn\_pick)

L <- to\_long\_levels(DT, scenarios = use\_scns); req(!is.null(L), NROW(L) > 0)

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")

return(p)

} else {

# Δ / % over policy columns

pol\_from\_var <- gsub("^(delta|pct)\_", "", grep("^(delta|pct)\_", names(DT), value = TRUE))

pol\_pick <- intersect(input$scn\_pick, policy\_scns(cfg))

pol\_pick <- intersect(pol\_pick, unique(pol\_from\_var))

if (length(pol\_pick) == 0) pol\_pick <- unique(pol\_from\_var)

L <- to\_long\_var(DT, use\_pct = isTRUE(input$asPercent), only\_policies = pol\_pick)

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

keys <- setdiff(names(L), c("scenario","value","t","year"))

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) { L[, grp := "all"] } else {

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

}

ylab <- if (isTRUE(input$asPercent)) "Change vs baseline (%)" else "Change vs baseline (level)"

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

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

geom\_line(alpha = 0.95) + geom\_point(size = 0.7, alpha = 0.9) +

labs(x = xcol, y = ylab,

title = paste(pair\_title(), "—", label\_of(input$sym)),

color = "Policy", linetype = "Policy") +

theme\_minimal()

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

return(p)

}

}) %>% shiny::bindCache(

input$sym, input$view, input$asPercent, input$scn\_pick,

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

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

# fidelioDiagnostics — Diagnostics app (multi-scenario ready)

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

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

# scenario helpers

base\_scn <- fidelioDiagnostics:::base\_scn

policy\_scns <- fidelioDiagnostics:::policy\_scns

scenario\_cols <- fidelioDiagnostics:::scenario\_cols

load\_config <- fidelioDiagnostics:::load\_config

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

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

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

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

# Config (project id in title)

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

.cfg <- try(load\_config(), silent = TRUE)

.project\_id <- tryCatch({

if (!inherits(.cfg, "try-error") && !is.null(.cfg$project$id) && nzchar(.cfg$project$id)) .cfg$project$id else NULL

}, error = function(e) NULL)

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

# Data loading

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

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

# App constants (set per app file)

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

APP\_KIND <- "diagnostic\_app" # or "results\_app"

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

# Bundle picker (by project id)

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

.pick\_bundle\_for\_project <- function(dir, project\_id, app\_kind = APP\_KIND) {

# Prefer: bundle\_<app>\_<project>\_\*.{rds,qs}

p1 <- sprintf("^bundle\_%s\_%s\_.\*\\.(rds|qs)$", app\_kind, project\_id)

# Fallbacks: any bundle\_\*\_<project>\_\*.{rds,qs}, or results\_bundle\_<project>\_\*.{rds,qs}

p2 <- sprintf("^bundle\_.\*\_%s\_.\*\\.(rds|qs)$", project\_id)

p3 <- sprintf("^results\_bundle\_%s\_.\*\\.(rds|qs)$", project\_id)

fs <- list.files(dir, full.names = TRUE)

cand <- fs[grepl(p1, basename(fs), ignore.case = TRUE)]

if (!length(cand)) cand <- fs[grepl(p2, basename(fs), ignore.case = TRUE)]

if (!length(cand)) cand <- fs[grepl(p3, basename(fs), ignore.case = TRUE)]

if (!length(cand)) return(NULL)

cand <- cand[order(file.info(cand)$mtime, decreasing = TRUE)]

tools::file\_path\_sans\_ext(basename(cand[1])) # bundle name (without extension)

}

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

# Data loading — auto-detect bundle by project id

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

get\_results <- function() {

cfg <- load\_config()

dir <- tryCatch(outputs\_dir\_from\_config(cfg), error = function(e) NULL)

if (is.null(dir) || !dir.exists(dir)) dir <- file.path("outputs", "derived")

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

cat("\n[", APP\_KIND, "] outputs dir:", normalizePath(dir, winslash = "/"), "\n", sep = "")

# 1) Pick bundle for this project id

project\_id <- cfg$project$id %||% ""

bundle\_name <- if (nzchar(project\_id)) .pick\_bundle\_for\_project(dir, project\_id) else NULL

if (!is.null(bundle\_name)) cat("[", APP\_KIND, "] Using bundle: ", bundle\_name, "\n", sep = "")

# 2) Try to load the chosen bundle, else fall back to "results\_app" / "diagnostic\_app"

try\_names <- unique(na.omit(c(bundle\_name, APP\_KIND)))

b <- list()

for (nm in try\_names) {

tmp <- try(load\_bundle(nm, dir = dir), silent = TRUE)

if (!inherits(tmp, "try-error") && length(tmp)) { b <- tmp; break }

}

if (!length(b)) {

cat("[", APP\_KIND, "] No bundle loaded. Will backfill per-symbol files.\n", sep = "")

b <- list()

}

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

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

if (length(missing)) {

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

for (s in missing) {

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

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

}

}

# 4) Drop NULL/empty & report

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

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

miss <- setdiff(wanted, have)

cat("[", APP\_KIND, "] Loaded symbols: ", paste(have, collapse = ", "), "\n", sep = "")

if (length(miss)) cat("[", APP\_KIND, "] Missing or empty: ", paste(miss, collapse = ", "), "\n", sep = "")

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, short descriptions)

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

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

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",

desc = c(

"Real GDP at constant prices.",

"Real trade balance (exports − imports).",

"Trade balance divided by GDP (dimensionless)."

),

keep = TRUE

),

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",

desc = c(

"Gross investment by 6 aggregated sector groups.",

"Share of real output by 6 aggregated sector groups (sums to 1)."

),

keep = TRUE

),

data.table(

symbol = "BITRADE\_REG\_t",

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

group = "Bilateral trade",

desc = "Bilateral trade matrix across macro regions (origin n → destination n1).",

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]

desc\_of <- function(sym) .meta[symbol == sym, desc ][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\_levels <- function(DT, scenarios = NULL) {

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

if (is.null(scenarios)) scenarios <- scenario\_cols(DT)

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

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

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

}

# long-format Δ / % over per-policy columns produced in PR2

to\_long\_var <- function(DT, use\_pct = TRUE, only\_policies = NULL) {

# build from delta\_<pol> or pct\_<pol>

cfg <- .cfg

b <- base\_scn(cfg)

pols <- policy\_scns(cfg)

if (!is.null(only\_policies)) pols <- intersect(pols, only\_policies)

if (length(pols) == 0L) return(NULL)

# pick columns that exist

suffix <- if (isTRUE(use\_pct)) "pct\_" else "delta\_"

cols <- paste0(suffix, pols)

cols <- intersect(cols, names(DT))

if (length(cols) == 0L) return(NULL)

id\_cols <- setdiff(names(DT), c(cols, "pct","delta")) # legacy cols ignored

L <- melt(DT, id.vars = id\_cols, measure.vars = cols,

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

# measure looks like pct\_<pol> or delta\_<pol> → extract scenario = <pol>

L[, scenario := sub("^.\*?\_", "", measure)]

L[, measure := NULL]

# pretty: if pct is in [0,1] (fraction), convert to %

if (isTRUE(use\_pct)) {

mx <- suppressWarnings(max(abs(L$value), na.rm = TRUE))

frac\_like <- is.finite(mx) && mx <= 1.001

if (isTRUE(frac\_like)) L[, value := 100 \* value]

}

L[]

}

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; }

.var-help { background: #f6f8fa; border: 1px solid #e1e4e8; padding: 10px 12px; border-radius: 6px; margin-top: 10px; }

"))),

titlePanel(

if (is.null(.project\_id)) "FIDELIO results" else paste0("FIDELIO results — ", .project\_id)

),

sidebarLayout(

sidebarPanel(

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

uiOutput("scenario\_filter"),

uiOutput("dynamic\_filters"),

radioButtons("view", "Show:",

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

selected = "plot\_lvl"),

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

hr(),

div(class = "sticky-help", uiOutput("var\_help\_box"))

),

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]] })

# ---- Scenario filter (from data actually present)

output$scenario\_filter <- renderUI({

DT <- symDT()

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

cfg <- .cfg

b <- base\_scn(cfg)

pols <- policy\_scns(cfg)

# from wide level cols

wide\_scns <- scenario\_cols(DT)

# from delta\_\*/pct\_\* cols

pol\_from\_var <- gsub("^(delta|pct)\_", "", grep("^(delta|pct)\_", names(DT), value = TRUE))

pol\_from\_var <- unique(pol\_from\_var)

# union everything

all\_scns <- unique(c(b, pols, wide\_scns, pol\_from\_var))

all\_scns <- all\_scns[nzchar(all\_scns)]

# default select = all available

selectizeInput(

"scn\_pick", "Scenarios:",

choices = setNames(all\_scns, all\_scns),

selected = all\_scns,

multiple = TRUE,

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

)

})

# ---- Dynamic filters for keys (skip scenario/time)

output$dynamic\_filters <- renderUI({

DT <- symDT()

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

scns <- scenario\_cols(DT)

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

grep("^delta\_", names(DT), value = TRUE),

grep("^pct\_", names(DT), value = TRUE)))

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

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

pickers <- vector("list", length(show\_keys))

for (ix in seq\_along(show\_keys)) {

k <- show\_keys[ix]

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

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

pickers[[ix]] <- selectizeInput(

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

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

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

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

multiple = TRUE

)

}

do.call(tagList, pickers)

})

observe({

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

scns <- scenario\_cols(DT)

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

grep("^delta\_", names(DT), value = TRUE),

grep("^pct\_", names(DT), value = TRUE)))

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

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

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

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

}

})

# ---- Filtered slice (keys only; scenario selection handled at plotting stage)

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]

}

}

# legacy pct column shown as percentage on demand

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

)

# ---- Variable help / context box

# helpers (add once, near label\_of/desc\_of)

note\_of <- function(sym) {

# put your custom per-variable guidance here (HTML allowed)

switch(sym,

"OUT\_COMP6\_SHARE\_REAL\_t" = paste(

"Interpretation:",

"Each point shows the sector share changes between base year and final year.",

"The x (y) axis shows the variation in the baseline (policy) scenario.",

"Points on the diagonal: same variation in baseline and policy scenario",

"Above: policy increases the sector's share vs baseline; below: decreases."

),

"BITRADE\_REG\_t" = paste(

"Interpretation:",

"Facets form a matrix of bilateral flows (rows = exporters, columns = importers)."

),

"TB\_GDP\_t" = "Interpretation: ratio of trade balance to GDP; positive = surplus/GDP, negative = deficit/GDP.",

"I\_PP\_SECT6\_t" = "Interpretation: gross investment aggregated into 6 sector groups.",

NULL # default = no extra note

)

}

output$var\_help\_box <- renderUI({

if (is.null(input$sym)) return(NULL)

HTML(sprintf(

'<div class="var-help">

<b>%s</b><br/>

<i>%s</i>%s

</div>',

label\_of(input$sym),

desc\_of(input$sym),

{

nt <- note\_of(input$sym)

if (is.null(nt) || !nzchar(nt)) "" else paste0("<br/>", nt)

}

))

})

# ---- Table

output$tbl <- renderDT({

req(input$view == "table")

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

})

# ---- Plot

output$plt <- renderPlot({

cfg <- .cfg

b <- base\_scn(cfg)

pols\_cfg <- policy\_scns(cfg)

# Which scenarios to show in plots?

scn\_pick <- input$scn\_pick

# For levels plot we allow baseline + any; for Δ/% plot we need policies only.

pol\_pick <- intersect(scn\_pick, pols\_cfg)

# Title helper

pair\_title <- function() {

if (length(pol\_pick) == 0) return("Policy vs baseline")

if (length(pol\_pick) == 1) return(paste(pol\_pick, "vs", b))

paste(paste(pol\_pick, collapse = " + "), "vs", b)

}

# ---- SPECIAL: BITRADE\_REG\_t (faceted matrix)

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

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

# prefer Δ/% long format over legacy columns if user chose plot\_var

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

L <- to\_long\_var(BT, use\_pct = isTRUE(input$asPercent), only\_policies = pol\_pick)

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

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

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

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

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

ylab <- if (isTRUE(input$asPercent)) "Deviation wrt baseline (%)" else "Change (level)"

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

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

geom\_line(linewidth = 0.8) +

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

labs(x = "Year", y = ylab, title = paste("Bilateral trade (macro regions):", pair\_title()),

color = "Policy", linetype = "Policy") +

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)

} else {

# Levels: wide scenarios → long; filter to selected scenarios

Scns <- scenario\_cols(BT)

use\_scns <- if (length(scn\_pick)) intersect(Scns, scn\_pick) else Scns

L <- to\_long\_levels(BT, scenarios = use\_scns); req(!is.null(L), NROW(L) > 0)

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

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

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

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

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

geom\_line() + geom\_point(size = 0.6) +

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

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

title = paste("Bilateral trade (macro regions) — Levels"),

color = "Scenario", linetype = "Scenario") +

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)

}

}

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

if (input$sym == "OUT\_COMP6\_SHARE\_REAL\_t" && length(pol\_pick) >= 1) {

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

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

# choose the first selected policy for the scatter

p1 <- pol\_pick[1]

# Build levels long to compute start/end deltas per scenario

L <- to\_long\_levels(DT, scenarios = intersect(scenario\_cols(DT), c(b, p1))); req(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")

# fraction-like → % points

if (all(c(b, p1) %in% names(SC))) {

mx <- SC[, max(abs(c(get(b), get(p1))), na.rm = TRUE)]

if (is.finite(mx) && mx <= 1.001) {

SC[, (b) := 100 \* get(b)]

SC[, (p1) := 100 \* get(p1)]

}

} else return(NULL)

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

SC <- SC[is.finite(get(b)) & is.finite(get(p1))]

req(NROW(SC) > 0)

p <- ggplot(SC, aes(x = .data[[b]], y = .data[[p1]], 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(b, " ", yr0, "→", yrT),

y = paste0(p1, " ", yr0, "→", yrT),

color = "Sector group (6)",

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

subtitle = "Above diagonal: policy share increases 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)

}

# ---- Generic plots

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

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

xcol <- if ("year" %in% names(DT)) "year" else if ("t" %in% names(DT)) "t" else setdiff(names(DT), c("delta","pct"))[1]

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

# Levels: wide → long; filter to selected scenarios

Scns <- scenario\_cols(DT)

use\_scns <- intersect(Scns, input$scn\_pick) # was: intersect(Scns, input$scn\_pick) or Scns

L <- to\_long\_levels(DT, scenarios = use\_scns); req(!is.null(L), NROW(L) > 0)

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")

return(p)

} else {

# Δ / %: use long builder over delta\_<pol> / pct\_<pol>, colored by policy

# just before building L <- to\_long\_var(...)

pols\_cfg <- policy\_scns(cfg)

scn\_pick <- input$scn\_pick

pol\_pick <- intersect(scn\_pick, pols\_cfg)

# Keep only those with corresponding delta\_/pct\_ cols

pol\_from\_var <- gsub("^(delta|pct)\_", "", grep("^(delta|pct)\_", names(DT), value = TRUE))

pol\_pick <- intersect(pol\_pick, unique(pol\_from\_var))

if (length(pol\_pick) == 0) {

# graceful fallback: show whatever policies are available in the data

pol\_pick <- unique(pol\_from\_var)

}

L <- to\_long\_var(DT, use\_pct = isTRUE(input$asPercent), only\_policies = pol\_pick)

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

L <- to\_long\_var(DT, use\_pct = isTRUE(input$asPercent), only\_policies = pol\_pick)

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

keys <- setdiff(names(L), c("scenario","value","t","year"))

facet\_key <- NULL

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

# group within facets

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

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

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

}

ylab <- if (isTRUE(input$asPercent)) "Change vs baseline (%)" else "Change vs baseline (level)"

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

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

geom\_line(alpha = 0.95) + geom\_point(size = 0.7, alpha = 0.9) +

labs(x = xcol, y = ylab,

title = paste(pair\_title(), "—", label\_of(input$sym)),

color = "Policy", linetype = "Policy") +

theme\_minimal()

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

return(p)

}

}) %>% shiny::bindCache(

input$sym, input$view, input$asPercent, input$scn\_pick,

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"

))

}

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

### App\_loaders

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

# ==== app\_loader.R ====

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

# Resolve project/output paths and load symbols/bundles for the Shiny apps.

# - Prefers project-specific outputs dir from YAML

# - Loads bundles with names that include project id and timestamp

# - Accepts both single "\_" and legacy "\_\_" separators

# - Supports .rds and .qs bundles

# - Accepts project-specific "\_latest" and legacy generic "\_latest"

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

# --- tiny safe infix -----------------------------------------------------------

`%||%` <- function(x, y) if (is.null(x)) y else x

# --- internal sanitizers / ids -------------------------------------------------

.sanitize\_id <- function(x) {

x <- gsub("[^A-Za-z0-9.\_-]+", "-", x)

x <- gsub("-+", "-", x)

x <- gsub("(^-|-$)", "", x)

tolower(x)

}

.project\_id <- function(cfg = NULL) {

pid <- try(cfg$project$id, silent = TRUE)

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

return(.sanitize\_id(pid))

}

gd <- try(cfg$paths$gdx\_dir, silent = TRUE)

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

return(.sanitize\_id(basename(gd)))

}

"project"

}

# --- resolve project root ------------------------------------------------------

#' @keywords internal

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

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

cand <- cand[nzchar(cand)]

for (start in cand) {

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

if (inherits(cur, "try-error")) next

repeat {

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.")

}

# --- derive outputs/derived from config ---------------------------------------

#' @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) # must exist in the package

out <- file.path(cfg$paths$outputs, "derived")

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

}

# --- looser resolver via options/env or default relative ----------------------

# Prefer config path if load\_config() exists and works; else fall back.

#' @keywords internal

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

# First, try config-based location

if (is.null(dir) && exists("load\_config", mode = "function")) {

cfg\_try <- try(load\_config(), silent = TRUE)

if (!inherits(cfg\_try, "try-error")) {

od\_try <- try(outputs\_dir\_from\_config(), silent = TRUE)

if (!inherits(od\_try, "try-error") && dir.exists(od\_try)) {

return(normalizePath(od\_try, winslash = "/", mustWork = TRUE))

}

}

}

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[dir.exists(cand)]

if (!length(hit)) stop(

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

"or env var FIDELIO\_OUTPUTS, or ensure config paths are valid."

)

hit[[1]]

}

# --- manifest loader -----------------------------------------------------------

# Manifest columns expected: symbol, format, path (older 'filename' is accepted)

#' @keywords internal

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

dir <- resolve\_outputs\_dir(dir)

manr <- file.path(dir, "manifest.rds")

manc <- file.path(dir, "manifest.csv")

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

if (file.exists(manc)) return(data.table::fread(manc))

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

}

# --- symbol loader (chooses preferred format) ---------------------------------

#' @keywords internal

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

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

dir <- resolve\_outputs\_dir(dir)

man <- load\_manifest(dir)

# accept older 'filename' column

if (!("path" %in% names(man)) && "filename" %in% names(man)) {

man[, path := filename]

}

need <- c("symbol", "format", "path")

if (!all(need %in% names(man))) {

stop("Manifest must contain columns: symbol, format, path (or filename).")

}

# select this symbol's rows (data.table-safe scoping)

symbol\_ <- symbol

man\_sym <- man[symbol == symbol\_]

if (is.null(man\_sym) || nrow(man\_sym) == 0L) {

stop("Symbol not found in manifest: ", symbol)

}

# pick first available format in 'prefer' order

avail <- match(prefer, man\_sym$format, nomatch = 0L)

avail <- avail[avail > 0L]

if (!length(avail)) stop("No preferred formats found for symbol '", symbol, "'.")

row <- man\_sym[avail[1L]]

p <- ifelse(file.exists(row$path), row$path, file.path(dir, row$path))

if (!file.exists(p)) stop("Data file listed in manifest not found: ", p)

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

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

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

if (grepl("\\.rds$", p, ignore.case = TRUE)) return(readRDS(p))

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

stop("Unknown/unsupported format for: ", p)

}

# --- bundle path resolver ------------------------------------------------------

# Tries, in order, for name = "<app\_kind>" or a logical bundle "name":

# 1) exact: bundle\_<name>.{rds,qs}

# 2) pointer: bundle\_<name>\_<pid>\_latest.{rds,qs} (project-specific), then legacy bundle\_<name>\_latest.{rds,qs}

# 3) newest versioned for this project id:

# bundle\_<name>\_<pid>\_<YYYY-mm-dd\_HH-MM-SS>.{rds,qs}

# (also accepts legacy with double underscores: bundle\_<name>\_\_<pid>\_\_<TS>.\*)

# 4) newest versioned (any pid) if none match #3

.resolve\_bundle\_path <- function(dir, name, cfg = NULL) {

pid <- .project\_id(cfg)

# 1) exact (rds/qs)

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

exact\_candidates <- c(paste0(exact, ".rds"), paste0(exact, ".qs"))

exact\_hit <- exact\_candidates[file.exists(exact\_candidates)]

if (length(exact\_hit)) return(exact\_hit[1])

# 2) project-specific latest, then legacy latest (rds/qs)

latest\_pid <- file.path(dir, paste0("bundle\_", name, "\_", pid, "\_latest"))

latest\_any <- file.path(dir, paste0("bundle\_", name, "\_latest"))

for (stem in c(latest\_pid, latest\_any)) {

cands <- c(paste0(stem, ".rds"), paste0(stem, ".qs"))

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

if (length(hit)) return(hit[1])

}

# 3) versioned for this pid (accept "\_" or "\_\_")

esc <- function(s) gsub("([.^$|()\*+?{}\\[\\]\\\\])", "\\\\\\1", s)

name\_re <- esc(name); pid\_re <- esc(pid)

ts\_re <- "\\d{4}-\\d{2}-\\d{2}\_\\d{2}-\\d{2}-\\d{2}"

rx\_pid <- sprintf("^bundle\_%s\_+%s\_+%s\\.(rds|qs)$", name\_re, pid\_re, ts\_re)

cand1 <- list.files(dir, pattern = rx\_pid, full.names = TRUE, ignore.case = TRUE)

# 4) any pid (accept "\_" or "\_\_")

rx\_any <- sprintf("^bundle\_%s\_+.\*\_+%s\\.(rds|qs)$", name\_re, ts\_re)

cand2 <- if (!length(cand1)) list.files(dir, pattern = rx\_any, full.names = TRUE, ignore.case = TRUE) else character()

candidates <- c(cand1, cand2)

if (length(candidates)) {

info <- file.info(candidates)

return(rownames(info)[which.max(info$mtime)])

}

# Nothing found — return the exact rds (so caller can error with path shown)

paste0(exact, ".rds")

}

# --- bundle loader using the resolver -----------------------------------------

#' @keywords internal

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

dir <- resolve\_outputs\_dir(dir)

# load cfg (for project id) if not provided

if (is.null(cfg) && exists("load\_config", mode = "function")) {

cfg\_try <- try(load\_config(), silent = TRUE)

if (!inherits(cfg\_try, "try-error")) cfg <- cfg\_try

}

p <- .resolve\_bundle\_path(dir, name, cfg = cfg)

if (!file.exists(p)) {

stop(

"Bundle not found: ", p, "\n",

"Looked for exact, project-specific \_latest, legacy \_latest, ",

"and versioned variants in: ", dir, "\n",

"Tip: run the pipeline to (re)create bundles for this project."

)

}

if (grepl("\\.qs$", p, ignore.case = TRUE)) {

if (!requireNamespace("qs", quietly = TRUE)) stop("Package 'qs' is required to read .qs")

obj <- qs::qread(p)

} else {

obj <- readRDS(p)

}

# A tiny sanity: ensure it's a named list

if (!is.list(obj)) stop("Bundle '", basename(p), "' is not a list.")

if (is.null(names(obj))) stop("Bundle '", basename(p), "' has no names.")

message("[load\_bundle] Using bundle: ", basename(p))

obj

}

# --- helper to list available bundles (nice for a dev picker) -----------------

#' @keywords internal

list\_bundles <- function(dir = NULL, name = NULL) {

dir <- resolve\_outputs\_dir(dir)

pat <- if (is.null(name)) "^bundle\_.\*\\.(rds|qs)$"

else sprintf("^bundle\_%s.\*\\.(rds|qs)$", gsub("([.^$|()\*+?{}\\[\\]\\\\])", "\\\\\\1", name))

files <- list.files(dir, pattern = pat, full.names = TRUE)

if (!length(files)) {

return(data.table::data.table(file = character(), mtime = as.POSIXct(character())))

}

info <- file.info(files)

data.table::data.table(file = rownames(info), mtime = info$mtime)[order(-mtime)]

}

### Config\_load.R

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

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

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

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

# Returns a list() in all cases (never NULL), even if the YAML file is empty.

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

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

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

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

obj

}

# ---------------------------- Small path helpers ------------------------------

.is\_abs <- function(p) {

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

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

}

`%||%` <- function(x, y) if (is.null(x)) y else x

.to\_abs <- function(p) {

if (is.null(p) || !nzchar(p)) return("")

p <- path.expand(p)

if (.is\_abs(p)) {

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

} else {

# Resolve relative to the \*current project root\*.

# If you have a proj\_path() in your package, use it; otherwise use getwd().

.proj\_path <- get0("proj\_path", mode = "function", inherits = TRUE,

ifnotfound = function(...) file.path(getwd(), ...))

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

}

}

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

#' Load and validate configuration (defaults + overrides)

#'

#' Robust to empty YAMLs. Ensures required sections/paths exist and are absolute.

#'

#' @param path Path to project YAML (default "config/project.yml"). If a list is

#' supplied, it is treated as the overrides object directly.

#' @param verbose Print resolved paths.

#' @export

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

# -- locate files ------------------------------------------------------------

def\_file <- .to\_abs("config/default.yml")

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

# -- read YAMLs safely (empty -> list()) ------------------------------------

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

override <- if (is.list(path)) path else safe\_read\_yaml(override\_file)

# -- merge (project overrides default) --------------------------------------

# safe: both are lists by construction above

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())

# -- sane 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"

# --- scenarios: must come from YAML; normalize & ensure baseline first if present

if (is.null(cfg$scenarios) || !length(cfg$scenarios)) {

stop("load\_config(): cfg$scenarios missing or empty in YAML. Define at least 'baseline'.")

}

scn <- unique(trimws(as.character(cfg$scenarios)))

if (!length(scn)) {

stop("load\_config(): cfg$scenarios resolved to empty after normalization.")

}

if ("baseline" %in% scn) {

scn <- c("baseline", setdiff(scn, "baseline"))

}

cfg$scenarios <- scn

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

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

if (nzchar(cfg$paths$outputs))

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

if (nzchar(cfg$paths$cache))

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

}

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

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

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

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

# Returns a list() in all cases (never NULL), even if the YAML file is empty.

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

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

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

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

obj

}

# ---------------------------- Small path helpers ------------------------------

.is\_abs <- function(p) {

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

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

}

`%||%` <- function(x, y) if (is.null(x)) y else x

.to\_abs <- function(p) {

if (is.null(p) || !nzchar(p)) return("")

p <- path.expand(p)

if (.is\_abs(p)) {

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

} else {

# Resolve relative to the \*current project root\*.

# If you have a proj\_path() in your package, use it; otherwise use getwd().

.proj\_path <- get0("proj\_path", mode = "function", inherits = TRUE,

ifnotfound = function(...) file.path(getwd(), ...))

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

}

}

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

#' Load and validate configuration (defaults + overrides)

#'

#' Robust to empty YAMLs. Ensures required sections/paths exist and are absolute.

#'

#' @param path Path to project YAML (default "config/project.yml"). If a list is

#' supplied, it is treated as the overrides object directly.

#' @param verbose Print resolved paths.

#' @export

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

# -- locate files ------------------------------------------------------------

def\_file <- .to\_abs("config/default.yml")

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

# -- read YAMLs safely (empty -> list()) ------------------------------------

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

override <- if (is.list(path)) path else safe\_read\_yaml(override\_file)

# -- merge (project overrides default) --------------------------------------

# safe: both are lists by construction above

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

### Derive\_indicators.R

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

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

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

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

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

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

ic <- as.character(ic)

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.

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

stopifnot(data.table::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 (e.g., "TOT")

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

# map to 6 groups and aggregate

G <- cbind(DT[, .(i = .sector\_group6\_vec(i), n, t)], DT[, ..scenarios])

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

# optional total across groups (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) {

wanted <- cfg$derived$include

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

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

}

out <- list()

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

Filter(Negate(is.null), out)

}

# legacy (no longer used) helper kept for backwards-compatibility

.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 (can be moved 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")

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 (multi-scenario) --------------

#' Build derived tables from wide (results\_by\_symbol)

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

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

#' @return named list of wide data.tables (all scenarios kept)

#' @export

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

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

scenarios <- cfg$scenarios

base\_scn <- scenarios[1]

pol\_scns <- scenarios[-1]

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 = scenarios, by = .(n, t)]

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

I\_TOT <- add\_var\_cols\_multi(I\_TOT, base = base\_scn, policies = pol\_scns, cfg = cfg)

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 = scenarios

)

TB\_M <- sum\_value\_by(

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

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

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

scenarios = 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 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", scenarios), with = FALSE]

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

TB <- add\_var\_cols\_multi(TB, base = base\_scn, policies = pol\_scns, cfg = cfg)

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), 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 = scenarios)

TBr <- add\_var\_cols\_multi(TBr, base = base\_scn, policies = pol\_scns, cfg = cfg)

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", scenarios), with = FALSE],

GDP[, c("n","t", scenarios), with = FALSE],

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

for (s in scenarios) {

TB\_GDP[, (s) := data.table::fifelse(

get(paste0(s,"\_GDP")) == 0 | is.na(get(paste0(s,"\_GDP"))),

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

)]

}

# keep only scenario columns after ratio

drop\_cols <- grep("\_(TB|GDP)$", names(TB\_GDP), value = TRUE)

if (length(drop\_cols)) TB\_GDP[, (drop\_cols) := NULL]

TB\_GDP <- add\_var\_cols\_multi(TB\_GDP, base = base\_scn, policies = pol\_scns, cfg = cfg)

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 = scenarios

)

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

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

if (!is.null(FS\_INV) && !is.null(out[["TB\_t"]]) &&

!is.null(HSAV) && !is.null(HDY) &&

!is.null(GSUR) && !is.null(GINV)) {

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

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

keys <- Reduce(function(a,b) unique(rbindlist(list(a,b), use.names = TRUE, fill = TRUE)),

lapply(objs, function(D) D[, .(n,t)]))

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

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

# bind all sources side-by-side

X <- keys

for (nm in names(objs)) {

X <- m2(X, objs[[nm]][, c("n","t", scenarios), with = TRUE])

}

# Compute per-scenario FS

for (s in scenarios) {

X[, (s) := get(paste0(s, ".x"))] # placeholder, we will overwrite below if duplicated by merges

}

# Disambiguate names after merges: ensure unique accessors

# Build local aliases

INVp <- objs$FS\_INV; TBp <- objs$TB; HSAVp <- objs$HSAV; HDYp <- objs$HDY; GSURp <- objs$GSUR; GINVp <- objs$GINV

FS\_out <- keys

for (s in scenarios) {

FS\_out <- merge(FS\_out, INVp[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(FS\_out, "v", paste0("INV\_", s))

FS\_out <- merge(FS\_out, TBp[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(FS\_out, "v", paste0("TB\_", s))

FS\_out <- merge(FS\_out, HSAVp[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(FS\_out, "v", paste0("HSAV\_", s))

FS\_out <- merge(FS\_out, HDYp[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(FS\_out, "v", paste0("HDY\_", s))

FS\_out <- merge(FS\_out, GSURp[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(FS\_out, "v", paste0("GSUR\_", s))

FS\_out <- merge(FS\_out, GINVp[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(FS\_out, "v", paste0("GINV\_", s))

FS\_out[, (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))]

}

# keep only (n,t, scenarios)

keep <- c("n","t", scenarios)

FS\_out <- FS\_out[, keep, with = FALSE]

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

FS\_out <- add\_var\_cols\_multi(FS\_out, base = base\_scn, policies = pol\_scns, cfg = cfg)

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"]]

if (!is.null(HSAV) && !is.null(HDY) && !is.null(GSUR) && !is.null(GINV)) {

HSAV <- norm\_key\_types(HSAV); HDY <- norm\_key\_types(HDY)

GSUR <- norm\_key\_types(GSUR); GINV <- norm\_key\_types(GINV)

keys <- Reduce(function(a,b) unique(rbindlist(list(a,b), use.names = TRUE, fill = TRUE)),

lapply(list(HSAV, HDY, GSUR, GINV), function(D) D[, .(n,t)]))

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

DS\_out <- keys

for (s in scenarios) {

DS\_out <- merge(DS\_out, HSAV[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(DS\_out, "v", paste0("HSAV\_", s))

DS\_out <- merge(DS\_out, HDY[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(DS\_out, "v", paste0("HDY\_", s))

DS\_out <- merge(DS\_out, GSUR[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(DS\_out, "v", paste0("GSUR\_", s))

DS\_out <- merge(DS\_out, GINV[, .(n,t, v = get(s))], by = c("n","t"), all.x = TRUE); data.table::setnames(DS\_out, "v", paste0("GINV\_", s))

DS\_out[, (s) := get(paste0("GSUR\_", s)) +

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

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

}

keep <- c("n","t", scenarios)

DS\_out <- DS\_out[, keep, with = FALSE]

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

DS\_out <- add\_var\_cols\_multi(DS\_out, base = base\_scn, policies = pol\_scns, cfg = cfg)

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", scenarios), with = FALSE]

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

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 scenarios) {

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", scenarios), with = FALSE]

KL <- add\_var\_cols\_multi(KL, base = base\_scn, policies = pol\_scns, cfg = cfg)

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

# ΣK & ΣL by (n,t), then ratio

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

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

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

for (s in scenarios) {

KL\_cty[, (s) := fifelse(get(paste0(s,"\_L")) == 0 | is.na(get(paste0(s,"\_L"))),

NA\_real\_, get(paste0(s,"\_K")) / get(paste0(s,"\_L")))]

}

KL\_cty[, grep("\_(K|L)$", names(KL\_cty), value = TRUE) := NULL]

KL\_cty <- add\_var\_cols\_multi(KL\_cty, base = base\_scn, policies = pol\_scns, cfg = cfg)

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", scenarios), with = FALSE]

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

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

for (s in scenarios) {

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", scenarios), with = FALSE]

PKL <- add\_var\_cols\_multi(PKL, base = base\_scn, policies = pol\_scns, cfg = cfg)

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)) {

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

I\_PP <- wide\_by\_scenario(I\_PP, scenarios = scenarios)

}

sel <- c("n","i","t", scenarios)

I\_PP\_sub <- I\_PP[, sel, with = FALSE]

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

I\_PP\_sub, scenarios = scenarios, append\_original = FALSE

)

I\_PP\_G6 <- add\_macroregions\_additive(I\_PP\_G6, EU28, scenarios = scenarios)

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

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

}

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

GOq\_by\_i <- NULL

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

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

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

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

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

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

for (s in scenarios) {

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", scenarios), 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 = scenarios, append\_original = FALSE)

# 2) Add EU macroregions \*\*before\*\* computing shares

GO6q <- add\_macroregions\_additive(GO6q, EU28, scenarios = scenarios)

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

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

for (s in scenarios) {

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 each policy)

GO6q <- add\_var\_cols\_multi(GO6q, base = base\_scn, policies = pol\_scns, cfg = cfg)

out[["OUT\_COMP6\_SHARE\_REAL\_t"]] <- GO6q[]

} else {

message("• Skipping OUT\_COMP6\_SHARE\_REAL\_t: need either Q\_t or (VA\_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 = scenarios, by = .(n, n1, c, t)]

imp\_tot <- BT\_reg[, lapply(.SD, sum), .SDcols = scenarios, by = .(n, c, t)]

imp\_tot[, n1 := "TOT"]; data.table::setcolorder(imp\_tot, c("n","n1","c","t", scenarios))

exp\_tot <- BT\_reg[, lapply(.SD, sum), .SDcols = scenarios, by = .(n1, c, t)]

exp\_tot[, n := "TOT"]; data.table::setcolorder(exp\_tot, c("n","n1","c","t", scenarios))

BT\_reg2 <- data.table::rbindlist(list(BT\_reg, imp\_tot, exp\_tot), use.names = TRUE)

prod\_tot <- BT\_reg2[, lapply(.SD, sum), .SDcols = scenarios, by = .(n, n1, t)]

prod\_tot[, c := "TOT"]; data.table::setcolorder(prod\_tot, c("n","n1","c","t", scenarios))

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\_multi(BT\_reg3, base = base\_scn, policies = pol\_scns, cfg = cfg)

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.

# --- internal config helper ----------------------------------------------------

.get\_cfg <- function(cfg = NULL) {

if (!is.null(cfg)) return(cfg)

if (exists("load\_config", mode = "function")) return(load\_config())

stop("No cfg provided and load\_config() not found.", call. = FALSE)

}

# --- 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

}

# --- discover scenario columns present in a \*wide\* table -----------------------

# Intersect cfg$scenarios with DT column names (ignores derived delta\_\*/pct\_\*).

scenario\_cols\_in\_DT <- function(DT, cfg = NULL) {

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

cfg <- .get\_cfg(cfg)

intersect(cfg$scenarios %||% character(), names(DT))

}

`%||%` <- function(a, b) if (is.null(a)) b else a

# --- PR1 legacy helper: single delta/pct (kept for backward compat) -----------

# Adds columns: delta, pct where pct = (pol/base) - 1

add\_var\_cols <- function(DT, base = NULL, pol = NULL, cfg = NULL) {

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

data.table::setDT(DT)

cfg <- .get\_cfg(cfg)

if (is.null(base)) base <- fidelioDiagnostics:::base\_scn(cfg)

if (is.null(pol)) pol <- fidelioDiagnostics:::policy\_scns(cfg)[1]

if (is.na(pol) || !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[]

}

# --- PR2 core: multi-scenario deltas/pcts vs baseline --------------------------

# For each policy in `policies`, adds: delta\_<pol>, pct\_<pol>

# pct\_<pol> = (policy / baseline) - 1, guarded for baseline ~ 0

add\_var\_cols\_multi <- function(DT,

base = NULL,

policies = NULL,

cfg = NULL,

tol\_zero = 1e-12) {

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

data.table::setDT(DT)

cfg <- .get\_cfg(cfg)

if (is.null(base)) base <- fidelioDiagnostics:::base\_scn(cfg)

if (is.null(policies)) policies <- fidelioDiagnostics:::policy\_scns(cfg)

# no-op if baseline missing or there are no policy columns

if (!base %in% names(DT)) return(DT[])

policies <- intersect(policies, names(DT))

if (length(policies) == 0L) return(DT[])

for (pol in policies) {

delta\_col <- paste0("delta\_", pol)

pct\_col <- paste0("pct\_", pol)

# delta = policy - baseline

DT[, (delta\_col) := get(pol) - get(base)]

# pct = (policy / baseline) - 1 with near-zero guard

DT[, (pct\_col) := {

b <- get(base)

p <- get(pol)

data.table::fifelse(

is.na(b) | abs(b) <= tol\_zero,

data.table::fifelse(is.na(p) & is.na(b), NA\_real\_,

data.table::fifelse(abs(p) <= tol\_zero, 0.0, NA\_real\_)),

(p / b) - 1.0

)

}]

}

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 = NULL, cfg = NULL) {

cfg <- .get\_cfg(cfg)

if (is.null(scenarios)) scenarios <- cfg$scenarios

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]]]

}

}

# scenario columns present + derived names to exclude from keys

scn\_cols <- c(cfg$scenarios, "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 vars over 'n' ------

add\_macroregions\_additive <- function(DT, eu\_members,

scenarios = NULL, cfg = NULL) {

cfg <- .get\_cfg(cfg)

if (is.null(scenarios)) scenarios <- cfg$scenarios

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) && nrow(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 (one col per scn) ---

wide\_by\_scenario <- function(DT, scenarios = NULL, cfg = NULL) {

if (is.null(DT) || !nrow(DT)) return(DT)

DT <- norm\_key\_types(DT)

cfg <- .get\_cfg(cfg)

if (is.null(scenarios)) scenarios <- cfg$scenarios

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\_

)

# Ensure all configured scenarios exist as columns (fill with NA if missing)

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\_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)

# Locate the app within the installed package

pkg <- utils::packageName()

app\_dir <- system.file("app", app, package = pkg)

if (!nzchar(app\_dir) || !dir.exists(app\_dir)) {

stop("App '", app, "' not found in this package (", pkg, ").")

}

# If the user passed project\_root, advertise it so resolve\_project\_root() works

if (!is.null(project\_root) && nzchar(project\_root)) {

options(fidelio.project = normalizePath(project\_root, winslash = "/", mustWork = TRUE))

on.exit(options(fidelio.project = NULL), add = TRUE)

}

# Resolve outputs/derived (preferred: from config; fallback: option/env)

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; return "" if not set

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 = '...') # uses 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."

)

}

# Set the option the loaders use

outputs\_dir <- normalizePath(outputs\_dir, winslash = "/", mustWork = TRUE)

options(fidelio.outputs = outputs\_dir)

on.exit(options(fidelio.outputs = NULL), add = TRUE)

# Lightweight sanity check so the user gets a helpful nudge

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

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

any\_manifest <- file.exists(man\_r) || file.exists(man\_c)

# Also check if at least one bundle exists (exact, \_latest, or versioned)

has\_any\_bundle <- length(list.files(

outputs\_dir,

pattern = "^bundle\_.\*\\.(rds)$",

full.names = FALSE

)) > 0

if (!any\_manifest && !has\_any\_bundle) {

warning(

"No manifest or bundles found in: ", outputs\_dir, "\n",

"Run `run\_pipeline()` to generate artifacts before launching the app."

)

}

message(

"[fidelioDiagnostics] Launching '", app, "' app\n",

" - package: ", pkg, "\n",

" - app dir: ", app\_dir, "\n",

" - outputs: ", outputs\_dir, "\n",

" - manifest: ", if (any\_manifest) "found" else "missing", "\n"

)

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-level once-guards (single system) -----------------------------------

.pipeline\_flags\_env <- new.env(parent = emptyenv())

.pipeline\_reset\_flags <- function() {

rm(list = ls(.pipeline\_flags\_env, all.names = TRUE), envir = .pipeline\_flags\_env)

}

.pipeline\_flag\_set <- function(key) {

if (exists(key, envir = .pipeline\_flags\_env, inherits = FALSE)) return(TRUE)

assign(key, TRUE, envir = .pipeline\_flags\_env)

FALSE

}

#' Run the end-to-end pipeline: extract → wide → aggregates → deltas → derived → save

#' @export

run\_pipeline <- function(config = "config/project.yml") {

.pipeline\_reset\_flags()

# --- config & runtime --------------------------------------------------------

cfg <- if (is.list(config)) config else load\_config(verbose = TRUE)

print\_runtime\_info(cfg)

ts <- .ts\_stamp()

`%||%` <- function(a, b) if (is.null(a)) b else a

# convenience

scenarios <- cfg$scenarios

if (is.null(scenarios) || length(scenarios) == 0L) {

stop("cfg$scenarios is empty. Ensure YAML defines 'scenarios:' with baseline first.")

}

base\_scn <- scenarios[1]

pol\_scns <- scenarios[-1]

# --- 1) what to extract ------------------------------------------------------

reg <- plan\_extractions(cfg)

# --- 2) extract long (dims + scenario + value) -------------------------------

raw <- extract\_all(cfg, reg)

# --- 3) wide with aggregates + Δ/% (build 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"

)

}

postproc <- function(dt, name) {

if (is.null(dt) || !nrow(dt)) return(NULL)

w <- wide\_by\_scenario(dt, scenarios = cfg$scenarios, cfg = cfg)

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, cfg = cfg)

}

if (length(pol\_scns)) {

w <- add\_var\_cols\_multi(w, base = base\_scn, policies = pol\_scns, cfg = cfg)

}

w[]

}

results\_by\_symbol <- lapply(names(raw), function(sym) postproc(raw[[sym]], sym))

names(results\_by\_symbol) <- names(raw)

# --- 4) derived from wide ----------------------------------------------------

derived <- derive\_from\_wide(results\_by\_symbol, cfg)

ensure\_derived\_shape <- function(DT, key\_cols = NULL) {

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

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

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)

}

sc\_cols\_present <- intersect(names(DT), cfg$scenarios)

if (length(sc\_cols\_present) >= 1L) {

if (base\_scn %in% sc\_cols\_present && length(pol\_scns)) {

policies\_here <- intersect(pol\_scns, sc\_cols\_present)

if (length(policies\_here)) add\_var\_cols\_multi(DT, base = base\_scn, policies = policies\_here, cfg = cfg)

}

return(DT[])

}

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

if (all(need %in% names(DT))) {

f <- if (length(key\_cols)) as.formula(paste(paste(key\_cols, collapse = " + "), "~ scenario"))

else as.formula("1 ~ scenario")

W <- data.table::dcast(DT, f, value.var = "value", fill = NA\_real\_)

sc\_cols\_present <- intersect(names(W), cfg$scenarios)

if (base\_scn %in% sc\_cols\_present && length(pol\_scns)) {

policies\_here <- intersect(pol\_scns, sc\_cols\_present)

if (length(policies\_here)) add\_var\_cols\_multi(W, base = base\_scn, policies = policies\_here, cfg = cfg)

}

return(W[])

}

DT[]

}

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()

for (nm in incl) {

if (nm %in% names(derived)) next

if (!(nm %in% names(results\_by\_symbol))) next

D <- data.table::copy(results\_by\_symbol[[nm]])

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

if (!.pipeline\_flag\_set("save-base")) {

save\_artifacts(results\_by\_symbol, cfg, subdir = "base", ts\_shared = ts)

} else message("[SKIP] save\_artifacts(base): already saved in this run.")

if (length(derived)) {

if (!.pipeline\_flag\_set("save-derived")) {

save\_artifacts(derived, cfg, subdir = "derived", ts\_shared = ts)

} else message("[SKIP] save\_artifacts(derived): already saved in this run.")

}

message("Built results\_by\_symbol: ", paste(names(results\_by\_symbol), collapse = ", "))

if (length(derived)) message("Built derived: ", paste(names(derived), collapse = ", "))

# --- 6) One bundling pass at the end ----------------------------------------

out\_root <- tryCatch(resolve\_outputs\_dir(cfg), error = function(e) cfg$paths$outputs)

make\_bundles(

objs = derived, # universe to pick from

out\_root = out\_root,

bundles\_spec = cfg$save$bundles, # e.g. results\_app / diagnostic\_app

cfg = cfg,

ts\_shared = ts

)

# --- 7) Optional CSV export (now uses a proper bundle name) ------------------

if (isTRUE(cfg$export\_csv$enabled)) {

# The bundle NAME, not the spec:

bundle\_name <- cfg$export\_csv$bundle\_name %||% "results\_app"

if (!.pipeline\_flag\_set(paste0("csv:", bundle\_name))) {

export\_results\_csv(

cfg = cfg,

bundle\_name = bundle\_name,

out\_basename = cfg$export\_csv$out\_basename %||% "results\_bundle\_template",

model\_name = cfg$export\_csv$model\_name %||% "FIDELIO",

pct\_as\_percent = isTRUE(cfg$export\_csv$pct\_as\_percent),

include\_dim\_names = isTRUE(cfg$export\_csv$include\_dim\_names),

unit\_overrides = cfg$export\_csv$unit\_overrides %||% list()

)

} else {

message("[SKIP] export\_results\_csv(): already written for '", bundle\_name, "' in this run.")

}

}

invisible(list(cfg = cfg, raw = raw, results\_by\_symbol = results\_by\_symbol, derived = derived))

}

### R2csv.R

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

# ==== R2csv.R =====

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

# Convert a saved bundle (e.g. "results\_app") into a long CSV with a PyPSA-like

# `variable` column and a `pct\_change` vs baseline for ALL scenarios.

#' 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() (quiet) for you.

#' @param bundle\_name name of the saved bundle to read (e.g. "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\_change by 100.

#' @param include\_dim\_names if TRUE, encode dims as `i=C24|n1=USA`;

#' if FALSE, encode as `C24|USA` (no set prefixes).

#' @param default\_unit fallback unit for symbols not listed in `unit\_overrides`.

#' @param unit\_overrides named list mapping symbol -> unit string.

#' @param csv\_sep field separator passed to fwrite (use ";" for EU Excel).

#' @param tol\_zero numeric tolerance for baseline ~ 0 (avoid Inf).

#' @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 = FALSE,

default\_unit = "million EUR",

unit\_overrides = list(),

csv\_sep = ",",

tol\_zero = 1e-12) {

# ---- package helpers ----

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

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

load\_config <- fidelioDiagnostics:::load\_config

.project\_id <- fidelioDiagnostics:::.project\_id

`%||%` <- function(x, y) if (is.null(x)) y else x

# ---- config (quiet & robust) ----

if (is.null(cfg) || is.character(cfg)) {

cfg <- tryCatch(load\_config(verbose = FALSE), error = function(e) NULL)

}

if (is.null(cfg)) {

stop("export\_results\_csv(): cannot load config; pass cfg or fix load\_config().")

}

if (is.null(cfg$scenarios) || length(cfg$scenarios) < 1) {

stop("export\_results\_csv(): cfg$scenarios missing or empty; set it in YAML.")

}

scn\_cfg <- cfg$scenarios

base\_name <- scn\_cfg[1]

# enforce scalar bundle\_name

if (length(bundle\_name) != 1L) {

stop("export\_results\_csv(): 'bundle\_name' must be length 1, got: ",

paste(bundle\_name, collapse = ", "))

}

# ---- locate bundle file (prefer \_latest; else newest versioned) ------------

out\_root <- tryCatch(resolve\_outputs\_dir(cfg), error = function(e) cfg$paths$outputs)

bundles\_dir <- file.path(out\_root, "derived")

dir.create(bundles\_dir, recursive = TRUE, showWarnings = FALSE)

pid <- .project\_id(cfg)

bundle\_latest <- file.path(bundles\_dir, sprintf("bundle\_%s\_%s\_latest.rds", bundle\_name, pid))

bundle\_file <- NULL

if (file.exists(bundle\_latest)) {

bundle\_file <- bundle\_latest

} else {

pats <- sprintf("^bundle\_%s\_%s\_.\*\\.rds$", bundle\_name, pid)

cand <- list.files(bundles\_dir, pattern = pats, full.names = TRUE)

if (length(cand)) bundle\_file <- cand[which.max(file.mtime(cand))]

}

if (is.null(bundle\_file) || !file.exists(bundle\_file)) {

stop("Bundle not found. Looked for: ",

sprintf("bundle\_%s\_%s\_latest.rds", bundle\_name, pid),

" or the newest versioned file in: ", bundles\_dir)

}

b <- readRDS(bundle\_file)

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) {

s <- as.character(symbol)

if (length(unit\_overrides) && s %in% names(unit\_overrides)) {

u <- unit\_overrides[[s]]

if (!is.null(u) && nzchar(u)) return(u)

}

if (s %in% names(special\_units)) return(special\_units[[s]])

default\_unit

}

# ---- normalize wide -> long (scenario/value) --------------------------------

to\_long\_scn <- function(DT) {

data.table::setDT(DT)

meas\_wide <- intersect(names(DT), scn\_cfg)

if (length(meas\_wide) >= 1L) {

# drop delta/pct columns if present

drop\_cols <- c(

meas\_wide,

grep("^delta($|\_)", names(DT), value = TRUE),

grep("^pct($|\_)", names(DT), value = TRUE)

)

id\_cols <- setdiff(names(DT), drop\_cols)

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("Table must be either wide (scenario columns) or long (scenario/value).")

}

L[]

}

# ---- pct\_change vs baseline (guard for ~0 baseline) ------------------------

compute\_pct\_change <- function(L) {

data.table::setDT(L)

key\_candidates <- c("n","n1","i","c","au","oc","t")

keys <- intersect(names(L), key\_candidates)

base\_tbl <- L[scenario == base\_name, c(keys, "value"), with = FALSE]

if (length(keys)) data.table::setkeyv(base\_tbl, keys)

data.table::setnames(base\_tbl, "value", "base\_value")

L2 <- if (length(keys)) merge(L, base\_tbl, by = keys, all.x = TRUE) else {

# no keys -> one baseline value per whole table

base\_tbl[, scenario := base\_name]

L[, base\_value := base\_tbl$base\_value[1]]

L

}

L2[, pct\_change := {

b <- base\_value

v <- value

out <- ifelse(

scenario == base\_name, NA\_real\_,

ifelse(is.na(b) | abs(b) <= tol\_zero,

ifelse(is.na(v) | abs(v) <= tol\_zero, 0.0, NA\_real\_),

(v / b) - 1.0)

)

if (pct\_as\_percent) out <- 100 \* out

out

}]

L2[, base\_value := NULL]

L2[]

}

# ---- build PyPSA-like variable string --------------------------------------

make\_var <- function(symbol, DT) {

base <- sub("\_t$", "", as.character(symbol))

# columns we never encode in the variable label

ignore <- c("n","region","t","year","scenario","value","delta","pct",

"pct\_change","model","unit","variable")

# preferred order of dimensions (the rest follow)

priority <- c("i","c","n1","au","oc")

dims\_all <- setdiff(names(DT), ignore)

dims\_ord <- c(intersect(priority, dims\_all), setdiff(dims\_all, priority))

if (!length(dims\_ord)) return(base)

# Build a character matrix of labels, row-aligned with DT

# 1) coerce each dim column to character (avoids nzchar() on factors)

labs <- lapply(dims\_ord, function(d) as.character(DT[[d]]))

labs\_dt <- data.table::as.data.table(labs)

# 2) empty strings -> NA (nzchar now safe because columns are character)

for (j in seq\_along(dims\_ord)) {

v <- labs\_dt[[j]]

v[!nzchar(v)] <- NA\_character\_

labs\_dt[[j]] <- v

}

# 3) optionally prefix with dim names (i=..., c=..., etc.)

if (isTRUE(include\_dim\_names)) {

for (j in seq\_along(dims\_ord)) {

d <- dims\_ord[j]

v <- labs\_dt[[j]]

labs\_dt[[j]] <- ifelse(is.na(v), NA\_character\_, paste0(d, "=", v))

}

}

# 4) row-wise collapse of non-NA pieces with "|"

parts <- apply(labs\_dt, 1L, function(r) {

r <- r[!is.na(r)]

if (!length(r)) "" else paste(r, collapse = "|")

})

# 5) prepend the symbol base

ifelse(parts == "", base, paste(base, parts, sep = "|"))

}

# ---- walk bundle, normalize and stack --------------------------------------

rows <- vector("list", length(b))

names(rows) <- names(b)

for (sym in names(b)) {

W <- data.table::as.data.table(b[[sym]])

if (!nrow(W)) next

# long

L <- to\_long\_scn(W)

# region & year conveniences

if (!"n" %in% names(L)) L[, n := NA\_character\_]

L[, region := as.character(n)]

if ("t" %in% names(L) && !"year" %in% names(L)) {

# if you later expose base year in cfg, replace 2014 with cfg$base\_year

L[, year := 2014L + as.integer(t)]

}

# pct\_change vs baseline

L <- compute\_pct\_change(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]

# assemble

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(bundles\_dir, 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 ========

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

# ---- tiny safe infix ----------------------------------------------------------

`%||%` <- function(x, y) if (is.null(x)) y else x

# ---- internal helpers ---------------------------------------------------------

.sanitize\_id <- function(x) {

x <- gsub("[^A-Za-z0-9.\_-]+", "-", x)

x <- gsub("-+", "-", x)

x <- gsub("(^-|-$)", "", x)

tolower(x)

}

.project\_id <- function(cfg) {

pid <- try(cfg$project$id, silent = TRUE)

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

return(.sanitize\_id(pid))

}

gd <- try(cfg$paths$gdx\_dir, silent = TRUE)

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

return(.sanitize\_id(basename(gd)))

}

"project"

}

.ts\_stamp <- function() {

format(Sys.time(), "%Y-%m-%d\_%H-%M-%S")

}

# --- 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::as.data.table(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)

for (k in dims\_all) if (!k %in% names(D)) D[, (k) := NA]

meas <- intersect(prefer, names(D))

D[, symbol := symbol]

data.table::setcolorder(D, c("symbol", dims\_all, meas))

D[]

}

# --- main entry: save according to config -------------------------------------

# objs: named list of data.tables (your results\_by\_symbol)

# cfg : parsed YAML config list

# subdir: optional subdirectory under outputs (rarely needed)

#' @export

# ------------------------------------------------------------------------------

# Save artifacts to outputs/<subdir>.

# - objs: named list of data.frames/data.tables keyed by symbol name

# - subdir: "base" or "derived" (or NULL for outputs root)

# - ts\_shared: optional run-level timestamp (so all files share the same stamp)

# - write\_list\_from: which subdir is allowed to write the consolidated list RDS

# (default "derived"); guarded so it happens once per run.

# Requires helpers:

# - .project\_id(cfg)

# - .ts\_stamp()

# - .pipeline\_flag\_set(key) # once-guard for the current run

# - resolve\_outputs\_dir(cfg) # optional; falls back to cfg$paths$outputs

# ------------------------------------------------------------------------------

save\_artifacts <- function(objs, cfg,

subdir = c("base", "derived"),

ts\_shared = NULL,

write\_list\_from = "derived") {

stopifnot(is.list(objs))

# --- resolve output root dir safely ---

out\_root <- tryCatch(resolve\_outputs\_dir(cfg), error = function(e) NULL)

if (is.null(out\_root)) out\_root <- cfg$paths$outputs

if (!dir.exists(out\_root)) dir.create(out\_root, recursive = TRUE, showWarnings = FALSE)

# normalize subdir

subdir <- if (is.null(subdir)) NULL else match.arg(subdir)

outdir <- if (is.null(subdir)) out\_root else file.path(out\_root, subdir)

dir.create(outdir, recursive = TRUE, showWarnings = FALSE)

# run-level timestamp (shared) + project id

pid <- .project\_id(cfg)

ts <- if (!is.null(ts\_shared)) ts\_shared else .ts\_stamp()

# ----- 0) per-subdir once-guard (prevents double-writes in same run) -----

# (you can remove this if you want to allow re-save within a run)

if (!is.null(subdir)) {

if (.pipeline\_flag\_set(paste0("save\_artifacts:", subdir))) {

message("[SKIP] save\_artifacts(", subdir, "): already saved in this run.")

return(invisible(outdir))

}

}

# ----------------------------------------------------------------------------

# 1) OPTIONAL: per-symbol files (disabled by default)

# ----------------------------------------------------------------------------

per\_symbol <- isTRUE(cfg$save$per\_symbol)

fmts <- cfg$save$formats

wrote\_any\_per\_symbol <- FALSE

if (per\_symbol) {

if (is.null(fmts) || !length(fmts)) fmts <- "csv" # default only when per\_symbol=TRUE

fmts <- tolower(unique(fmts))

for (nm in names(objs)) {

dt <- objs[[nm]]

# skip empty or NULL safely

nrows <- tryCatch(NROW(dt), error = function(e) 0L)

if (is.null(dt) || nrows == 0L) next

base <- file.path(outdir, nm)

if ("feather" %in% fmts) {

arrow::write\_feather(dt, paste0(base, ".feather"))

wrote\_any\_per\_symbol <- TRUE

}

if ("parquet" %in% fmts) {

arrow::write\_parquet(dt, paste0(base, ".parquet"))

wrote\_any\_per\_symbol <- TRUE

}

if ("fst" %in% fmts) {

fst::write\_fst(dt, paste0(base, ".fst"))

wrote\_any\_per\_symbol <- TRUE

}

if ("csv" %in% fmts) {

data.table::fwrite(dt, paste0(base, ".csv"))

wrote\_any\_per\_symbol <- TRUE

}

if ("rds" %in% fmts) {

saveRDS(dt, paste0(base, ".rds"))

wrote\_any\_per\_symbol <- TRUE

}

}

}

# ----------------------------------------------------------------------------

# 2) CONSOLIDATED list RDS (one file per run), written only from `write\_list\_from`

# ----------------------------------------------------------------------------

if (identical(subdir, write\_list\_from) && isTRUE(cfg$save$list\_rds$enabled)) {

if (!.pipeline\_flag\_set("list-rds")) {

bundles\_dir <- file.path(out\_root, "derived") # keep list RDS under derived/

dir.create(bundles\_dir, recursive = TRUE, showWarnings = FALSE)

base\_name <- cfg$save$list\_rds$basename %||% "results\_by\_symbol"

versioned <- file.path(bundles\_dir, sprintf("%s\_%s\_%s.rds", base\_name, pid, ts))

latest <- file.path(bundles\_dir, sprintf("%s\_%s\_latest.rds", base\_name, pid))

saveRDS(objs, versioned)

file.copy(from = versioned, to = latest, overwrite = TRUE, copy.mode = TRUE)

message("Wrote list RDS: ", basename(versioned), " and ", basename(latest))

} else {

message("[SKIP] list RDS: already written in this run.")

}

}

# ----------------------------------------------------------------------------

# 3) Manifest (only if we actually wrote per-symbol files)

# ----------------------------------------------------------------------------

if (isTRUE(wrote\_any\_per\_symbol)) {

make\_manifest(objs, outdir)

}

# ----------------------------------------------------------------------------

# 4) Bundles: intentionally NOT here anymore (call once at end of run\_pipeline)

# ----------------------------------------------------------------------------

# (no-op)

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

}

# --- write selective bundles (and optional combined CSV) ----------------------

make\_bundles <- function(objs, out\_root, bundles\_spec, cfg, ts\_shared = NULL) {

if (is.null(bundles\_spec) || !length(bundles\_spec)) {

message("[BUNDLES] No bundles\_spec provided; skipping.")

return(invisible(NULL))

}

# prevent double writes within one run

if (.pipeline\_flag\_set("write-bundles")) {

message("[SKIP] make\_bundles(): already wrote bundles in this run.")

return(invisible(NULL))

}

pid <- .project\_id(cfg)

ts <- if (!is.null(ts\_shared)) ts\_shared else .ts\_stamp()

# bundles in .../outputs/derived

bundles\_dir <- file.path(out\_root, "derived")

dir.create(bundles\_dir, recursive = TRUE, showWarnings = FALSE)

for (bn in names(bundles\_spec)) {

spec <- bundles\_spec[[bn]]

# Flexible selection:

# - shorthand: results\_app: ["GDPr\_t","TBr\_t",...]

# - or named: results\_app: { include: [...], filters: {...}, csv\_combine: true }

# - or with 'select:' instead of 'include:'

if (is.character(spec)) {

sel\_names <- spec

spec\_filters <- list()

} else {

sel\_names <- (spec$include %||% spec$select %||% character(0))

spec\_filters <- spec$filters %||% list()

}

# keep only symbols we actually have

have <- intersect(unique(sel\_names), names(objs))

if (!length(have)) {

warning("[BUNDLES] Bundle '", bn, "' has no matching symbols in 'objs'; skipping.")

next

}

# apply per-symbol filters if provided

filtered <- lapply(have, function(nm) apply\_filters(objs[[nm]], spec\_filters[[nm]]))

names(filtered) <- have

# bundle files: versioned + latest

versioned <- file.path(bundles\_dir, sprintf("bundle\_%s\_%s\_%s.rds", bn, pid, ts))

latest <- file.path(bundles\_dir, sprintf("bundle\_%s\_%s\_latest.rds", bn, pid))

saveRDS(filtered, versioned)

file.copy(from = versioned, to = latest, overwrite = TRUE, copy.mode = TRUE)

message("[BUNDLES] Wrote ", basename(versioned), " and ", basename(latest))

# optional combined CSV

if (isTRUE(spec$csv\_combine)) {

shape <- tolower(spec$csv\_shape %||% "wide") # reserved for future use

pieces <- mapply(norm\_for\_csv, filtered, names(filtered), SIMPLIFY = FALSE)

csv\_name <- spec$csv\_basename %||% bn

csv\_path <- file.path(bundles\_dir, sprintf("%s\_%s\_%s.csv", csv\_name, pid, ts))

data.table::fwrite(data.table::rbindlist(pieces, use.names = TRUE, fill = TRUE), csv\_path)

message("[BUNDLES] Wrote ", basename(csv\_path))

}

}

invisible(NULL)

}

### Scenarios.R

# R/scenarios.R

# Return cfg (if not provided) using your existing loader

.get\_cfg <- function(cfg = NULL) {

if (is.null(cfg)) load\_config() else cfg

}

# Baseline is the first scenario

base\_scn <- function(cfg = NULL) {

cfg <- .get\_cfg(cfg)

stopifnot(length(cfg$scenarios) >= 1)

cfg$scenarios[[1]]

}

# All policy scenarios (may be length 0, 1, 2, ...)

policy\_scns <- function(cfg = NULL) {

cfg <- .get\_cfg(cfg)

if (length(cfg$scenarios) <= 1) character(0) else cfg$scenarios[-1]

}

# Given a data.frame/data.table, return the scenario columns it actually has

scenario\_cols <- function(x, cfg = NULL) {

cfg <- .get\_cfg(cfg)

intersect(names(x), cfg$scenarios)

}