# Package: MSEtool (via r-universe)

October 15, 2026

**Title** Management Strategy Evaluation Toolkit

**Version** 4.0.0

**Description** Development, simulation testing, and implementation of management procedures for fisheries (see Carruthers & Hordyk (2018) <doi:10.1111/2041-210X.13081>).

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**Depends** R (>= 4.4.0)

**Imports** abind, cli, digest, dplyr (>= 1.1.0), ggplot2, grDevices, lifecycle, lubridate, methods, parallel, patchwork, purrr, Rcpp, rlang, stats, tidyr, utils

**Suggests** boot, broom, covr, crayon, DT, furrr, future, fmsb, kableExtra, knitr, ggrepel, gridExtra, Matrix, MASS, mvtnorm, openxlsx, pak, pbapply, r4ss, readxl, reshape2, rfishbase, rmarkdown, shiny, Slick, snowfall, spict, testthat, tibble, tidyr, TMB, usethis

**LinkingTo** Rcpp, RcppArmadillo

**Remotes** DTUAqua/spict/spict

**BugReports** https://github.com/Blue-Matter/MSEtool/issues

**URL** https://msetool.openmse.com/

**Collate** 'MSEtool-package.R' 'RcppExports.R' 'add-fleet.R' 'add-index.R' 'adjust-seasonal-fecundity.R' 'calc-advice.R' 'calc-age-size-key.R' 'calc-asymptotic-distribution.R' 'calc-blow.R' 'calc-catch-at-size.R' 'calc-conditioned-key.R' 'calc-equilibrium.R' 'calc-fishery-dynamics.R' 'calc-initial-timestep.R' 'calc-innovation-cov.R' 'calc-mgt.R' 'calc-mice.R' 'calc-msy-cpp.R' 'calc-msy.R' 'calc-per-recruit.R' 'calc-pm.R' 'calc-ref-points.R' 'calc-ref-yield.R' 'calc-spatial-movement.R' 'calc-spr0.R'

'calc-survival-unfished.R' 'calc-survival.R'
'calc-unfished-distribution.R' 'calc-unfished-dynamic.R'
'calc-unfished-equilibrium.R' 'calc-unfished-number.R'
'calc-years.R' 'check-advice.R' 'check-data-catch.R'
'check-effort-allocation.R' 'check-fleet-allocation.R'
'check-misc.R' 'check-mse-run.R' 'check-obs.R' 'check-om.R'
'check-package.R' 'check-seasonal-allocation.R'
'class-unions.R' 'class-discardmortality.R' 'class-retention.R'
'class-selectivity.R' 'class-advice.R' 'class-ages.R'
'class-aggbaglimit.R' 'class-bioeconomic.R'
'class-catchability.R' 'class-refpointsMSY.R'
'class-depletion.R' 'class-spatial.R' 'class-srr.R'
'class-fecundity.R' 'class-maturity.R'
'class-naturalmortality.R' 'class-weight.R' 'class-length.R'
'class-stock.R' 'class-data.R' 'class-effort.R'
'class-equilibrium.R' 'class-fleet.R' 'class-popdynamics.R'
'class-reference.R' 'class-timeseries.R' 'class-unfished.R'
'zz_Class_definitions.R' 'class-stocktransition.R'
'class-stock-targeting.R' 'class-imp-slot.R' 'class-imp.R'
'class-obs.R' 'class-om.R' 'class-hist.R' 'class-mse.R'
'class-perrecruit.R' 'class-pm.R' 'class-tune.R'
'collapse-seasons.R' 'combine-fleet-allocation.R'
'combine-fleet-settings.R' 'combine-fleet.R' 'combine-mse.R'
'combine-om.R' 'compare-assessment.R' 'compare-bam.R'
'compare-iscam.R' 'compare-ss3.R' 'condition-obs-catch.R'
'condition-obs-comp.R' 'condition-obs-effort.R'
'condition-obs-index.R' 'condition-obs.R'
'constructor-advice.R' 'constructor-ages.R'
'constructor-aggbaglimit.R' 'constructor-bioeconomic.R'
'constructor-catchability.R' 'generate-data-hist-sizecomp.R'
'constructor-data.R' 'constructor-depletion.R'
'constructor-discardmortality.R' 'constructor-effort.R'
'constructor-fecundity.R' 'constructor-fleet.R'
'constructor-hist.R' 'constructor-imp-slot.R'
'constructor-imp.R' 'constructor-length.R'
'constructor-maturity.R' 'constructor-naturalmortality.R'
'constructor-obs.R' 'constructor-om.R' 'constructor-ref-MSY.R'
'constructor-retention.R' 'constructor-selectivity.R'
'constructor-simcontrol.R' 'constructor-spatial.R'
'constructor-srr.R' 'constructor-stock-targeting.R'
'constructor-stock.R' 'constructor-stocktransition.R'
'constructor-weight.R' 'convert-data.R' 'convert-fleet.R'
'convert-imp.R' 'convert-legacy-mp.R' 'convert-mom.R'
'convert-mse-to-slick.R' 'convert-obs.R' 'convert-om.R'
'convert-om2fleet.R' 'convert-om2stock.R' 'convert-stock.R'
'zz_MOM_object.r' 'convert.R' 'data-trim.R' 'doc-om-control.R'
'example-MPs.R' 'example-fleet.R' 'example-imp.R'
'example-obs.R' 'example-om-herm.R' 'example-om.R'

'example-stock.R' 'extract-F-timeseries.R' 'extract-advice.R'
'extract-bio-timeseries.R' 'extract-catch-timeseries.R'
'extract-effort.R' 'extract-number.R' 'extract-relative.R'
'extract-unfished.R' 'find-model.R' 'fit-sp-spict.R' 'fit-sp.R'
'fit-stock-targeting.R' 'generate-data-hist-agecomp.R'
'generate-data-hist-catch.R' 'generate-data-hist-effort.R'
'generate-data-hist-indices.R' 'generate-data-hist.R'
'generate-data-index-timing.R' 'generate-data-proj-agecomp.R'
'generate-data-proj-catch.R' 'generate-data-proj-effort.R'
'generate-data-proj-helpers.R' 'generate-data-proj-indices.R'
'generate-data-proj-sizecomp.R' 'generate-data-proj.R'
'generate-hist-effort.R' 'generate-mean-at.R'
'generate-multistock-recdevs.R' 'generate-rec-devs.R'
'generate-residuals.R' 'generate-stock-targeting.R'
'get-catch.R' 'helpers-data.R' 'helpers-mp.R' 'hist-to-mse.R'
'import-bam-data.R' 'import-bam.R' 'import-iscam-data.R'
'import-iscam.R' 'import-ss3-data.R' 'import-ss3-report.R'
'import-ss3.R' 'import-wham.R' 'index-nominal.R'
'initialize-hist.R' 'interim-advice.R' 'models-fecundity.R'
'models-length.R' 'models-maturity.R'
'models-natural-mortality.R' 'models-retention.R'
'models-selectivity.R' 'models-srr.R' 'models-weight.R'
'models.R' 'mp-args.R' 'mp-empirical-index.R'
'mp-hockeystickHCR.R' 'mp-reference.R'
'mp-surplus-production.R' 'mp-utils.R' 'mse-update.R'
'opt-effort-choke.R' 'opt-effort-helpers.R'
'opt-effort-single-stock.R' 'opt-final-depletion.R'
'opt-ref-yield.R' 'opt-targeting-helpers.R'
'opt-targeting-multi-stock.R' 'plot-alk.R' 'plot-awk.R'
'plot-catchability.R' 'plot-closure.R' 'plot-comp.R'
'plot-data.R' 'plot-depletion.R' 'plot-fleet.R' 'plot-hist.R'
'plot-index-fit.R' 'plot-recdevs.R' 'plot-schedule.R'
'plot-spatial.R' 'plot-srr.R' 'plot-stock.R' 'pm-years.R'
'populate-ASK.R' 'populate-catchability.R' 'populate-classes.R'
'populate-closure.R' 'populate-depletion.R'
'populate-discard-mortality.R' 'populate-effort.R'
'populate-fecundity.R' 'populate-fleet.R' 'populate-imp.R'
'populate-length.R' 'populate-maturity.R' 'populate-mean-at.R'
'populate-natural-mortality.R' 'populate-obs-catch.R'
'populate-obs-comp.R' 'populate-obs-effort.R'
'populate-obs-helpers.R' 'populate-obs-index.R'
'populate-obs.R' 'populate-om.R' 'populate-random.R'
'populate-retention.R' 'populate-selectivity.R'
'populate-spatial.R' 'populate-srr.R' 'populate-stock.R'
'populate-weight.R' 'populate.R' 'ppd-access.R' 'print.R'
'project-hist.R' 'project-mp.R' 'project.R' 'reduce-nSim.R'
'reduce.R' 'rename.R' 'reorder.R' 'runMSE.R'
'set-seasonal-R0.R' 'show.R' 'simulate-om.R' 'simulate.R'

'standardize-effort.R' 'subset.R' 'summarize-convergence.R'
'tune-mp.R' 'unit-conversions.R' 'update-baglimit.R'
'update-closure.R' 'update-discardmortality.R'
'update-effort.R' 'update-object.R' 'update-selectivity.R'
'update-tac-helpers.R' 'update-tac.R' 'util-accessors.R'
'util-add-dimension.R' 'util-array-create.R'
'util-convert-to-list.R' 'util-dim.R' 'util-get-metadata.R'
'utils-add.R' 'utils-advice-null.R' 'utils-array-operations.R'
'utils-array-to-df.R' 'utils-atage-atsize.R'
'utils-calc-ageclasses.R' 'utils-check-hist-misc.R'
'utils-cli.R' 'utils-compdata.R' 'utils-cr2h.R'
'utils-dirichlet-multinomial.R' 'utils-empty-object.R'
'utils-extend-hist.R' 'utils-extend-om.R' 'utils-extend.R'
'utils-fill-from-misc.R' 'utils-get_nSim.R' 'utils-helpers.R'
'utils-herm.R' 'utils-identical-age.R' 'utils-identical-sim.R'
'utils-join-year.R' 'utils-list-to-array.R' 'utils-load-args.R'
'utils-log-mp-error.R' 'utils-log.R' 'utils-logs-delete.R'
'utils-make-named-list.R' 'utils-management-schedule-table.R'
'utils-misc.R' 'utils-msy-accessors.R' 'utils-parallel.R'
'utils-prep-hist-misc.R' 'utils-recycle-fleets.R'
'utils-refpoints-accessors.R' 'utils-reverse-list.R'
'utils-save.R' 'utils-set-age-dimnames.R' 'utils-slot-sizes.R'
'utils-startup.R' 'utils-stocknames.R' 'utils-structure.R'
'utils-transformation.R' 'utils-unique-years.R'
'utils-unit-conversion.R' 'utils-valid-units.R'
'utils-wrapped-normal.R' 'validate-shift.R' 'zz_ASAP2OM.R'
'zz_Awatea2OM.R' 'zz_BAM2OM.R' 'zz_B_Low_functions.R'
'zz_DFO.R' 'zz_Data2csv.R' 'zz_Data_Functions.R'
'zz_Data_Plotting.R' 'zz_Data_fit_index.R'
'zz_Data_make_update.R' 'zz_Empirical_MP.R'
'zz_Fease_Functions.R' 'zz_MICE_Rel.R' 'zz_MMSE_Object.r'
'zz_MMSE_functions.R' 'zz_MMSE_plots.R' 'zz_MOM_import.R'
'zz_MOM_plots.R' 'zz_MP_tuning.R' 'zz_MPs_Reference.R'
'zz_MSC.R' 'zz_MSE_Plotting.R' 'zz_MSE_Plotting_PMs.R'
'zz_MSE_functions.R' 'zz_Misc_Exported.R' 'zz_Misc_Internal.R'
'zz_Movement_calcs.R' 'zz_OM_Plotting.R' 'zz_OM_functions.R'
'zz_OM_import.R' 'zz_OM_init_doc.R' 'zz_OM_plots.R'
'zz_PMobjects.R' 'zz_PMtable.R' 'zz_Parallel.R' 'zz_SAM2OM.R'
'zz_SS2Data.R' 'zz_SS2MOM.R' 'zz_SS2MOM_plots.R' 'zz_SS2OM.R'
'zz_SSinternal.R' 'zz_SampleOM.R' 'zz_Sketch_Functions.R'
'zz_Splot.R' 'zz_Turing.R' 'zz_VPA2OM.R' 'zz_WHAM2OM.R'
'zz_aaImportFrom.R' 'zz_applyMMP.R' 'zz_data_documentation.R'
'zz_fit-sr.R' 'zz_iSCAM2OM.R' 'zz_multiMSE.R'
'zz_multiMSE_internal.R' 'zz_multi_q_estimation.R'
'zz_popdyn.R' 'zz_popdynOneMICEDEV.R' 'zz_popdyn_MICE.R'
'zz_project_Hist.R' 'zz_runMSE.R' 'zz_simulate-OM.R'
'zz_spatial_targ_opt.R' 'zz_utils.R'

**Config/roxygen2/version** 8.1.0

**Config/testthat/edition** 3

**Config/pak/sysreqs** libicu-dev

**Repository** https://blue-matter.r-universe.dev

**Date/Publication** 2026-10-15 02:36:05 UTC

**RemoteUrl** https://github.com/Blue-Matter/MSEtool

**RemoteRef** HEAD

**RemoteSha** ca5020003a6a302fe5583d18f8d4280e788c2451

# Contents

.AddAutoCorrelation . . . 15
Access . . . 16
AddFleet . . . 21
AddIndex . . . 22
addMMPs . . . 24
AdjustSeasonalFecundity . . . 25
Advice . . . 26
advice-class . . . 31
advicedata-class . . . 32
Ages . . . 33
ages-class . . . 35
AgeStructuredObs . . . 36
aggbaglimit-class . . . 37
AggregateBagLimit . . . 38
Albacore . . . 40
Albacore_TwoFleet . . . 41
AlbacoreExStock . . . 41
AnnualData . . . 44
ApplyAC . . . 45
applyMMP . . . 46
applyMP . . . 46
array_df_conversion . . . 47
ArrayOperations . . . 48
ASAP2OM . . . 50
Assess2MOM . . . 52
Assess2OM . . . 55
AsympExFleet . . . 58
Atlantic_mackerel . . . 60
avail . . . 61
Awatea2OM . . . 62
BAM2MOM . . . 63
BAMFleetNames . . . 64
bio_timeseries . . . 65
Bioeconomic . . . 67
bioeconomic-class . . . 70
boxplot.Data . . . 70

ButterfishExStock . . . 71
CalcAgeClasses . . . 73
CalcAgeSizeKey . . . 75
CalcAsymDist . . . 77
CalcBLow . . . 78
CalcEquilibrium . . . 79
CalcIndexResiduals . . . 80
CalcInnovationCov . . . 81
CalcMGT . . . 82
CalcMSY . . . 83
CalcPerRecruit . . . 85
CalcRefPoints . . . 86
calcRefYield . . . 87
CalcResidualStats . . . 88
CalcSPR0 . . . 89
CalcSurvival . . . 90
CalcTSUnits . . . 91
CalcUnfished_Dynamic . . . 92
CalcUnfished_Equilibrium . . . 93
CalcUnfishedDist . . . 94
CalcUnfishedNumber . . . 95
CalcUnfishedSurvival . . . 95
CALsimp . . . 96
Can . . . 97
catch_timeseries . . . 98
Catchability . . . 101
catchability-class . . . 105
CatchAndSurveyObs . . . 106
CatchData . . . 107
catchdata-class . . . 108
CatchFrac . . . 109
CatchObs . . . 110
catchobs-class . . . 113
CheckCatch . . . 114
CheckDuplicate . . . 115
CheckMPs . . . 115
checkMSE . . . 116
CheckOM . . . 117
CheckPackage . . . 118
Choose . . . 118
CollapseSeasons . . . 120
CombineFleets . . . 121
CombineMMP . . . 124
CombineMSE . . . 124
CombineOMs . . . 125
CommercialFleetObs . . . 127
CompareBAM . . . 128
CompareiSCAM . . . 129

CompareSS . . . 130
CompData . . . 132
compdata-class . . . 134
ComplexOM . . . 135
CompObs . . . 137
compobs-class . . . 140
ConstrainTAC . . . 141
Converge . . . 142
Convert . . . 144
ConvertData . . . 145
ConvertFleet . . . 147
ConvertImp . . . 149
ConvertMOM . . . 149
ConvertObs . . . 150
ConvertOM . . . 152
ConvertStock . . . 153
Cos_thresh_tab . . . 154
cparscheck . . . 155
Cplot . . . 156
CR2h . . . 157
Data . . . 158
data-class . . . 160
Data-legacy-class . . . 162
Data_xl . . . 166
Data2csv . . . 167
DataDir . . . 168
DataHelpers . . . 168
DataInit . . . 170
DataLag . . . 171
DataRichObs . . . 173
DataSlots . . . 174
DataTrim . . . 174
DecE_Dom . . . 175
DefaultAges . . . 176
DefaultSlickPMs . . . 177
DeleteLogs . . . 178
Depletion . . . 179
depletion-class . . . 182
DFO_bar . . . 183
DFO_hist . . . 184
DFO_plot . . . 184
DFO_plot2 . . . 185
DFO_proj . . . 185
DFO_quant . . . 186
DFO_report . . . 187
DFO_spider . . . 187
DFO_tab . . . 188
DFO_tab_formatted . . . 188

dim . . . 189
dimension-utils . . . 190
DiscardMortality . . . 191
discardmortality-class . . . 195
DLMDataDir . . . 196
doHCR . . . 196
doIfreq . . . 197
Dom . . . 198
DomeExFleet . . . 199
doRec . . . 201
Effort . . . 202
effort-class . . . 210
EffortAllocation . . . 211
EffortData . . . 213
effortdata-class . . . 214
EffortObs . . . 215
effortobs-class . . . 217
Emp . . . 218
EmptyObject . . . 220
equilibrium-class . . . 221
EstimateBeta . . . 222
ExampleMPs . . . 223
exploitationdata-class . . . 225
ExploitationObs . . . 226
exploitationobs-class . . . 227
Extend . . . 228
extract-unfished . . . 231
Fease . . . 232
Fecundity . . . 233
fecundity-class . . . 239
Fecundity-Models . . . 241
FilterTAC . . . 243
FInteract . . . 244
FitMovement . . . 246
FitSP . . . 247
FitStockTargeting . . . 250
Fleet . . . 252
fleet-class . . . 256
Fleet-legacy-class . . . 258
FMSYref . . . 260
FullComplianceImp . . . 262
GenerateStockTargeting . . . 263
Generic_Obs . . . 265
GenHistEffort . . . 265
GenMeanAtLength . . . 266
GenMultiStockRecDevs . . . 267
GenRecDevs . . . 268
GenResiduals . . . 270

getclass . . . 271
getDataList . . . 272
getfirstlev . . . 272
GetMetaData . . . 273
getmov2 . . . 274
getMP . . . 275
getnIVs . . . 275
getsel . . . 276
GetSSRefPoints . . . 276
hconv . . . 277
Herm . . . 278
Herm-int . . . 280
HermOM . . . 281
Hist . . . 283
hist-class . . . 284
Hist-legacy-class . . . 285
hist2 . . . 288
HistoricalWeight . . . 288
HockeyStickHCR . . . 290
Imp . . . 292
imp-class . . . 294
Imp-legacy-class . . . 295
ImportBAM . . . 297
ImportBAMData . . . 299
ImportiSCAM . . . 302
ImportiSCAMData . . . 304
ImportSS . . . 305
ImportSSData . . . 308
ImportSSReport . . . 309
ImpSlot . . . 310
IndexFitTable . . . 311
IndexMPs . . . 313
IndexSelectivity . . . 318
IndicesData . . . 319
indicesdata-class . . . 320
IndicesObs . . . 321
indicesobs-class . . . 326
initialize-methods . . . 327
Input . . . 327
InterimAdvice . . . 328
Interval . . . 330
iSCAM . . . 332
iSCAM2OM . . . 333
iSCAMcomps . . . 336
iSCAMinds . . . 336
isNewObject . . . 337
join_plots . . . 338
joinData . . . 338

JoinYear . . . 339
Kplot . . . 340
Lag_Data . . . 341
ldim . . . 342
Length . . . 343
Length-at-Age-Models . . . 347
length-class . . . 349
LengthStructuredObs . . . 351
LH2OM . . . 352
LHdatabase . . . 354
lifehistorydata-class . . . 354
LifeHistoryObs . . . 355
lifehistoryobs-class . . . 357
list-array-conversion . . . 358
ListParallelWorkers . . . 359
LoadArgs . . . 360
Log . . . 361
logit . . . 362
makeMOM . . . 363
makemov . . . 364
makemov2 . . . 364
MakeNamedList . . . 365
makeRel . . . 366
makeTransparent . . . 368
ManagementScheduleTable . . . 368
Maturity . . . 369
maturity-class . . . 376
Maturity-Models . . . 378
ML2D . . . 380
MMSE-legacy-class . . . 380
model-helpers . . . 382
MOM-legacy-class . . . 384
movestockCPP . . . 386
movfit_Rcpp . . . 386
MPCalcsNAs . . . 387
MPFailureRate . . . 388
MPStartYear . . . 388
MPtype . . . 390
mse-class . . . 390
MSE-legacy-class . . . 391
MSE2Slick . . . 393
MSEextra . . . 397
MSY-accessors . . . 397
MSYCalcs . . . 398
multiData . . . 400
multiDataS . . . 400
multidebug . . . 401
MultiStockOM . . . 402

name-accessors . . . 404
NaturalMortality . . . 405
naturalmortality-class . . . 410
NaturalMortality-Models . . . 411
NewPM . . . 413
NIL . . . 414
NOAA_plot . . . 415
Number . . . 415
Obs . . . 416
obs-class . . . 419
Obs-legacy-class . . . 420
OM . . . 425
OM-accessors . . . 430
om-class . . . 433
OM-legacy-class . . . 436
OMControl . . . 437
OMdoc . . . 439
OMexample . . . 441
OMinit . . . 441
OverageImp . . . 442
Overages . . . 443
PartialSizeComplianceImp . . . 444
PerformanceMetric . . . 445
perrecruit-class . . . 446
plot.Data . . . 448
plot.MMSE . . . 449
plot.MOM . . . 450
plot.MSE . . . 450
plot.pars . . . 451
plot_comp . . . 455
plot_data . . . 458
plot_hist . . . 460
plot_schedule . . . 468
PlotALK . . . 470
PlotAWK . . . 471
PlotCatchability . . . 472
PlotClosure . . . 473
PlotCompShift . . . 474
PlotDepletion . . . 475
PlotDiscardMortality . . . 476
PlotEffort . . . 478
PlotFecundity . . . 480
PlotFleet . . . 481
plotFun . . . 483
PlotIndexFit . . . 483
PlotLength . . . 485
PlotMaturity . . . 486
plotmulti . . . 488

PlotNaturalMortality . . . 488
plotOFL . . . 490
plotquant . . . 490
PlotRecDevs . . . 491
plotRel . . . 492
PlotRetention . . . 493
PlotSelectivity . . . 494
PlotSpatial . . . 496
PlotSRR . . . 497
PlotStock . . . 499
PlotWeight . . . 500
PM . . . 501
pm-class . . . 506
PM-equations . . . 507
PMLimit . . . 510
PMobj-class . . . 512
PMYears . . . 513
popdynamics-class . . . 514
Populate . . . 514
PopulateCatchability . . . 516
PopulateCatchObs . . . 517
PopulateClosure . . . 518
PopulateCompObs . . . 519
PopulateDepletion . . . 521
PopulateDiscardMortality . . . 522
PopulateEffort . . . 523
PopulateEffortObs . . . 525
PopulateFecundity . . . 526
PopulateFleet . . . 527
PopulateImp . . . 529
PopulateImpSlot . . . 529
PopulateIndexObs . . . 530
PopulateLength . . . 532
PopulateMaturity . . . 533
PopulateNaturalMortality . . . 535
PopulateObs . . . 536
PopulateOM . . . 538
PopulateRetention . . . 539
PopulateSelectivity . . . 541
PopulateSpatial . . . 543
PopulateSRR . . . 544
PopulateStock . . . 545
PopulateWeight . . . 547
PPD . . . 549
Pplot . . . 549
Pplot2 . . . 550
print.multiHist . . . 552
print.PlotFleetList . . . 552

print.PlotStockList . . . 553
print.RepList . . . 553
Project . . . 554
PWhisker . . . 555
quantile_plot . . . 556
rDirichletMultinomial . . . 557
RealFease . . . 558
Rec-class . . . 559
ReduceDims . . . 560
ReduceNSim . . . 561
reference-class . . . 562
referencedata-class . . . 562
ReferenceMPs . . . 563
refpoints-accessors . . . 564
refpointsMSY-class . . . 566
RefSeason . . . 566
relative_ref . . . 568
Rename . . . 571
Reorder . . . 572
Replace . . . 573
replic8 . . . 574
Report . . . 575
ReqData . . . 576
Required . . . 576
Retention . . . 577
retention-class . . . 581
Retention-Models . . . 582
runCOSEWIC . . . 585
runInMP . . . 588
runMP . . . 589
runMSE . . . 589
S4SlotSizes . . . 592
Save . . . 593
SeasonalSpatialExStock . . . 594
SeasonalSpatialOM . . . 596
select_MP . . . 598
Selectivity . . . 599
selectivity-class . . . 603
Selectivity-Models . . . 604
Sense . . . 607
SetMPArgs . . . 608
SetSeasonalR0 . . . 609
setup . . . 610
SetupParallel . . . 611
show-methods . . . 613
show-MSEtool . . . 613
show,PMobj-method . . . 614
show,Rec-method . . . 614

SIL . . . 614
simCAA . . . 615
simCAL . . . 616
SimControl . . . 617
simmov . . . 619
simmov2 . . . 620
Simulate . . . 622
SimulatedData . . . 624
SimulateMOM . . . 624
SingleStockOM . . . 626
SketchFun . . . 627
SmoothSeries . . . 628
smoothy . . . 629
Spatial . . . 630
spatial-class . . . 637
SPControl . . . 638
SPEstimates . . . 640
SpictControl . . . 641
Splot . . . 644
SRR . . . 644
srr-class . . . 652
SRRModels . . . 654
SRRSteepness . . . 657
SS2Data . . . 658
SS2DataMOM . . . 660
SS2MOM . . . 661
SSBrefplot . . . 665
StandardizeEffort . . . 665
Stock . . . 667
stock-class . . . 671
Stock-legacy-class . . . 673
StockTargeting . . . 676
stocktargeting-class . . . 677
stocktransition-class . . . 678
Sub . . . 679
SubCpars . . . 680
SubOM . . . 681
Subset . . . 681
SummarizeConvergence . . . 683
summary,Data-method . . . 684
summary,MMSE-method . . . 685
summary,MSE-method . . . 685
SumOverAge . . . 686
SurplusProduction . . . 687
TAC . . . 692
TACfilter . . . 693
TACs . . . 694
Taxa_Table . . . 695

TEG . . . 695
testOM . . . 696
Thresh_tab . . . 696
timeseries-class . . . 697
tinyErr . . . 698
TradePlot . . . 699
tune_MP . . . 701
TuneControl . . . 702
TuneMP . . . 704
tunemp-class . . . 707
TuneObjective . . . 708
TuneTable . . . 710
Turing . . . 712
TwoFleetOM . . . 713
UnderageImp . . . 715
unfished-class . . . 716
unit-conversions . . . 717
UpdateObject . . . 718
Uses . . . 719
validcpars . . . 719
ValidUnits . . . 721
VOI . . . 722
VOI2 . . . 723
VOIplot . . . 723
Weight . . . 725
weight-class . . . 730
Weight-Models . . . 732
WHAM2OM . . . 733
wormplot . . . 735
WrappedNormal . . . 736
WrasseExStock . . . 737
writeCSV . . . 738
XL2Data . . . 739
XL2Fleet . . . 739
XL2OM . . . 740
XL2Stock . . . 741
Years . . . 742

**Index** **744**

---

`.AddAutoCorrelation` *Add Auto-Correlation to a Vector of Values*

---

## Description

Applies an AR(1) auto-correlation structure to a vector of independent values, conditioning on a provided last observed value. The transformation scales each value to preserve the marginal variance under the AR(1) process.

### Usage

```
.AddAutoCorrelation(values, ac, last_value)
```

### Arguments

| | |
|---|---|
| `values` | `numeric` vector. Independent values to be transformed with auto-correlation (e.g., log recruitment deviations). |
| `ac` | `numeric` scalar. Auto-correlation coefficient, typically in [-1, 1]. |
| `last_value` | `numeric` scalar. The last observed value from the preceding time period, used to condition the first element of `values`. |

The AR(1) recursion applied is:

$$x_t = \rho \cdot x_{t-1} + \epsilon_t \cdot \sqrt{1 - \rho^2}$$

where $\rho$ is `ac` and $\epsilon_t$ are the input `values`. The $\sqrt{1 - \rho^2}$ scaling ensures the marginal variance of $x_t$ equals the variance of $\epsilon_t$.

### Value

A `numeric` vector of the same length as `values` with AR(1) auto-correlation applied.

### See Also

`GenMultiStockRecDevs()`

---

## `Access` *Access and Assign Slots from Compatible S4 Objects*

---

### Description

Generic accessor and replacement functions to retrieve and assign named slots from compatible S4 objects across the `openMSE` package ecosystem. Functions work on any S4 object that contains the corresponding slot.

### Usage

```
AC(x)

AC(x) <- value

Areas(x)

Areas(x) <- value

Beta(x)

Beta(x) <- value
```

```
Bias(x)

Bias(x) <- value

Caption(x)

Caption(x) <- value

Classes(x)

Classes(x) <- value

Compliance(x)

Compliance(x) <- value

CPUE(x)

CPUE(x) <- value

CV(x)

CV(x) <- value

CVatAge(x)

CVatAge(x) <- value

Dist(x)

Dist(x) <- value

DiscardsAtAge(x)

DiscardsAtAge(x) <- value

DiscardsAtSize(x)

DiscardsAtSize(x) <- value

ESS(x)

ESS(x) <- value

Error(x)

Error(x) <- value
```

```
Exploitation(x)

Exploitation(x) <- value

LandingsAtAge(x)

LandingsAtAge(x) <- value

LandingsAtSize(x)

LandingsAtSize(x) <- value

LifeHistory(x)

LifeHistory(x) <- value

Mean(x)

Mean(x) <- value

MeanAtAge(x)

MeanAtAge(x) <- value

MeanAtLength(x)

MeanAtLength(x) <- value

MeanAtWeight(x)

MeanAtWeight(x) <- value

Misc(x)

Misc(x) <- value

MPs(x)

MPs(x) <- value

Model(x)

Model(x) <- value

Name(x)

Name(x) <- value
```

```
nSim(x)

nSim(x) <- value

nArea(x, st = 1)

nAge(x, st = NULL)

nComplex(x)

nStock(x)

nFleet(x)

Pars(x)

Pars(x) <- value

Period(x)

Period(x) <- value

Prob(x)

Prob(x) <- value

Random(x)

Ref(x)

Ref(x) <- value

RefCV(x)

RefCV(x) <- value

Reference(x)

Reference(x) <- value

SampleSize(x)

SampleSize(x) <- value

Size(x)

Size(x) <- value
```

```
SD(x)

SD(x) <- value

Stat(x)

Stat(x) <- value

Shift(x)

Shift(x) <- value

Stats(x)

Stats(x) <- value

Survey(x)

Survey(x) <- value

Timing(x)

TruncSD(x)

TruncSD(x) <- value

Type(x)

Type(x) <- value

Units(x)

Units(x) <- value

Value(x)

Value(x) <- value

YearLH(x)

YearLH(x) <- value
```

**Arguments**

| | |
|---|---|
| `x` | An S4 object with the corresponding slot. |
| `value` | The value to assign to the slot |
| `st` | Integer. Stock index used when `x` contains multiple stocks. Default `1` for |

`nArea()` and `NULL` for `nAge()` (returns all stocks as a list when `NULL`).

### Value

- Accessor functions return the value stored in the named slot of `x`. For `Stat()`, `Prob()`, and `Mean()` on a pm object, the named `Sim x Stock x MP` (or `Stock x MP`) array is converted to a tidy data frame via `Array2DF()` instead of being returned as a raw array.
- Replacement functions return `x` with the named slot updated to `value`.

### Examples

```
MyLength <- Length()
MeanAtAge(MyLength)
```

---

## AddFleet *Add a Fleet to an Operating Model*

---

### Description

Adds a new fleet to an om object, appending a fleet, an obs, and an imp object for every stock in the OM.

### Usage

```
AddFleet(OM, FleetName, Fleet = NULL, Obs = NULL, Imp = NULL)
```

### Arguments

| | |
|---|---|
| `OM` | An object of class `om`. |
| `FleetName` | A length-1 character string giving the name of the new fleet. Must not already exist in `OM`. |
| `Fleet` | Optional. An object of class `fleet`. If `NULL` (default), an empty `fleet` object is created with `Name = FleetName`. |
| `Obs` | Optional. An obs object. If `NULL` (default), an empty `obs` object is created with `Name = FleetName`. |
| `Imp` | Optional. An imp object. If `NULL` (default), an empty `imp` object is created with `Name = FleetName` |

### Value

The `om` object with the new fleet appended to `OM@Fleet` and `OM@Obs` for every stock.

### Examples

```
OM <- SingleStockOM
FleetNames(OM)
OM2 <- AddFleet(OM, 'DummyFleet')
FleetNames(OM2)
```

---

## AddIndex *Add an Abundance Index to an Operating Model*

---

### Description

Adds an observed CPUE or survey index to an om object, together with its observation model: the index values are appended to `OM@Data[[Complex]]@Survey` (or `@CPUE`), and an indicesobs object with the index's `Selectivity` is placed in `OM@Obs[[Complex]][[Name]]`.

### Usage

```
AddIndex(
  OM,
  Value,
  Name = NULL,
  CV = NULL,
  Units = c("Biomass", "Number", "Recruitment"),
  Timing = 0,
  Selectivity = NULL,
  Type = c("Survey", "CPUE"),
  Complex = 1,
  IndexObs = IndicesObs()
)

AddAgeIndices(
  OM,
  Value,
  Name = "Survey",
  CV = NULL,
  Units = c("Number", "Biomass"),
  Timing = 0,
  Type = c("Survey", "CPUE"),
  Complex = 1,
  IndexObs = IndicesObs()
)
```

### Arguments

| | |
|---|---|
| `OM` | An om object. |

| | |
|---|---|
| Value | For AddIndex(), a numeric vector of index values named by year, or a one-column matrix with years as row names. For AddAgeIndices(), a numeric matrix with years as rows and age classes as columns. Years must be in Years(OM, "Historical"); missing years are NA. |
| Name | character(1). Name of the index. For AddIndex(), defaults to the column name of a one-column matrix Value. A CPUE index named after a fleet in OM@Fleet uses that fleet's selectivity unless Selectivity is given. |
| CV | NULL, a scalar, or values matching Value. Coefficient of variation of each observation. Default NULL. |
| Units | character(1). How the population is converted to the index: "Biomass", "Number", or "Recruitment" (AddIndex(), default "Biomass"); "Number" or "Biomass" (AddAgeIndices(), default "Number"). See IndicesObs(). |
| Timing | numeric(1). Timing of the observations as a fraction of the time step (0-1). Default 0. |
| Selectivity | Selectivity-at-age of the index: "Biomass", "SBiomass", a numeric vector named by age, an array, or a list of these by stock. See IndicesObs() for details. Required unless the index is named after a fleet. |
| Type | character(1). "Survey" (default) for fishery-independent indices, or "CPUE" for fishery-dependent indices. |
| Complex | Integer. Index of the stock complex (OM@Data element) the index belongs to. Default 1. |
| IndexObs | An indicesobs object with other observation model settings for the index, e.g. CV, Beta, or Areas. Default IndicesObs(). For AddIndex(), give Selectivity either here or as the Selectivity argument, not both. For AddAgeIndices(), used as the template for every age, and Selectivity must not be set. |

## Details

AddAgeIndices() adds an index observed by age class (e.g., survey numbers-at-age) as one index per age class, named "<Name> Age <age>", each with a selectivity of 1 for that age and 0 for all others. Each age-specific index is conditioned separately, so each has its own Efficiency, Beta, residual CV, and autocorrelation.

## Value

OM with the index (or indices) added to OM@Data and OM@Obs.

## See Also

IndicesData(), IndicesObs()

## Examples

```
OM <- SingleStockOM
Years <- Years(OM, "Historical")
```

```
# spawning biomass survey
Index <- setNames(runif(length(Years)), Years)
OM <- AddIndex(OM, Index, Name = "Acoustic", Selectivity = "SBiomass", CV = 0.2)

# survey of ages 3-5
OM <- AddIndex(OM, Index, Name = "Trawl", Units = "Number",
               Selectivity = c("3" = 1, "4" = 1, "5" = 1))

# survey numbers-at-age, one index per age
Ages <- 2:5
Value <- matrix(runif(length(Years) * length(Ages)), length(Years), length(Ages),
                dimnames = list(Year = Years, Age = Ages))
OM <- AddAgeIndices(OM, Value, Name = "Longline Survey", CV = 0.3)

OM@Data[[1]]@Survey@Name
```

---

`addMMPs` *Utility functions for multiMSE objects*

---

## Description

`addMMPs()` adds additional management procedures to a MMSE object by combining multiple MMSE objects that have identical historical OM values, and population/ fleet structures.

## Usage

```
addMMPs(MMSEobjs)
```

## Arguments

| | |
|---|---|
| `MMSEobjs` | A list `MMSE` object |

## Value

An object of class `MMSE`

## Author(s)

Q. Huynh

## See Also

`addMPs()`

`AdjustSeasonalFecundity` *Adjust Fecundity to Reflect Seasonal Spawning Activity*

## Description

Scales `Fecundity@MeanAtAge` for each stock in an om object so that seasonal spawning production in model outputs is concentrated in the same seasons as recruitment, matching the biological reality that fish actively spawn only during certain months.

## Usage

```
AdjustSeasonalFecundity(OM, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `OM` | An om object. `Stock@SRR@R0` should already encode the seasonal recruitment pattern (e.g. via `SetSeasonalR0()`). |
| `silent` | Logical. Suppress messages. Default `FALSE`. |

## Details

Seasonal recruitment is controlled entirely by `SRR@R0`: at unfished equilibrium the stock-recruit relationship returns exactly `R0[m]` recruits in season `m`, regardless of how fecundity is distributed across seasons. Use `SetSeasonalR0()` to encode the target seasonal proportions into `R0` before calling this function.

What this function adds is biological realism in spawning output. Without it, fecundity (and therefore reported spawning production) is spread uniformly across all seasons even when recruitment is highly concentrated, a scenario that is not biologically feasible. Fish that do not spawn in a given season contribute no eggs regardless of their maturity, and if they did spawn but consistently produced fewer recruits than expected, natural selection would have eliminated that strategy over evolutionary time.

Applying a per-season spawning activity fraction `phi_m` to fecundity concentrates spawning production in the same seasons as recruitment, representing the fraction of mature fish that are actively spawning each season.

## Value

The om object with `Fecundity@MeanAtAge` updated for all stocks with non-uniform seasonal recruitment.

## See Also

`SetSeasonalR0()`, `WrappedNormal()`, `CalcUnfishedNumber()`

`Advice` *Management Advice Constructor and Accessors*

## Description

Construct an advice object defining management advice returned by a management procedure.

## Usage

```
Advice(
  TAC = NULL,
  TACType = "Removals",
  TACUnit = "Biomass",
  Effort = NULL,
  EffType = "Rel",
  Closure = NULL,
  Selectivity = NULL,
  Retention = NULL,
  DiscardMortality = NULL,
  BagLimit = NULL,
  LimitType = "angler",
  ClosureMode = "discard",
  Misc = list()
)

TAC(x) <- value

TACType(x)

TACType(x) <- value

TACUnit(x)

TACUnit(x) <- value

EffType(x)

EffType(x) <- value

BagLimit(x)

BagLimit(x) <- value

LimitType(x)

LimitType(x) <- value
```

```
ClosureMode(x)

ClosureMode(x) <- value
```

### Arguments

| | |
|---|---|
| `TAC` | Numeric vector or matrix specifying total allowable catch. See `Details` |
| `TACType` | Character. Does the TAC refer to `"Removals"` (default) or `"Landings"`. Either length 1 (applied to all fleets) or a character vector of length `nFleet`. |
| `TACUnit` | Character. Units in which the TAC is expressed: `"Biomass"` (default), `"Number"`, or a catch unit such as `Data@Landings@Units` (e.g. `"t"`, `"kg"`, `"n"`), which is stored as `"Biomass"` or `"Number"`. The TAC is not rescaled, so a TAC in biomass is in the OM's `Weight` units. Either length 1 (applied to all fleets) or a character vector of length `nFleet`. |
| `Effort` | Numeric vector or matrix specifying relative or absolute fishing effort. See `Details` |
| `EffType` | Character. Effort interpretation: `"Rel"` for relative to the last historical year, or `"Abs"` for absolute effort units (in units of `Effort()`). Either length 1 (applied to all fleets) or a character vector of length `nFleet`. Default is `"Rel"`. |
| `Closure` | Numeric vector or array specifying spatial closures. See `Details` |
| `Selectivity` | A `Selectivity()` object or an `nFleet`-length list of such objects. See `Details` |
| `Retention` | A `Retention()` object or an `nFleet`-length list of such objects. See `Details` |
| `DiscardMortality` | A `DiscardMortality()` object or an `nFleet`-length list of such objects. See `Details` |
| `BagLimit` | Numeric vector or `NULL`. Bag limit in fish per angler per trip (when `LimitType = "angler"`) or fish per vessel per trip (when `LimitType = "boat"`), for this stock's catch by each fleet. Accepted forms:<br>• `NULL` (default): no bag limit regulation is active for any fleet.<br>• Single numeric value: the same limit applied to all fleets.<br>• Numeric vector of length `nFleet`: fleet-specific bag limits. `NA` for a given fleet position means no limit applies to that fleet.<br>The bag limit regulation persists across time steps until a new `Advice()` object is returned by the MP. See Details.<br>An aggregate bag limit pooling catch across several stocks for one fleet is declared separately via `AggregateBagLimit()`, available to `mmp`-class management procedures. A stock's own `BagLimit` then acts as an optional species-specific sub-cap within that pooled limit for any stock included in the group. |

`LimitType` Character or `NULL`. Whether `BagLimit` is a per-angler or per-vessel regulation. Either length 1 (applied to all fleets) or a character vector of length `nFleet`. Valid values:

- `"angler"` (default): limits are per angler per trip; the fleet-level retention cap is $B_f \cdot A_{f,t} \cdot T_{f,t}$, where $A_{f,t}$ is mean anglers per trip (`AnglerPerTrip()`) and $T_{f,t}$ is the number of trips (`TripsScalar()`).
- `"boat"`: limits are per vessel per trip; the fleet-level retention cap is $B_f \cdot T_{f,t)}$, and `AnglerPerTrip()` is not used.

`ClosureMode` Character or `NULL`. Determines how the OM handles catch that exceeds the bag limit. Either length 1 (applied to all fleets) or a character vector of length `nFleet`. Valid values:

- `"discard"` (default): catch exceeding the bag limit is converted to discards; discard mortality is applied via the fleet's `DiscardMortality()` object.
- `"stop"`: effort is reduced so that projected catch does not exceed the bag limit retention cap; no additional discards are generated beyond the baseline discard rate from non-retention of undersized fish.

`Misc` Miscellaneous list. Will be passed to `Data@Misc` in following time steps.

`x` An advice object, for the accessor and replacement functions below.

`value` The replacement value, for the replacement functions below.

## Details

`Advice()` constructs a container for management regulations returned by a management procedure. All slots are optional; any regulation not supplied will remain unchanged from the previous time step.

The management regulations in an `Advice` object will be applied in the year/time-step that the `Advice` object is produced (i.e., when the MP is run) and will apply to all future time-steps until a new `Advice` object is returned by an MP.

The `Closure`, `Selectivity`, `Retention`, and `DiscardMortality` regulations (if any) are applied first before calculating the fishing mortality corresponding with `TAC`.

If both `TAC` and `Effort` regulations are returned, the expected fishing effort will be calculated to achieve a catch equal to the `TAC`. If the expected effort is greater than that set in `Effort`, the catch will be constrained to that corresponding with `Effort`. Otherwise, the actual fishing effort will be lower than `Effort`.

### Valid Configurations for Management Regulations:

There are several options for populating the slots in an `Advice` object:

### TAC:

Total allowable catch in units of `"Biomass"` (default) or `"Number"` (see `TACUnit`). `TACType` and `TACUnit` are each either length 1 (applied uniformly across all fleets) or a character vector of length `nFleet` specifying per-fleet values.

- single numeric value: global TAC distributed across the stocks and fleets in the `Data()` object according to `FleetAllocation(OM)`.
- numeric vector length `nFleet`: fleet-specific TAC.

**Effort:**

Either in absolute units corresponding with fleet-specific effort (`EffType = "Abs"`), or relative to the effort in the last historical year (`EffType = "Rel"`; default). `EffType` is either length 1 (applied uniformly across all fleets) or a character vector of length `nFleet` specifying per-fleet values.

Note that when `Effort` is supplied as a matrix (fleet x area), it is always treated as absolute regardless of `EffType`.

- single numeric value:
  - If `EffType="Rel"`: Effort relative to last historical year, applied to all fleets. E.g., if `Effort=0.5`, effort for all fleets will be set to half the effort in the last historical year.
  - If `EffType="Abs"`: the total effort, split across fleets according to `EffortAllocation()` (unless provided as a length `nFleet` vector, which is already fleet-specific). Note that in this case all fleets in the OM must have the same units for `Effort`).
- numeric vector length `nFleet`: Fleet-specific relative or absolute Effort.
- numeric matrix: Fleet- and Area-specific absolute Effort. Must have `nFleet` rows and `nArea` columns.

**Closure:**

Can only be `0` (closed) or `1` (open). Otherwise will be converted to `0` (`Closure`$<0.5$) or `1`

- vector length `nArea`
- matrix with `nFleet` rows and `nArea` columns: fleet-specific closures.

**Selectivity:**

Either a `Selectivity()` object, or a list length `nFleet` of `Selectivity` objects.

If a single `Selectivity` object, the prescribed selectivity schedule is applied to all fleets.

See section below for more details.

**Retention:**

Same as described for `Selectivity`, but for `Retention()` objects.

**Discard Mortality:**

Same as described for `Selectivity`, but for `DiscardMortality()` objects.

**Bag Limit:**

`BagLimit`, `LimitType`, and `ClosureMode` together define a bag-limit regulation for this stock's catch by each fleet: the number of fish an angler (or vessel) may retain per trip. `NA` in a fleet position of `BagLimit` means no limit applies for that fleet.

When `ClosureMode = "discard"`, catch exceeding the retention cap is converted to discards and discard mortality is applied via the fleet's `DiscardMortality()` object. When `ClosureMode = "stop"`, effort is reduced to prevent exceeding the bag limit and no additional discards are generated.

A bag limit pooling catch across several stocks for one fleet (rather than a single stock's own catch) is declared separately via `AggregateBagLimit()`, available to `mmp`-class management procedures. A stock's own `BagLimit` may still be set for any stock included in such a group, acting as a species-specific sub-cap within the pooled limit.

**Misc:**
A list of miscellaneous information that needs to be stored and accessed by the MP in future timesteps. Contents of `Advice@Misc` will be available in the `Data@Misc` slot in subsequent time steps.
`Advice@Misc` replaces `Data@Misc` as a whole, so an MP that keeps state across management cycles should start from `Data@Misc` and add to it. When an MP is called in a projection, the framework also sets `Data@Misc$MPName`, `Data@Misc$StockName`, `Data@Misc$AdviceYear` (the time step the advice is first applied in), and `Data@Misc$Interval` (the management interval in years); these are reset at every call.

**Populating Selectivity, Retention, and Discard Mortality Objects:**
The `Selectivity`, `Retention`, and `DiscardMortality` objects are used to set the selectivity-, retention-, and discard mortality- at-age or -at-size schedules.
If values are provided for these slots in the `Advice` object, the new regulations/specifications will apply to all future time steps until modified again by the `MP`.
Values can be set for these objects in the following ways:

- `MeanAtAge`:
  - A numeric vector length `nAge`, where `nAge` is the number of age classes for the stock (or stocks) that are being managed by the `MP`.
  - For area-specific schedules, a numeric matrix with `nAge` rows and `nArea` columns.
- `MeanAtLength`:
  - A numeric vector length `nClass`, where `nClass` is the number of length classes for the stock (or stocks) that are being managed by the `MP`. The `Classes` slot can be used to set the size classes corresponding with `MeanAtLength`. Otherwise, the classes will be taken from the corresponding `Length` object in the `OM`(if they exist, otherwise an error).
  - For area-specific schedules, a numeric matrix with `nClass` rows and `nArea` columns.
- `MeanAtWeight`:
  - A numeric vector length `nClass`, where `nClass` is the number of weight classes for the stock (or stocks) that are being managed by the `MP`. The `Classes` slot can be used to set the size classes corresponding with `MeanAtWeight`. Otherwise, the classes will be taken from the corresponding `Weight` object in the `OM` (if they exist, otherwise an error).
  - For area-specific schedules, a numeric matrix with `nClass` rows and `nArea` columns.
- `Pars` and `Model`: for `Selectivity` and `Retention` only.

These slots can be used to set the age- or size-schedules using parameters and a model. See `SelectivityModels()` and `RetentionModels()` for a details of built-in models.
`Pars` should be a named list of parameters, where each parameter can be either a single numeric value, or a numeric vector length `nArea` for area-specific schedules.

## Value

- `Advice()` returns an advice object.
- `TAC()`, `TACType()`, `TACUnit()`, `Effort()`, `EffType()`, `Closure()`, `Selectivity()`, `Retention()`, `DiscardMortality()`, `BagLimit()`, `LimitType()`, `ClosureMode()`, `Misc()` return the corresponding slot from an advice object `x`.
- Their replacement forms (e.g. `TAC<-`) return `x` with the corresponding slot updated.

### See Also

- advice for the class definition and slot-level documentation.
- `Selectivity()`, `Retention()`, `DiscardMortality()` for gear regulation sub-objects.
- `TripsScalar()`, `AnglerPerTrip()` for the effort-to-trips and angler-scaling parameters consumed by the bag limit model.
- `Theta()` for the within-trip catch overdispersion parameter.

### Examples

```
Advice(TAC = 1000)
```

---

## `advice-class` *The* `advice` *S4 Class*

---

### Description

The `advice` class defines management advice produced by a management procedure. Advice may include output controls (e.g. total allowable catch), input controls (e.g. effort), spatial controls (closures), gear effects, direct fishing mortality rates, or bag-limit regulations. See `Advice()` for details on valid entries and options for each slot.

### Slots

`TAC` Numeric vector length `1` or numeric length `nFleet` specifying total allowable catch in units of `TACUnit`.

`TACType` Character. Does the TAC refer to `"Removals"` (default) or `"Landings"`. Either length `1` (applied to all fleets) or a character vector of length `nFleet`.

`TACUnit` Character. Units of the TAC: `"Biomass"` (default) or `"Number"`. Either length 1 (applied to all fleets) or a character vector of length `nFleet`.

`Effort` Numeric vector length `1` or numeric length `nFleet` specifying relative or absolute fishing effort (in units of `Fleet@Effort@Effort`). Can also be a matrix with dimensions `nFleet` x `nArea` to set area-specific effort limits.

`EffType` Character. Are effort regulations relative to last historical year (`"Rel"`) or absolute (`"Abs"`; in units of `Effort()`). Either length 1 (applied to all fleets) or a character vector of length `nFleet`. Default is `"Rel"`.

`Closure` Numeric vector length `nArea` or array with dimensions `nFleet` x `nArea` specifyin an area open (`1`; default) or closed (`0`) to fishing.

`Selectivity` A `Selectivity()` object or an `nFleet` long list of `Selectivity()` objects defining gear selectivity prescribed by the `MP`

`Retention` A `Retention()` object or an `nFleet` long list of `Retention()` objects defining retention prescribed by the `MP`

`DiscardMortality` A `DiscardMortality()` object or an `nFleet` long list of `DiscardMortality()` objects defining discard mortality set in the `MP`

`BagLimit` Numeric vector or `NULL`. Bag limit in fish per angler per trip (when `LimitType = "angler"`) or fish per vessel per trip (when `LimitType = "boat"`), for this stock's catch by each fleet. Either length 1 (applied to all fleets) or a numeric vector of length `nFleet` for fleet-specific limits. `NULL` (default) means no bag limit regulation is active. `NA` for a given fleet position means no limit applies to that fleet. An aggregate bag limit pooling several stocks under one fleet is set separately via `AggregateBagLimit()` (`mmp` management procedures only); a stock's own `BagLimit` then acts as an optional species-specific sub-cap within that pooled limit. See `Advice()`.

`LimitType` Character or `NULL`. Specifies whether `BagLimit` is per-angler (`"angler"`; default) or per-vessel (`"boat"`) regulations. Either length 1 (applied to all fleets) or a character vector of length `nFleet`. See `Advice()`.

`ClosureMode` Character or `NULL`. Determines how the OM handles catch that exceeds the bag limit: `"discard"` (default) converts excess catch to discards, with discard mortality applied via the fleet's `DiscardMortality()` object; `"stop"` reduces effort to prevent the limit from being exceeded. Either length 1 (applied to all fleets) or a character vector of length `nFleet`. See `Advice()`.

`Misc` Miscellaneous list. Will be passed to `Data@Misc` in following time steps.

`Log` `list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

### See Also

`Advice()`, `Selectivity()`, `Retention()`, `DiscardMortality()`

---

## advicedata-class *The* `advicedata` *S4 Class*

---

### Description

Stores management advice outputs generated during a model run or provided as input. Used in the `Advice` slot of a data object and accessed by `LastTAC()` when retrieving the most recent catch limit.

### Usage

```
AdviceData()
```

### Value

`AdviceData()` returns a `advicedata` object.

### Slots

`TAC` A numeric vector, array, or list of Total Allowable Catch (TAC) values. When a vector, elements correspond to successive advice years.

`Effort` A numeric vector, array, or list of advised fishing effort values. Structured analogously to `TAC`.

`Misc` A named list for any additional advice-level metadata (e.g., harvest control rule outputs, reference point comparisons).

`AdviceData()` creates a new `advicedata` object.

### See Also

data, `Data()`, `LastTAC()`

---

## `Ages` *Ages Constructor and Accessors*

---

### Description

Construct an ages object defining the discrete age structure of a stock, or access and replace the `Ages` slot of a stock object and its individual slots.

### Usage

```
Ages(MaxAge, MinAge = 0, Units = "year", PlusGroup = TRUE)

Ages(x) <- value

MaxAge(x)

MaxAge(x) <- value

MinAge(x)

MinAge(x) <- value

PlusGroup(x)

PlusGroup(x) <- value
```

### Arguments

| | |
|---|---|
| `MaxAge` | `numeric(1)`. Maximum age in units of `Units`. When `PlusGroup = TRUE` (the default), fish at or older than this age are pooled into an open-ended plus group. Required for a non-empty object; if omitted the constructor returns an uninitialised ages suitable for placeholder use inside a stock. Alternatively, a stock object, in which case `Ages()` returns `x@Ages` directly — see *Pass-Through Access*. |

| | |
|---|---|
| `MinAge` | `numeric(1)`. Minimum (youngest) age class in units of `Units`. Must be non-negative and less than or equal to `MaxAge`. Default `0`. |
| `Units` | `character(1)`. Time unit for `MinAge` and `MaxAge`. Must be one of the strings returned by `ValidUnits()` (e.g., `"year"`, `"month"`). Default `"year"`. |
| `PlusGroup` | `logical(1)`. If `TRUE`, `MaxAge` is an open-ended plus group that accumulates all fish at or beyond that age. If `FALSE`, the age classes terminate at exactly `MaxAge` with no pooling. Default `TRUE`. |
| `x` | An ages object for slot accessors (`MaxAge()`, `MinAge()`, `PlusGroup()`), or a stock object for `Ages<-`. |
| `value` | For `Ages<-`: an ages object. For `MaxAge<-`, `MinAge<-`, `PlusGroup<-`: the replacement value for the corresponding slot (see *Slot Accessors*). |

## Details

### Age Class Calculation:

The `Classes` slot — a numeric vector of age classes expressed in years — is derived automatically from `MinAge`, `MaxAge`, and `Units` via `CalcAgeClasses()` and cannot be set directly. For seasonal models (`Units != "year"`), fractional year values are produced (e.g., `0`, `0.25`, `0.5`, … for quarterly seasons with `MinAge = 0`). Retrieve the computed vector with `Classes(x)`.

### Plus Group:

When `PlusGroup = TRUE`, the final age class absorbs all individuals at or beyond `MaxAge`. Mortality, growth, and selectivity at the plus-group age represent the average for that pooled cohort. Setting `PlusGroup = FALSE` treats `MaxAge` as an exact terminal age with no accumulation.

### Uninitialised Objects:

Calling `Ages()` without `MaxAge` returns an object with empty `MaxAge` and `MinAge` slots (`numeric(0)`). Validity checks are skipped for uninitialised objects, so they can be stored as placeholders inside a stock before parameters are specified. The model will not run until `MaxAge` is supplied.

### Pass-Through Access from a Stock:

When `MaxAge` is a stock object, `Ages()` acts as an accessor:

```
Ages(my_stock)          # returns my_stock@Ages
Ages(my_stock) <- a     # replaces my_stock@Ages with ages-class object `a`
```

### Slot Accessors:

Individual slots can be read or replaced using functions that match the slot names. All replacement functions re-validate the object after assignment:

```
MaxAge(a)       <- 20
MinAge(a)       <- 1
PlusGroup(a)    <- FALSE
```

### Value

`Ages()` returns an ages object. If `MaxAge` is a stock, returns `x@Ages`.
`Ages<-` returns the stock `x` with the `Ages` slot replaced by `value` and the object re-validated.

`MaxAge()`, `MinAge()`, `PlusGroup()` return the value of the corresponding slot from `x`.
`MaxAge<-`, `MinAge<-`, `PlusGroup<-` return `x` with the named slot updated and the object re-validated.

### See Also

ages for the class definition and slot-level documentation. `Stock()` for the enclosing stock constructor. `ValidUnits()` for accepted unit strings. `Classes()` to retrieve the derived age-class vector.

Other ages: `ages-class`

### Examples

```
# Basic construction
a <- Ages(MaxAge = 20)
a

# Seasonal model: quarterly age classes
a_qtr <- Ages(MaxAge = 20, Units = "quarter")
Classes(a_qtr)

# No plus group
a_exact <- Ages(MaxAge = 20, PlusGroup = FALSE)

# Slot accessors
MaxAge(a)
MaxAge(a) <- 25
PlusGroup(a) <- FALSE

# Pass-through access from a stock
s <- Stock(Name = "Cod", Ages = Ages(MaxAge = 15))
Ages(s)
Ages(s) <- Ages(MaxAge = 20)
```

---

## ages-class *The* ages *S4 Class*

---

### Description

Defines the discrete age structure of a stock object. Objects are typically created via `Ages()`, which documents all parameters and validates inputs.

### Details

An object is considered uninitialised when `MaxAge` or `MinAge` is `numeric(0)`.

Validity is enforced on non-empty objects: `MaxAge` and `MinAge` must be finite scalars, `MinAge` must be non-negative, `MaxAge >= MinAge`, and `Units` must be a scalar string accepted by `ValidUnits()`.

Direct construction via `methods::new()` is not recommended; use `Ages()` instead, which populates `Classes` automatically.

### Slots

`MaxAge numeric(1)`. Maximum age in units of `Units`. When `PlusGroup = TRUE`, fish older than this age are pooled into the plus group.

`MinAge numeric(1)`. Minimum (youngest) age class in units of `Units`. Must be non-negative and less than or equal to `MaxAge`.

`Units character(1)`. Time unit for `MinAge` and `MaxAge`. Must be one of the values returned by `ValidUnits()` (e.g., `"year"`).

`PlusGroup logical(1)`. If `TRUE`, `MaxAge` is treated as an open-ended plus group that accumulates all fish at or beyond that age.

`Classes numeric`. Vector of age classes in years, derived automatically from `MinAge`, `MaxAge`, and `Units` by `CalcAgeClasses()`. Not intended to be set directly; use `Ages()` to trigger recalculation.

### See Also

- `Ages()` for the constructor and slot-accessor functions.
- stock for the enclosing object.
- `ValidUnits()` for accepted unit strings.
- `Classes()` to retrieve the computed age-class vector.

Other ages: `Ages()`

---

## `AgeStructuredObs` *Age-Structured Observation Model*

---

### Description

An obs object representing a data-moderate scenario suitable for age-structured stock assessment. Landings, dicards, a fishery-independent survey, and annual age-composition samples from both landed and discarded fish are simulated.

### Usage

```
AgeStructuredObs
```

### Format

An obs object. Populated slots:

- `Landings` (catchobs): precisely reported landings with CV in `[0.05, 0.10]` and near-unbiased reporting `[0.95, 1.05]`.
- `Discards` (catchobs): less precisely reported discards with CV in `[0.15, 0.25]` and near-unbiased reporting `[0.95, 1.05]`.
- `Survey` (indicesobs): annual survey targeting spawning biomass with CV in `[0.10, 0.20]` and negligible autocorrelation `[0.0, 0.1]`.
- `LandingsAtAge` (compobs): age-composition samples of landed fish from a port-sampling programme. Nominal sample size `[200, 400]` fish per year; ESS `[50, 100]` reflecting within-trip clustering; Dirichlet-Multinomial dispersion `Theta` in `[0.5, 1.0]`.
- `DiscardsAtAge` (compobs): age-composition samples of discarded fish from at-sea observers. Nominal sample size `[100, 200]`; ESS `[30, 80]`; `Theta` in `[0.4, 0.8]`, reflecting higher uncertainty relative to port sampling.

Empty slots (no data simulated): `Effort`, `CPUE`, `LandingsAtSize`, `DiscardsAtSize`.

### See Also

obs, `Obs()`, `CatchObs()`, `IndicesObs()`, `CompObs()`, CatchAndSurveyObs, CommercialFleetObs, LengthStructuredObs, DataRichObs

Other obs-examples: `CatchAndSurveyObs`, `CommercialFleetObs`, `DataRichObs`, `LengthStructuredObs`

### Examples

```
AgeStructuredObs
LandingsAtAge(AgeStructuredObs)
DiscardsAtAge(AgeStructuredObs)
```

---

## aggbaglimit-class *The* aggbaglimit *S4 Class*

---

### Description

The `aggbaglimit` class declares an aggregate bag limit pooling catch across several independently-modelled stocks for a single fleet - management advice that does not belong to any one stock's advice object. Returned by `mmp`-class management procedures alongside a per-stock `Advice` list. See `AggregateBagLimit()` for the constructor and full parameter documentation.

### Slots

`Fleet` Character or numeric, length 1. Identifies the fleet the aggregate limit applies to, by name or position.

`Stocks` Character or numeric, length 2 or more. Identifies the stocks pooled under this limit, by name or position.

`BagLimit` Numeric, length 1. The aggregate limit, in fish per trip.

`LimitType` Character. `"boat"` or `"angler"` - see `Advice()` for the shared definition. Governs enforcement for the aggregate limit and for any per-stock `BagLimit` set (as a species-specific sub-cap) on a stock included in `Stocks`.

`ClosureMode` Character. `"discard"` or `"stop"` - see `Advice()` for the shared definition. Governs enforcement for the aggregate limit and for any per-stock `BagLimit` sub-cap on a stock included in `Stocks`.

`Misc` Miscellaneous list.

### See Also

`AggregateBagLimit()`, advice

---

## `AggregateBagLimit` *Aggregate Bag Limit Constructor*

---

### Description

Construct an aggbaglimit object declaring an aggregate bag limit that pools catch across several independently-modelled stocks for a single fleet - management advice that does not belong to any one stock's `Advice()` object. Used by `mmp`-class management procedures, which return a list of these objects alongside the ordinary per-stock `Advice` list.

### Usage

```
AggregateBagLimit(
  Fleet,
  Stocks,
  BagLimit,
  LimitType = "boat",
  ClosureMode = "discard",
  Misc = list()
)
```

### Arguments

| | |
|---|---|
| `Fleet` | Character or numeric, length 1. The fleet the aggregate limit applies to, by name or position. |

| | |
|---|---|
| `Stocks` | Character or numeric, length 2 or more. The stocks pooled under this limit, by name or position. A limit on a single stock does not need `AggregateBagLimit()` - set `BagLimit` directly on that stock's `Advice` object instead (see `Advice()`). |
| `BagLimit` | Numeric, length 1. The aggregate limit, in fish per trip (per angler if `LimitType = "angler"`, per vessel if `"boat"`). |
| `LimitType` | Character. `"boat"` (default) or `"angler"` - see `Advice()` for the shared definition. Also governs enforcement of any per-stock `BagLimit` set as a species-specific sub-cap on a stock included in `Stocks` (that stock's own `LimitType` is ignored). |
| `ClosureMode` | Character. `"discard"` (default) or `"stop"` - see `Advice()` for the shared definition. Also governs enforcement of any per-stock `BagLimit` sub-cap on a stock included in `Stocks` (that stock's own `ClosureMode` is ignored). |
| `Misc` | Miscellaneous list. Default `list()`. |

## Details

`AggregateBagLimit()` declares the one piece of a pooled bag-limit regulation that cannot be expressed on any single stock's `Advice` object: which stocks are pooled, and the ceiling that applies to their combined catch. Everything else reuses the ordinary `Advice` mechanism:

- A species-specific sub-cap for one stock within the group is just an ordinary per-stock `BagLimit`, set on that stock's own `Advice` object (see `Advice()`). No separate slot or argument is needed for this.
- `Theta`, `TripsScalar`, and `AnglerPerTrip` are read directly from the fleet's `Effort` object for the *first* stock listed in `Stocks` (`Fleet(OM)[[Stocks[1]]][[Fleet]]@Effort`) - not set here. This assumes the fleet's trip behaviour is consistent across the stocks it is pooled over, since these describe fleet activity rather than any one species.

`mmp`-class management procedures return a list of `AggregateBagLimit()` objects (one per pooled regulation) alongside the ordinary per-stock `Advice` list.

## Value

An aggbaglimit object.

## See Also

aggbaglimit for the class definition. `Advice()` for the per-stock advice object aggregate bag limits complement.

`Albacore` *Stock class objects*

## Description

Example objects of class Stock

## Usage

```
Albacore

Blue_shark

Bluefin_tuna

Bluefin_tuna_WAtl

Butterfish

Herring

Mackerel

Porgy

Rockfish

Snapper

Sole

Toothfish
```

## Format

An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.
An object of class `Stock` of length 1.

An object of class `Stock` of length 1.

An object of class `Stock` of length 1.

An object of class `Stock` of length 1.

### Examples

```
avail("Stock")
```

---

## `Albacore_TwoFleet` *MOM class objects*

---

### Description

Example objects of class MOM

### Usage

```
Albacore_TwoFleet
```

### Format

An object of class `MOM` of length 1.

### Examples

```
avail("MOM")
```

---

## `AlbacoreExStock` *Albacore Example Stock*

---

### Description

An example stock object representing an Albacore tuna (*Thunnus alalunga*) stock. This is a long-lived, slow-growing, large pelagic species with low natural mortality. See also ButterfishExStock for a contrasting short-lived, fast-growing example.

### Usage

```
AlbacoreExStock
```

### Format

A stock object with the following slots populated:

- `Name`: `"Example Albacore Stock"`.
- `CommonName`: `"Albacore"`.
- `Species`: `"Thunnus alalunga"`.
- `Ages`: An `Ages()` object with `MaxAge = 20` years.
- `Length`: A `Length()` object with von Bertalanffy parameters — `Linf` (121–135 cm), `K` (0.16–0.22 yr^-1^), `t0` (-1.86 to -1.41 yr) — and `CVatAge` (0.1–0.15).
- `Weight`: A `Weight()` object with allometric parameters `alpha = 1.34e-05` and `beta = 3.106`.
- `NaturalMortality`: A `NaturalMortality()` object with `M` = 0.35–0.45 yr^-1^.
- `Maturity`: A `Maturity()` object with logistic maturity-at-length parameters `L50` (81–91 cm) and `L50_95` (10–12 cm).
- `Fecundity`: An empty `Fecundity()` object; fecundity-at-age is computed internally during population (see Details).
- `SRR`: A `SRR()` object with Beverton-Holt steepness `h` (0.65–0.85), unfished recruitment `R0 = 1000`, recruitment variability `SD` (0.15–0.3), and autocorrelation `AC` (0.1–0.9).
- `Spatial`: A `Spatial()` object with `UnfishedDist` (0.095–0.105), `ProbStaying` (0.8–0.9), and `RelativeSize` (0.095–0.105).
- `Depletion`: A `Depletion()` object with current biomass sampled from 5–60% of `B0`.
- `nYear`, `pYear`, `nSim`, `CurrentYear`, `Years`, `Seasons`: Set by `PopulateStock()`.
- `Misc`: Empty list for user-defined information.
- `Log`: Internal list for tracking object state.

Two-element numeric vectors in the slot descriptions above are uniform bounds. `PopulateStock()` samples from these bounds across simulations and expands the results into `[nSim x nAge x nYear]` arrays. The sections below describe what each component-level `Populate*()` function produces.

**Ages:**
`CalcAgeClasses()` generates the integer age vector `Ages@Classes` (`0:20`), which is used as the age dimension in all subsequent arrays.

**Length:**
`PopulateLength()` samples `Linf`, `K`, and `t0` independently for each simulation from uniform distributions defined by the bounds, then computes mean length-at-age using the von Bertalanffy growth equation. `CVatAge` is expanded across simulations and, when `ALK = TRUE`, an age–length key is constructed.
See also `LengthModels()`.

**Weight:**
`PopulateWeight()` applies the allometric relationship $W =$ `alpha` $\times L^{\wedge}$`beta`$^{\wedge}$ to the mean length-at-age array to produce mean weight-at-age. An age–weight key is optionally constructed when `AWK = TRUE`.
See also `WeightModels()`.

**Natural Mortality:**

`PopulateNaturalMortality()` samples `M` for each simulation from the uniform bounds and expands the result into a natural mortality-at-age array. Stochastic annual deviates are then added.

See also `NaturalMortalityModels()`.

**Maturity:**

`PopulateMaturity()` applies the logistic maturity-at-length curve, parameterised by `L50` and `L50_95`, to the populated length-at-age array to produce a maturity-at-age array.

See also `MaturityModels()`.

**Fecundity:**

Because the `Fecundity` slot is empty, `PopulateFecundity()` computes fecundity-at-age as the element-wise product of weight-at-age and maturity-at-age, representing spawning output per individual.

See also `FecundityModels()`.

**Stock–Recruitment:**

`PopulateSRR()` samples `h` for each simulation from the uniform prior and generates a time series of log-normal recruitment deviates with standard deviation `SD` and autocorrelation `AC`.

**Spatial:**

`PopulateSpatial()` samples `UnfishedDist`, `ProbStaying`, and `RelativeSize` for each simulation and uses them to construct a full inter-area movement matrix, then expands all spatial arrays across simulations and years. The relatively high `ProbStaying` (0.8–0.9) reflects low connectivity between the two areas.

See also `SRRModels()`.

**Depletion:**

`PopulateDepletion()` samples current biomass depletion relative to `B0` for each simulation from the uniform prior, setting the initial conditions of the operating model.

## See Also

ButterfishExStock, `SeasonalSpatialExStock()`, `Stock()`, `PopulateStock()`, `Populate()`, stock

## Examples

```
AlbacoreExStock

## Not run:
PopulatedStock <- PopulateStock(
  AlbacoreExStock,
  nYear = 50,
  pYear = 30,
  nSim = 48,
  seed = 42
```

```
)

## End(Not run)
```

---

`AnnualData` *Aggregate Seasonal Data to Annual Time Steps*

---

## Description

Converts a seasonal data object (`Seasons > 1`, one row per time step) to one row per calendar year. Data with `Seasons = 1` is returned unchanged.

## Usage

```
AnnualData(Data, IndexSeasons = NULL, CompleteYears = TRUE)
```

## Arguments

| | |
|---|---|
| `Data` | A data object. |
| `IndexSeasons` | Seasons (integers in `1:Data@Seasons`) averaged to form each annual index value. `NULL` (default) uses every season. Either an integer vector applied to every index, or a list with elements `Survey` and/or `CPUE`, each either an integer vector applied to every index in that slot or a list with one element (integer vector, or `NULL` for every season) per index. Ignored when `Seasons = 1`. |
| `CompleteYears` | Logical. If `TRUE` (default), calendar years with fewer than `Seasons` time steps in `Data@Years` (e.g. a partially observed final year) are dropped, so a partial year's catch is not read as a low annual catch. |

## Details

- `Landings`, `Discards`, `Effort`, and the composition slots (`LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, `DiscardsAtSize`) are summed over the seasons of each calendar year. A cell with no non-`NA` value in a year stays `NA`. The `CV` of summed catch and effort assumes independent errors among seasons.
- `CPUE` and `Survey` values and `CV`s are averaged (simple mean, ignoring `NA` and negative values) over the seasons in `IndexSeasons`. Index `Ref` values are not modified, and so are assumed to be in the units of the aggregated annual index.

The `Advice`, `LifeHistory`, `Exploitation`, and `Reference` slots are not modified.

## Value

A data object with `Seasons = 1` and whole-number `Years`. Every aggregated slot has one row per element of the returned `Years`.

## See Also

DataHelpers, IndexMPs

`ApplyAC` *Apply AR(1) autocorrelation to residuals*

## Description

Applies AR(1) propagation to a numeric matrix of multiplicative (normal-space) residuals (`sim x year`), respecting missing values (NAs).

## Usage

```
ApplyAC(Resid, AC, LastError)
```

## Arguments

| | |
|---|---|
| `Resid` | Numeric matrix of multiplicative (normal-space) residuals, dimensions nSim x nYear. |
| `AC` | Numeric vector of length nSim, autocorrelation coefficient per simulation (must be in between -1 & 1). |
| `LastError` | Numeric vector of length nSim giving the starting value for the AR(1) recursion, in normal (natural) space — i.e. the last observed multiplicative residual. |

## Details

Autocorrelation is applied only across non-NA values within each simulation. The recursion itself happens in log-space (the only space in which it is valid), but both the input and the returned residuals are in normal space.

## Value

Numeric matrix of same dimensions as `Resid` with autocorrelated multiplicative (normal-space) residuals.

## See Also

`GenResiduals()`, `CalcResidualStats()`

---

`applyMMP` *Apply multi Management Procedures (class MMP) to a hierarchical list of Data class objects*

---

## Description

Apply multi Management Procedures (class MMP) to a hierarchical list of Data class objects

## Usage

```
applyMMP(
  DataList,
  MP = NA,
  reps = 1,
  nsims = NA,
  silent = FALSE,
  parallel = .sfIsRunning()
)
```

## Arguments

| | |
|---|---|
| `DataList` | A hierarchical list of `Data` objects (Fleets nested in Stocks) |
| `MP` | Name of the MMP to run |
| `reps` | Number of samples |
| `nsims` | Optional. Number of simulations. |
| `silent` | Logical. Should messages be suppressed? |
| `parallel` | Logical. Whether to run MPs in parallel |

## Value

A hierarchical list of management recommendations (object class Rec), Fleets nested in Stocks

---

`applyMP` *Apply Management Procedures to an object of class Data*

---

## Description

Apply Management Procedures to an object of class Data

### Usage

```
applyMP(
  Data,
  MPs = NA,
  reps = 100,
  nsims = NA,
  silent = FALSE,
  parallel = .sfIsRunning()
)
```

### Arguments

| | |
|---|---|
| Data | An object of class Data |
| MPs | Name(s) of the MPs to run |
| reps | Number of samples |
| nsims | Optional. Number of simulations. |
| silent | Logical. Should messages be suppressed? |
| parallel | Logical. Whether to run MPs in parallel. Can be a vector of length(MPs) |

### Value

A list with the first element a list of management recommendations, and the second the updated Data object

---

## array_df_conversion *Convert Between Arrays and Data Frames*

---

### Description

`Array2DF()` converts a matrix or array to a tidy data frame, with dimension names mapped to columns (`Sim`, `Stock`, `Age`, `Year`, `Fleet`, `Area`) and values in a `Value` column. Numeric columns are coerced with `as.numeric()`; `Stock` and `Fleet` are returned as ordered factors.

### Usage

```
Array2DF(array)

DF2Array(DF)
```

### Arguments

| | |
|---|---|
| array | A `matrix` or `array` object. |
| DF | A `data.frame` containing a `Value` column and at least one dimension column (`Sim`, `Stock`, `Age`, `Year`, `Fleet`, `Area`). |

### Details

`DF2Array()` is the inverse: it reshapes a tidy data frame back into a named array, using any columns whose names match `Sim`, `Stock`, `Age`, `Year`, `Fleet`, or `Area` as array dimensions.

### Value

- `Array2DF()` returns a `data.frame` with one row per array element and one column per dimension plus a `Value` column.
- `DF2Array()` returns a named `array` whose dimensions and dimnames are derived from the unique values of the dimension columns in `DF`.

### Examples

```
# Round-trip: array -> data frame -> array
a <- array(1:24, dim = c(2, 3, 4),
           dimnames = list(Sim = 1:2, Year = 1:3, Age = 1:4))
df <- Array2DF(a)
df
DF2Array(df)
```

---

## `ArrayOperations` *Perform Operations on Two Arrays*

---

### Description

Multiply, divide, subtract, or add two arrays together.

### Usage

```
ArraySum(array1, array2)

ArrayDivide(array1, array2)

ArrayMultiply(array1, array2)

ArraySubtract(array1, array2)

ArrayExtend(array1, array2)

ArrayFill(object) <- value
```

### Arguments

| | |
|---|---|
| `array1` | An array with named dimensions. |
| `array2` | An array with the same named dimensions as `array1`. |
| `object` | An array with named dimensions |
| `value` | An array with the same dimension names as `object` |

### Details

The arrays must have the same named dimensions, but do not need to have the same length for each dimension.

- `ArraySum`: Sum `array1` and `array2`
- `ArrayDivide`: Divide `array1` by `array2`
- `ArrayMultiply`: Multiply `array1` and `array2`
- `ArraySubtract`: Subtract `array2` from `array1`
- `ArrayExtend`: extends the arrays to have matching dimensions (see `Extend()`)
- `ArrayFill`: Update array `object` with the values in `value`

### Value

The resulting array

### Examples

```
# Array with Sim, Year
array1 <- array(1:20,
                dim=c(2,5),
                dimnames = list(
                  Sim=1:2,
                  Year=2021:2025
                ))

array2 <- array(1:20,
                dim=c(1,3),
                dimnames = list(
                  Sim=1,
                  Year=2024:2026
                ))

ArrayExtend(array1, array2)

ArraySum(array1, array2)
ArrayDivide(array1, array2)
ArrayMultiply(array1, array2)
ArraySubtract(array1, array2)



# Array with Age, Year
array1 <- array(1:20,
                dim=c(5,2),
                dimnames = list(
                  Age=1:5,
                  Year=2021:2022
                ))
```

```
array2 <- array(1:20,
                dim=c(1,3),
                dimnames = list(
                  Age=1,
                  Year=2024:2026
                ))

ArrayExtend(array1, array2)

ArraySum(array1, array2)
ArrayDivide(array1, array2)
ArrayMultiply(array1, array2)
ArraySubtract(array1, array2)


# ArrayFill

## Same dimensions
object <- array(NA,
               dim=c(3,5),
               dimnames=list(Sim=1:3,
                             Year=2021:2025)
)

value <- array(1:2,
               dim=c(1,2),
               dimnames=list(Sim=2,
                             Year=2022:2023)
)

ArrayFill(object) <- value
object


## Extend Dimensions
object <- array(NA,
                dim=c(3,5),
                dimnames=list(Sim=1:3,
                              Year=2021:2025)
)

value <- array(1:12,
               dim=c(3,4),
               dimnames=list(Sim=1:3,
                             Year=c(2019, 2025:2027))
)

ArrayFill(object) <- value
object
```

`ASAP2OM` *Convert ASAP 3 assessments into an operating model*

---

## Description

Reads a fitted ASAP model and uses the MLE estimates with identical reconstruction among simulations. Future recruitment is sampled from a lognormal distribution with autocorrelation. `ASAP2Data` imports a Data object.

## Usage

```
ASAP2OM(
  asap,
  nsim = 48,
  proyears = 50,
  mcmc = FALSE,
  spawn_time_frac = 0,
  Name = "ASAP Model",
  Source = "No source provided",
  nyr_par_mu = 3,
  Author = "No author provided",
  report = FALSE,
  silent = FALSE
)

ASAP2Data(asap, Name = "ASAP assessment")
```

## Arguments

| | |
|---|---|
| `asap` | A list returned by ASAP, e.g., `asap <- dget("asap3.rdat")`. |
| `nsim` | The number of simulations in the operating model |
| `proyears` | The number of MSE projection years |
| `mcmc` | Logical, whether to use mcmc samples. Currently unsupported. |
| `spawn_time_frac` | Numeric, the fraction of a year when spawning takes place (e.g., 0.5 is the midpoint of the year) |
| `Name` | The name of the operating model |
| `Source` | Reference to assessment documentation e.g. a url |
| `nyr_par_mu` | integer, the number of recent years to estimate vulnerability over for future projections |
| `Author` | Who did the assessment |
| `report` | Logical, should a comparison of biomass reconstruction be produced? |
| `silent` | Logical, should progress reporting be printed to the console? |

### Details

Length at age is not used in ASAP so arbitrary placeholder values are used for length-based parameters. Update these parameters to model length in the operating model.

### Value

An operating model `OM` class.

### Author(s)

Q. Huynh

### See Also

Assess2OM

---

`Assess2MOM` *Reads bootstrap estimates from a stock assessment model into a multi-fleet operating model.*

---

### Description

A function that develops a multiple fleet operating model (`MOM`) and either models a unisex or 2-sex stock from arrays of abundance, fishing mortality, and biological parameters. The user still needs to parameterize most of the observation and implementation portions of the operating model.

### Usage

```
Assess2MOM(
  Name = "MOM created by Assess2MOM",
  proyears = 50,
  interval = 2,
  CurrentYr = as.numeric(format(Sys.Date(), "%Y")),
  h = 0.999,
  Obs = MSEtool::Imprecise_Unbiased,
  Imp = MSEtool::Perfect_Imp,
  naa,
  faa,
  waa,
  Mataa,
  Maa,
  laa,
  fecaa,
  nyr_par_mu = 3,
  LowerTri = 1,
  recind = 0,
```

```
  plusgroup = TRUE,
  altinit = 0,
  fixq1 = TRUE,
  report = FALSE,
  silent = FALSE,
  ...
)
```

**Arguments**

| | |
|---|---|
| `Name` | Character string. The name of the multi-OM. |
| `proyears` | Positive integer. The number of projection years for MSE. |
| `interval` | Positive integer. The interval at which management procedures will update the management advice in multiMSE, e.g., 1 = annual updates. |
| `CurrentYr` | Positive integer. The current year (e.g., final year of fitting to data) |
| `h` | The steepness of the stock-recruitment curve. Either a single numeric or a length nsim vector. |
| `Obs` | Either a single observation model to be used for all sexes and populations (class `Obs`), or a list where `Obs[[f]]` is the `Obs` object for fleet `f` (identical between sexes). |
| `Imp` | Either a single implementation model to be used for all sexes and populations (class `Imp`), or a list where `Imp[[f]]` is the `Obs` object for fleet `f` (identical between sexes). |
| `naa` | Numbers-at-age by sex `[first age is age zero]`. Four-dimensional numeric array `[sim, ages, year, p]`. `[p]` indexes the population, where `[p = 1]` for females and `[p = 2]` for males. |
| `faa` | Fishing mortality rate-at-age by sex and fleet `[first age is age zero]`. Five-dimensional numeric array `[sim, ages, year, p, f]` where `[f]` indexes fishery fleet. |
| `waa` | Weight-at-age `[first age is age zero]`. Four-dimensional numeric array `[sim, ages, year, p]`. |
| `Mataa` | Maturity (spawning fraction)-at-age `[first age is age zero]`. Four-dimensional numeric array `[sim, ages, year, p]`. |
| `Maa` | Natural mortality rate-at-age `[first age is age zero]`. Four-dimensional numeric array `[sim, ages, year, p]`. |
| `laa` | Length-at-age `[first age is age zero]`. Four-dimensional numeric array `[sim, ages, year, p]`. |
| `fecaa` | Fecundity at age `[first age is age zero]`. If missing, default fecundity is the product of maturity and weight at age. |
| `nyr_par_mu` | Positive integer. The number of recent years that natural mortality, age vulnerability, weight, length and maturity parameters are averaged over for defining future projection conditions. |
| `LowerTri` | Integer. The number of recent years for which model estimates of recruitment are ignored (not reliably estimated by the assessment) |

`recind` Positive integer. The first age class that fish 'recruit to the fishery'. The default is 0 - ie the first position in the age dimension of naa is age zero

`plusgroup` Logical. Does the assessment assume that the oldest age class is a plus-group?

`altinit` Integer. Various assumptions for how to set up the initial numbers. 0: standard, 1: no plus group, 2: temporary fix for MSEtool plus group initialization

`fixq1` Logical. Should q be fixed (ie assume the F-at-age array faa is accurate?

`report` Logical, if TRUE, a diagnostic will be reported showing the matching of the OM reconstructed numbers at age vs the assessment.

`silent` Whether to silence messages to the console.

`...` Additional arguments (for all, either a numeric or a length nsim vector):

- `SRrel` Stock-recruit relationship. (`1` for Beverton-Holt (default), `2` for Ricker)
- `R0` unfished recruitment
- `phi0` unfished spawners per recruit associated with R0 and h. With time-varying parameters, openMSE uses the mean phi0 in the first `ageM` (age of 50 percent maturity) years for the stock-recruit relationship. `Assess2OM` will re-calculate R0 and h in the operating model such that the stock-recruit `alpha` and `beta` parameters match values implied in the input.
- `Perr` recruitment standard deviation (lognormal distribution) for sampling future recruitment
- `AC` autocorrelation in future recruitment deviates.

## Details

Use a seed for the random number generator to sample future recruitment.

## Value

An object of class `MOM`

## Author(s)

Q. Huynh

## See Also

SS2MOM multiMSE Assess2OM

| | |
|---|---|
| `Assess2OM` | *Reads bootstrap estimates from a stock assessment model (including VPA) into an operating model.* `Assess2OM` *is identical to* `VPA2OM`*.* |

## Description

A function that uses a set of bootstrap estimates of numbers-at-age, fishing mortality rate-at-age, M-at-age, weight-at-age, length-at-age and Maturity-at-age to define a fully described MSEtool operating model. The user still needs to parameterize most of the observation and implementation portions of the operating model.

## Usage

```
Assess2OM(
  Name = "A fishery made by VPA2OM",
  proyears = 50,
  interval = 2,
  CurrentYr = as.numeric(format(Sys.Date(), "%Y")),
  h = 0.999,
  Obs = MSEtool::Imprecise_Unbiased,
  Imp = MSEtool::Perfect_Imp,
  naa,
  faa,
  waa,
  Mataa,
  Maa,
  laa,
  nyr_par_mu = 3,
  LowerTri = 1,
  recind = 0,
  plusgroup = TRUE,
  altinit = 0,
  fixq1 = TRUE,
  report = FALSE,
  silent = FALSE,
  ...
)

VPA2OM(
  Name = "A fishery made by VPA2OM",
  proyears = 50,
  interval = 2,
  CurrentYr = as.numeric(format(Sys.Date(), "%Y")),
  h = 0.999,
  Obs = MSEtool::Imprecise_Unbiased,
  Imp = MSEtool::Perfect_Imp,
```

```
  naa,
  faa,
  waa,
  Mataa,
  Maa,
  laa,
  nyr_par_mu = 3,
  LowerTri = 1,
  recind = 0,
  plusgroup = TRUE,
  altinit = 0,
  fixq1 = TRUE,
  report = FALSE,
  silent = FALSE,
  ...
)
```

**Arguments**

| | |
|---|---|
| `Name` | Character string. The name of the operating model. |
| `proyears` | Positive integer. The number of projection years for MSE. |
| `interval` | Positive integer. The interval at which management procedures will update the management advice in runMSE, e.g., 1 = annual updates. |
| `CurrentYr` | Positive integer. The current year (final year of fitting to data) |
| `h` | The steepness of the stock-recruitment curve (greater than 0.2 and less than 1, assumed to be close to 1 to match VPA assumption). Either a single numeric or a length nsim vector. |
| `Obs` | The observation model (class Obs). This function only updates the catch and index observation error. |
| `Imp` | The implementation model (class Imp). This function does not update implementation parameters. |
| `naa` | Numeric array `[sim, ages, year]`. Numbers-at-age `[first age is age zero]`. |
| `faa` | Numeric array `[sim, ages, year]`. Fishing mortality rate-at-age `[first age is age zero]`. |
| `waa` | Numeric array `[sim, ages, year]`. Weight-at-age `[first age is age zero]`. |
| `Mataa` | Numeric array `[sim, ages, year]`. Maturity (spawning fraction)-at-age `[first age is age zero]`. |
| `Maa` | Numeric array `[sim, ages, year]`. Natural mortality rate-at-age `[first age is age zero]`. |
| `laa` | Numeric array `[sim, ages, year]`. Length-at-age `[first age is age zero]`. |
| `nyr_par_mu` | Positive integer. The number of recent years that natural mortality, age vulnerability, weight, length and maturity parameters are averaged over for defining future projection conditions. |
| `LowerTri` | Integer. The number of recent years for which model estimates of recruitment are ignored (not reliably estimated by the assessment) |
| `recind` | Positive integer. The first age class that fish 'recruit to the fishery'. The default is 0 - ie the first position in the age dimension of naa is age zero |

`plusgroup` Logical. Does the assessment assume that the oldest age class is a plus-group?

`altinit` Integer. Various assumptions for how to set up the initial numbers. 0: standard, 1: no plus group, 2: temporary fix for MSEtool plus group initialization

`fixq1` Logical. Should q be fixed (ie assume the F-at-age array faa is accurate?

`report` Logical, if TRUE, a diagnostic will be reported showing the matching of the OM reconstructed numbers at age vs the assessment.

`silent` Whether to silence messages to the console.

`...` Additional arguments (for all, either a numeric or a length nsim vector):

- `fecaa` Fecundity at age. Default fecundity is the product of maturity and weight at age.
- `SRrel` Stock-recruit relationship. (`1` for Beverton-Holt (default), `2` for Ricker)
- `R0` unfished recruitment
- `phi0` unfished spawners per recruit associated with R0 and h. With time-varying parameters, openMSE uses the mean phi0 in the first `ageM` (age of 50 percent maturity) years for the stock-recruit relationship. `Assess2OM` will re-calculate R0 and h in the operating model such that the stock-recruit `alpha` and `beta` parameters match values implied in the input.
- `Perr` recruitment standard deviation (lognormal distribution) for sampling future recruitment
- `AC` autocorrelation in future recruitment deviates.
- `spawn_time_frac` The fraction of a year when spawning takes place (e.g., 0.5 is the midpoint of the year)

## Details

Use a seed for the random number generator to sample future recruitment.

## Value

An object of class `OM`.

## Author(s)

T. Carruthers

## See Also

SS2OM iSCAM2OM WHAM2OM ASAP2OM

`AsympExFleet` *Asymptotic Selectivity Example Fleet*

## Description

An example fleet object demonstrating a fleet with effort that stabilises early in the historical period and asymptotic length-based selectivity. Intended for use with AlbacoreExStock or ButterfishExStock to illustrate fleet parameterization in `MSEtool`. See also DomeExFleet for a contrasting dome-shaped selectivity example.

## Usage

```
AsympExFleet
```

## Format

A fleet object with the following slots populated:

- `Name`: `"AsympExFleet"`. Unique fleet identifier.
- `Effort`: An `Effort()` object with a piecewise-linear historical effort trajectory defined at four relative time points (0, 0.3, 0.6, 1.0), with lower and upper bounds of (0, 0.4, 1, 1) and (0, 0.6, 1, 1) respectively, and `CV = 0.1` controlling stochastic variation around the trend.
- `Selectivity`: A `Selectivity()` object with asymptotic length-based selectivity parameters — `L5` (0.4–0.5), `LFS` (0.7–0.8), and `Vmaxlen = 1` (fixed) — with `isRel = TRUE` indicating that `L5` and `LFS` are expressed relative to maturity `L50`.
- `Catchability`: An empty `Catchability()` object; efficiency defaults to 1 across all years during population (see Details).
- `Retention`: An empty `Retention()` object; full retention is assumed during population (see Details).
- `DiscardMortality`: An empty `DiscardMortality()` object.
- `Closure`: `NULL`. No spatial or temporal closures defined.
- `WeightFleetRetained`, `WeightFleetSelected`: `NA`. Computed from selectivity, retention, and the stock's weight-at-length during population (see Details).
- `Bioeconomic`: An empty `Bioeconomic()` object.
- `nYear`, `pYear`, `nSim`, `CurrentYear`, `Years`, `Seasons`: Inherited from the paired `Stock()` object by `PopulateFleet()`.
- `Misc`: Empty list for user-defined information.

## Details

`PopulateFleet()` expands the input parameters into `[nSim x nAge x nYear]` arrays using the age, length, and spatial structure of the paired populated stock. The sections below describe what each component-level `Populate*()` function produces.

**Effort:**

`PopulateEffort()` interpolates the piecewise-linear effort trajectory across all historical years for each simulation, then applies log-normal stochastic deviates scaled by `CV = 0.1`. The trajectory rises from 0 to 40–60% of maximum effort by 30% of the historical period, then reaches full effort by 60% of the period and remains stable thereafter. This contrasts with DomeExFleet, where effort continues to rise through to the final historical year.

**Catchability:**

Because the `Catchability` slot is empty, `PopulateCatchability()` sets catchability efficiency to 1 across all simulations and years. No trend (`qInc`) or inter-annual variability (`qCV`) is applied.

**Selectivity:**

`PopulateSelectivity()` samples `L5` and `LFS` for each simulation from uniform distributions defined by the prior bounds. Because `isRel = TRUE`, both are scaled by the maturity `L50` of the paired stock before computing the selectivity curve. `Vmaxlen = 1` is fixed, so selectivity reaches 1.0 at maximum length in all simulations, producing a strictly asymptotic curve. The resulting selectivity-at-length is converted to selectivity-at-age using the stock age–length key, and an area dimension is added.

**Retention:**

Because the `Retention` slot is empty, `PopulateRetention()` sets full retention (1.0) across all ages, lengths, simulations, and years.

**Discard Mortality:**

Because the `DiscardMortality` slot is empty, `PopulateDiscardMortality()` applies default discard mortality values. All discarded fish are assumed to survive (discard mortality = 0) unless overridden.

**Spatial Closures:**

Because `Closure` is `NULL`, `PopulateClosure()` applies no spatial or temporal closures; all areas are open to fishing in all years.

**Fleet Weight-at-Age:**

Because `WeightFleetRetained`/`WeightFleetSelected` are `NA`, `PopulateFleet()` computes them from selectivity, retention, and the stock's weight-at-length, so fleet-level landings/discards biomass calculations reflect the fleet's own selectivity and retention pattern rather than the stock's unselected mean weight-at-age.

## See Also

DomeExFleet, AlbacoreExStock, ButterfishExStock, `Fleet()`, `PopulateFleet()`, `Populate()`, fleet

## Examples

```
AsympExFleet

## Not run:
```

```
PopulatedStock <- PopulateStock(AlbacoreExStock, nYear = 50, pYear = 30, nSim = 48)
PopulatedFleet <- PopulateFleet(AsympExFleet, Stock = PopulatedStock, seed = 42)

## End(Not run)
```

---

`Atlantic_mackerel` *Data class objects*

---

## Description

Example objects of class Data

## Usage

```
Atlantic_mackerel

China_rockfish

Cobia

Example_datafile

Gulf_blue_tilefish

ourReefFish

Red_snapper

Simulation_1
```

## Format

An object of class `Data` of length 1.

An object of class `Data` of length 1.

An object of class `Data` of length 1.

An object of class `Data` of length 1.

An object of class `Data` of length 1.

An object of class `Data` of length 1.

An object of class `Data` of length 1.

An object of class `Data` of length 1.

## Examples

```
avail("Data")
```

`avail` *What objects of this class are available*

## Description

Generic class finder

## Usage

```
avail(classy, package = NULL, msg = TRUE, abc = FALSE)
```

## Arguments

| | |
|---|---|
| `classy` | A class of object (character string, e.g. 'Fleet') |
| `package` | Optional. Names(s) of the package to search for object of class `classy`. String Default is all `openMSE` packages. Always searches the global environment as well. |
| `msg` | Print messages? |
| `abc` | Logical, whether to alphabetize the results. By default, the function returns results found in the global environment, then core openMSE packages, and any additional packages in argument `package`. |

## Details

Finds objects of the specified class in the global environment or the openMSE packages.

## Author(s)

T. Carruthers

## See Also

Can Cant avail

## Examples

```
avail("OM", msg=FALSE)
Stocks <- avail("Stock")
Fleets <- avail("Fleet")
MPs <- avail("MP")
```

`Awatea2OM` *Reads MCMC estimates from Awatea (Paul Starr) processed r file structure into an operating model*

## Description

A function that generates an operating model from the MCMC samples of an Awatea model. Code optimized for the BC Pacific ocean perch assessment (Haigh et al. 2018).

## Usage

```
Awatea2OM(
  AwateaDir,
  nsim = 48,
  proyears = 50,
  Name = "OM made by Awatea2OM",
  Source = "No source provided",
  Author = "No author provided",
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `AwateaDir` | A folder with Awatea files |
| `nsim` | The number of simulations |
| `proyears` | The number of projection years for the MSE |
| `Name` | The name of the operating model |
| `Source` | Reference to assessment documentation e.g. a url |
| `Author` | Who did the assessment |
| `verbose` | Return detailed messages? |

## Details

This function averages biological parameters across sex and then sends arrays to VPA2OM, assumes unfished status (B/B0 = 1) in the first year, and assumes a single fishing fleet.

## Author(s)

Q. Huynh and T. Carruthers

## References

Haigh, R., et al. 2018. Stock assessment for Pacific Ocean Perch (*Sebastes alutus*) in Queen Charlotte Sound, British Columbia in 2017. Canadian Science Advisory Secretariat (CSAS) Research Document 2018/038. 232 pp. https://www.dfo-mpo.gc.ca/csas-sccs/Publications/ResDocs-DocRech/2018/2018_038-eng.html

`BAM2MOM` *Import a multi-stock, multi-fleet OM from a BAM object*

## Description

Import a multi-stock, multi-fleet OM from a BAM object

## Usage

```
BAM2MOM(
  rdat,
  nsim = 48,
  proyears = 50,
  interval = 1,
  stock_name = NULL,
  fleet_name = NULL,
  LowerTri = 0,
  report = FALSE,
  ...
)

BAM2OM(rdat, nsim = 48, proyears = 50, interval = 2, report = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `rdat` | A list object from the `BAMextras` package. Use `bamExtras::standardize_rdat(rdat)` |
| `nsim` | the number of simulations |
| `proyears` | the number of projection years |
| `interval` | the management interval |
| `stock_name` | Name of the stock(s) |
| `fleet_name` | Name of the fleet(s) |
| `LowerTri` | Integer. The number of recent years for which model estimates of recruitment are ignored (not reliably estimated by the assessment) |
| `report` | Logical, if TRUE, a diagnostic will be reported showing the matching of the OM reconstructed numbers at age vs the assessment. |
| `...` | Additional arguments passed to `MSEtool::Assess2MOM` |

## Value

An object of class `MOM`

## Functions

- `BAM2OM()`: Create a single stock/fleet OM from a BAM object

BAMFleetNames *Get and Fix BAM Fleet Names*

## Description

GetBAMFleetNames() extracts and classifies fleet names from a BAM stock assessment, separating retained (landings) fleets from discard fleets. FixFleetNames() is a helper that standardises raw BAM fleet name strings by stripping leading "L." prefixes and converting leading "D." prefixes to trailing ".D" suffixes.

## Usage

```
GetBAMFleetNames(Stock)

FixFleetNames(FleetNames)
```

## Arguments

| | |
|---|---|
| Stock | A character string matching a stock name available in bamExtras (e.g., 'Red Snapper'), or a list of BAM output objects containing elements rdat and dat. |
| FleetNames | A character vector of raw fleet name strings as extracted from BAM output. |

## Value

GetBAMFleetNames() returns a named list with four elements:

- $FleetNames: character vector of all fleet names, cleaned via FixFleetNames().
- $RetainFleets: character vector of landings (retained catch) fleets, i.e. all fleets not classified as discard fleets.
- $DiscardFleets: character vector of discard fleets, identified as those ending in ".D" or, if none found, starting with "D.".
- $FleetNamesOrig: character vector of original fleet names as they appear in BAMdata$parms, before cleaning.

FixFleetNames() returns a character vector the same length as FleetNames with standardised names.

## See Also

GetBAMOutput()

### Examples

```
## Not run:
GetBAMFleetNames('Red Snapper')

## End(Not run)

FixFleetNames(c('L.cHL', 'D.rHB', 'cPT'))
```

---

`bio_timeseries` *Extract Biological Time Series*

---

### Description

`Biomass()`, `SBiomass()`, and `SProduction()` extract total biomass, spawning biomass, and spawning production, respectively, from a hist or mse object. `VBiomass()` extracts vulnerable biomass (biomass weighted by fleet selectivity/retention).

### Usage

```
Biomass(
  object,
  df = TRUE,
  byAge = FALSE,
  byArea = FALSE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

SBiomass(
  object,
  df = TRUE,
  byAge = FALSE,
  byArea = FALSE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

SProduction(
  object,
  df = TRUE,
  byAge = FALSE,
  byArea = FALSE,
```

```
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

VBiomass(
  object,
  df = TRUE,
  byFleet = TRUE,
  type = c("Removals", "Landings"),
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)
```

## Arguments

| | |
|---|---|
| `object` | A hist or mse object. |
| `df` | Logical. If `FALSE` the raw array slot is returned. If `TRUE` (default) a tidy `data.frame` is returned. |
| `byAge` | Logical. If `TRUE` the data frame retains the `Age` dimension. Values are derived by multiplying numbers-at-age by the relevant biological schedule before optionally summing over age. Ignored when `df = FALSE`. Not applicable to `VBiomass()`. |
| `byArea` | Logical. If `TRUE` the data frame retains the `Area` dimension. Ignored when `df = FALSE`. Not applicable to `VBiomass()`. |
| `Reduce` | Logical. If `TRUE` (default) simulation dimensions are reduced using `ReduceDims()` before conversion to a data frame. |
| `IncYear` | Logical. Passed to `ReduceDims()`; controls whether the year dimension is retained during reduction. Default `FALSE`. |
| `Extend` | Logical. If `TRUE`, rows sharing one value across every simulation (e.g. the historical period, or any deterministic quantity – see `Reduce`/`ReduceDims()`) are broadcast to all `nSim` simulations via `ExtendSims()`, rather than appearing once with `Sim = 1`. Default `FALSE`. Ignored when `df = FALSE`. |
| `silent` | Logical. If `FALSE` and `Extend = FALSE`, emits a message when the returned data frame has rows sharing one value across every simulation, pointing at `Extend = TRUE`. Default `TRUE`. Ignored when `df = FALSE`. |
| `byFleet` | Logical. `VBiomass()` only. If `TRUE` (default) the data frame retains the `Fleet` dimension. Otherwise values are summed over fleets. |
| `type` | Character. `VBiomass()` only. `"Removals"` (default) weights biomass by `Selectivity x (Retention + (1 - Retention) x DiscardMortality)` (i.e. fish that die from the encounter, whether landed or discarded). `"Landings"` weights by `Selectivity x Retention` only. |

### Details

When applied to an mse object the historical and projection periods are row-bound and labelled via the `Period` column; historical rows carry `MP = "Historical"`.

### Value

- `df = FALSE` — the raw array slot (`Biomass`, `SBiomass`, or `SProduction`); for `VBiomass()`, for hist a numeric array `Sim x Stock x Fleet x Year`, or for mse a named list (`Historical` plus one element per MP) of such arrays.
- `df = TRUE` — a tidy `data.frame` with columns `Sim`, `Stock`, `Year`, `Period`, `MP` (MSE only), and optionally `Age` / `Area` (`Fleet` for `VBiomass()`), plus `Value`, `Variable`, and `Units` (`VBiomass()` omits `Units`).

### See Also

`Number()`, `ExtendSims()`

### Examples

```
Hist <- Simulate(SingleStockOM)
MSE <- Project(Hist, 'CurrentEffort')

# Tidy data frames
Biomass(Hist, df = TRUE)
Biomass(MSE,  df = TRUE)

# Retain age and area structure
Biomass(MSE, df = TRUE, byAge = TRUE, byArea = TRUE)

# Spawning biomass and production follow the same structure
SBiomass(MSE, df = TRUE, byAge = TRUE)
SProduction(MSE, df = TRUE, byArea = TRUE)
```

---

## `Bioeconomic` *Bioeconomic*

---

### Description

Construct and manipulate a bioeconomic object storing revenue, cost, and investment dynamics for fleet-level or stock-level bioeconomic analyses.

### Usage

```
Bioeconomic(
  Revenue = NULL,
  Cost = NULL,
  Investment = NULL,
  Disinvestment = NULL,
```

```
  Depreciation = NULL,
  Discount = NULL,
  Misc = list()
)

Bioeconomic(x) <- value

Revenue(x)

Revenue(x) <- value

Cost(x)

Cost(x) <- value

Investment(x)

Investment(x) <- value

Disinvestment(x)

Disinvestment(x) <- value

Depreciation(x)

Depreciation(x) <- value

Discount(x)

Discount(x) <- value
```

## Arguments

| | |
|---|---|
| `Revenue` | Numeric array or `NULL`. Revenue by simulation, fleet, and year. Default `NULL`. |
| `Cost` | Numeric or `NULL`. Operating cost per unit of effort. Default `NULL`. |
| `Investment` | Numeric or `NULL`. Cost of adding a unit of effort. Default `NULL`. |
| `Disinvestment` | Numeric or `NULL`. Cost of removing a unit of effort. Default `NULL`. |
| `Depreciation` | Numeric or `NULL`. Depreciation rate of effort units. Default `NULL`. |
| `Discount` | Numeric or `NULL`. Discount factor applied to future values. Default `NULL`. |
| `Misc` | List. Miscellaneous additional inputs. Default `list()`. |
| `x` | A bioeconomic object, or a fleet object for `Bioeconomic<-`. |
| `value` | For `Bioeconomic<-`: a bioeconomic object. For slot replacement functions: the new value for the corresponding slot. |

## Details

**Note:** The bioeconomic is not currently used by the MSE framework. It is included for future development and is available for custom analyses outside the standard model loop.

A bioeconomic object stores parameters and outputs for fleet-level bioeconomic modelling, including revenue, effort costs, and discounting. All slots are optional and default to `NULL`.

This class is not currently used by the MSE framework. It is included for future development.

### Attaching to a Fleet:

A bioeconomic object can be attached to a `Fleet()` with `Bioeconomic(Fleet) <- MyBioeconomic` and retrieved with `Bioeconomic(Fleet)`.

Individual slots may be accessed or modified using `Revenue()`, `Cost()`, `Investment()`, `Disinvestment()`, `Depreciation()`, and `Discount()`.

## Value

- `Bioeconomic()` returns a bioeconomic object. If `Revenue` is a fleet object, the `Bioeconomic` slot of that fleet is returned.
- `Bioeconomic<-` returns `x` with the `Bioeconomic` slot replaced by `value`.
- `Revenue()`, `Cost()`, `Investment()`, `Disinvestment()`, `Depreciation()`, `Discount()` return the corresponding slot from `x`.
- Their replacement forms return `x` with the corresponding slot updated.

## See Also

- bioeconomic for the class definition and slot-level documentation.
- `Fleet()` for the enclosing fleet constructor.

Other fleet: `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## Examples

```
b <- Bioeconomic(Cost = 100, Discount = 0.05)
Cost(b)
Cost(b) <- 200
Discount(b)

# Attach to a Fleet
f <- Fleet(Name = "Trawl")
Bioeconomic(f) <- Bioeconomic(Cost = 100, Investment = 500, Discount = 0.05)
Cost(Bioeconomic(f))
Discount(Bioeconomic(f))
```

## bioeconomic-class *The* bioeconomic *S4 Class*

### Description

Stores revenue, cost, and investment dynamics for fleet-level or stock-level bioeconomic analyses. Objects are typically created via the `Bioeconomic()` constructor, which documents all parameters in detail.

### Details

**Note:** The bioeconomic is not currently used by the MSE framework. It is included for future development.

### Slots

`Revenue numeric` array or `NULL`. Revenue by simulation, fleet, and year. See `Bioeconomic()`.

`Cost numeric` or `NULL`. Operating cost per unit of effort. See `Bioeconomic()`.

`Investment numeric` or `NULL`. Cost of adding a unit of effort. See `Bioeconomic()`.

`Disinvestment numeric` or `NULL`. Cost of removing a unit of effort. See `Bioeconomic()`.

`Depreciation numeric` or `NULL`. Depreciation rate of effort units. See `Bioeconomic()`.

`Discount numeric` or `NULL`. Discount factor applied to future values. See `Bioeconomic()`.

`Misc list`. Miscellaneous additional inputs.

### See Also

`Bioeconomic()` for the constructor and full parameter documentation. fleet for the enclosing fleet object.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## boxplot.Data *Boxplot of TAC recommendations*

### Description

Boxplot of TAC recommendations

### Usage

```
## S3 method for class 'Data'
boxplot(x, upq = 0.9, lwq = 0.1, ylim = NULL, outline = FALSE, col = NULL, ...)
```

### Arguments

| | |
|---|---|
| `x` | An object of class MSE |
| `upq` | Upper quantile of TACs for max ylim |
| `lwq` | Lower quantile of TACs for min ylim |
| `ylim` | Optional numeric vector of length 2 to specify limits of y-axis. |
| `outline` | Logical. Include outliers in plot? |
| `col` | Optional colours to pass to `boxplot` |
| `...` | Optional additional arguments passed to `boxplot` |

### Value

Returns a data frame containing the information shown in the plot

### Author(s)

A. Hordyk

---

## `ButterfishExStock` *Butterfish Example Stock*

---

### Description

An example stock object representing a Butterfish (*Peprilus triacanthus*) stock. This is a short-lived, fast-growing forage species with high natural mortality and wide uncertainty on recruitment and maturity. See also AlbacoreExStock for a contrasting long-lived, slow-growing example.

### Usage

```
ButterfishExStock
```

### Format

A stock object with the following slots populated:

- `Name`: `"Example Butterfish Stock"`.
- `CommonName`: `"Butterfish"`.
- `Species`: `"Peprilus triacanthus"`.
- `Ages`: An `Ages()` object with `MaxAge = 8`.
- `Length`: A `Length()` object with von Bertalanffy parameters — `Linf` (38–42 cm), `K` (0.16–0.24 yr^-1^), `t0` (-0.032 to -0.028 yr) — and `CVatAge` (0.1–0.15).
- `Weight`: A `Weight()` object with allometric parameters `alpha = 1.59e-05` and `beta = 3.1`.
- `NaturalMortality`: A `NaturalMortality()` object with `M` = 0.7–0.9 yr^-1^.

- `Maturity`: A `Maturity()` object with logistic maturity-at-length parameters `L50` (4.5–10.2 cm) and `L50_95` (1–8 cm). The wide ranges reflect substantial uncertainty in maturity schedule.
- `Fecundity`: An empty `Fecundity()` object; fecundity-at-age is computed internally during population (see Details).
- `SRR`: A `SRR()` object with Beverton-Holt steepness `h` (0.4–0.8), unfished recruitment `R0 = 1000`, recruitment variability `SD` (0.7–1.1), and autocorrelation `AC` (0.1–0.9). The high `SD` reflects the characteristically variable recruitment of forage species.
- `Spatial`: A `Spatial()` object with `UnfishedDist` (0.095–0.105), `ProbStaying` (0.4–0.6), and `RelativeSize` (0.095–0.105).
- `Depletion`: A `Depletion()` object with current biomass sampled from 5–60% of `B0`.
- `nYear`, `pYear`, `nSim`, `CurrentYear`, `Years`, `Seasons`: Set by `PopulateStock()`.
- `Misc`: Empty list for user-defined information.
- `Log`: Internal list for tracking object state.

## Details

Two-element numeric vectors in the slot descriptions above are uniform bounds. `PopulateStock()` samples from these bounds across simulations and expands the results into `[nSim x nAge x nYear]` arrays. The sections below describe what each component-level `Populate*()` function produces.

**Ages:**

`CalcAgeClasses()` generates the age vector `Ages@Classes` (`0:8`), which is used as the age dimension in all subsequent arrays.

**Length:**

`PopulateLength()` samples `Linf`, `K`, and `t0` independently for each simulation from uniform distributions defined by the bounds, then computes mean length-at-age using the von Bertalanffy growth equation. `CVatAge` is expanded across simulations and, when `ALK = TRUE`, an age–length key is constructed.

**Weight:**

`PopulateWeight()` applies the allometric relationship $W = \texttt{alpha} \times L\hat{}\texttt{beta}\hat{}$ to the mean length-at-age array to produce mean weight-at-age. An age–weight key is optionally constructed when `AWK = TRUE`.

**Natural Mortality:**

`PopulateNaturalMortality()` samples `M` for each simulation from the uniform prior and expands the result into a natural mortality-at-age array. Stochastic annual deviates are then added. The high `M` range (0.7–0.9 yr$\hat{}$-1$\hat{}$) is consistent with the short lifespan of this species.

**Maturity:**

`PopulateMaturity()` applies the logistic maturity-at-length curve, parameterised by `L50` and `L50_95`, to the populated length-at-age array to produce a maturity-at-age array. The wide priors on both parameters result in high inter-simulation variability in the maturity schedule.

**Fecundity:**

Because the `Fecundity` slot is empty, `PopulateFecundity()` computes fecundity-at-age as the element-wise product of weight-at-age and maturity-at-age, representing spawning output per individual.

**Stock–Recruitment:**

`PopulateSRR()` samples `h` for each simulation from the uniform prior and generates a time series of log-normal recruitment deviates with standard deviation `SD` and autocorrelation `AC`. The high `SD` (0.7–1.1) produces strongly variable recruitment, typical of forage fish.

**Spatial:**

`PopulateSpatial()` samples `UnfishedDist`, `ProbStaying`, and `RelativeSize` for each simulation and uses them to construct a full inter-area movement matrix, then expands all spatial arrays across simulations and years. The moderate `ProbStaying` (0.4–0.6) allows substantial mixing between the two areas.

**Depletion:**

`PopulateDepletion()` samples current biomass depletion relative to `B0` for each simulation from the uniform prior, setting the initial conditions of the operating model.

## See Also

AlbacoreExStock, `SeasonalSpatialExStock()`, `Stock()`, `PopulateStock()`, `Populate()`, stock

## Examples

```
ButterfishExStock

## Not run:
PopulatedStock <- PopulateStock(
  ButterfishExStock,
  nYear = 50,
  pYear = 30,
  nSim = 48,
  seed = 42
)

## End(Not run)
```

---

`CalcAgeClasses` *Calculate Age Classes*

---

## Description

Computes the numeric vector of age classes (in years) from an `Ages()` object, respecting sub-annual time steps.

### Usage

```
CalcAgeClasses(Ages)
```

### Arguments

| | |
|---|---|
| `Ages` | An `Ages()` object with slots `MinAge`, `MaxAge`, and `Units` set. |

### Details

Age classes are always expressed in years, regardless of the time unit used to define `MinAge` and `MaxAge`. `Units` controls the within-year resolution via `CalcSeasons()`, which maps unit strings to a seasons divisor:

| `Units` | Seasons | Step size (yr) |
|---|---|---|
| `"year"` | 1 | 1 |
| `"half-year"` | 2 | 0.5 |
| `"quarter"` | 4 | 0.25 |
| `"month"` | 12 | 0.0833 |
| `"week"` | 52 | 0.0192 |

The sequence runs from `MinAge / Seasons` to `MaxAge / Seasons` in steps of `1 / Seasons`, rounded to three decimal places.

Returns `NULL` if `Units`, `MinAge`, or `MaxAge` are not set.

### Value

A numeric vector of age classes in years, or `NULL` if any of `Ages@Units`, `Ages@MinAge`, or `Ages@MaxAge` are empty.

### See Also

`Ages()`, ages, `PopulateStock()`

### Examples

```
# Annual age classes 0-5
CalcAgeClasses(Ages(MinAge = 0, MaxAge = 5, Units = "year"))
#> [1] 0 1 2 3 4 5

# Quarterly age classes 0-1
CalcAgeClasses(Ages(MinAge = 0, MaxAge = 4, Units = "quarter"))
#> [1] 0.00 0.25 0.50 0.75 1.00
```

`CalcAgeSizeKey` *Calculate an Age-Size Key*

## Description

Generates an Age-Size key given mean and standard deviation of size-at-age. The size-at-age can be normally or log-normally distributed. By default the distribution is truncated at 2 standard deviations.

## Usage

```
CalcAgeSizeKey(
  MeanAtAge,
  CVatAge,
  Classes,
  TruncSD = 2,
  Dist = c("normal", "lognormal"),
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| `MeanAtAge` | Numeric vector, matrix, or 3D array of mean size-at-age, or a `Length()` object. |
| `CVatAge` | Numeric vector, matrix, or 3D array of coefficient of variation (CV) at age. Same structure as `MeanAtAge`. |
| `Classes` | Numeric vector of lower bounds of size classes. Must be strictly increasing and non-negative. Bins need not be equal width. |
| `TruncSD` | Numeric value for the number of standard deviations at which the distribution is truncated. |
| `Dist` | Character string, either `"normal"` or `"lognormal"` indicating the distribution of size-at-age. |
| `silent` | Logical; if `TRUE`, suppress progress bar. |

## Details

Each bin `k` spans `[Classes[k], Classes[k+1])`. The final bin captures all probability above `Classes[nClass]`. Bins need not be equal width.

## Value

A 4D array with dimensions `Sim`, `Age`, `Class`, and `Year`.

## Examples

```
nSim <- 5
nAge <- 15
nYear <- 2
ages <- 0:(nAge-1)
years <- 2001:2002

TruncSD <- 2
Dist <- 'normal'

set.seed(123)
Linf <- runif(nSim, min = 150, max = 200)
K <- 0.3
t0 <- 0

MeanAtAge <- array(0,
                   dim = c(nSim, nAge, nYear),
                   dimnames = list(Sim = 1:nSim, Age = ages, Year = years))

for (sim in 1:nSim) {
  for (yr in 1:nYear) {
    MeanAtAge[sim, , yr] <- Linf[sim] * (1 - exp(-K * (ages - t0)))
  }
}

# Coefficient of variation (CV)
CVatAge <- array(0.1, dim = dim(MeanAtAge), dimnames = dimnames(MeanAtAge))

# Define size classes
SDatAge <- MeanAtAge * CVatAge
class_min <- floor(min(MeanAtAge - TruncSD * SDatAge))
class_max <- ceiling(max(MeanAtAge + TruncSD * SDatAge))

Classes <- seq(class_min, class_max, by = 10)


# Calculate Age-Size Key
ASK <- CalcAgeSizeKey(
  MeanAtAge = MeanAtAge,
  CVatAge = CVatAge,
  Classes = Classes,
  TruncSD = TruncSD,
  Dist = Dist
)

dim(ASK)


# Inspect first simulation, first year
ASK[1,,,1]
```

`CalcAsymDist` *Calculate the Asymptotic Spatial Distribution of a Movement Matrix*

## Description

Computes the stationary (asymptotic) distribution of a Markov chain movement matrix — the long-run proportion of the population expected to occupy each area. For a 2-area matrix a closed-form solution is used. For larger matrices, either a linear-algebra or iterative Markov chain approach is available.

## Usage

```
CalcAsymDist(Movement, method = c("la", "mc"), tol = 1e-10, maxiter = 10000)
```

## Arguments

| | |
|---|---|
| `Movement` | A square row-stochastic matrix of dimensions `nArea × nArea`, where `Movement[i, j]` is the probability of moving from area `i` to area `j`. All rows must sum to 1. |
| `method` | Character. Solution method for matrices larger than 2 × 2. One of `"la"` (linear algebra; default) or `"mc"` (Markov chain iteration). Partially matched. |
| `tol` | Numeric. Convergence tolerance. Used to check row sums, detect reducible 2-area chains, clamp near-zero negative probabilities, and assess convergence of the `"mc"` method. Default `1e-10`. |
| `maxiter` | Integer. Maximum number of iterations for the `"mc"` method. A warning is emitted if convergence is not reached. Default `1e4`. |

## Details

The movement matrix is treated as a row-stochastic transition matrix, where `Movement[i, j]` is the probability of moving from area `i` to area `j`. Diagonal entries represent the probability of remaining in an area. All rows must sum to 1.

Two solution methods are available for matrices larger than 2 × 2 via the `method` argument:

- `"la"` (default): solves the system `pi * P = pi` subject to `sum(pi) = 1` using least-squares linear algebra (`solve(t(A) %*% A, t(A) %*% B)`). Exact and fast for well-conditioned matrices.
- `"mc"`: iteratively multiplies the current distribution by `Movement` until convergence. More robust for nearly-reducible chains but slower.

For 2-area matrices the stationary distribution is computed analytically regardless of `method`:

`pi[1] = P[2,1] / (P[1,2] + P[2,1])`

`pi[2] = P[1,2] / (P[1,2] + P[2,1])`

### Value

A numeric vector of length `nArea` giving the stationary probability of occupying each area. Values sum to 1.

### See Also

`Spatial()`, `CalcUnfishedDist()`

### Examples

```
# 2-area movement matrix
M <- matrix(c(0.8, 0.2,
              0.3, 0.7), nrow=2, byrow=TRUE)
CalcAsymDist(M)

# 3-area movement matrix using linear algebra (default)
M3 <- matrix(c(0.7, 0.2, 0.1,
               0.1, 0.8, 0.1,
               0.2, 0.2, 0.6), nrow=3, byrow=TRUE)
CalcAsymDist(M3)

# Using Markov chain iteration
CalcAsymDist(M3, method="mc")
```

---

## CalcBLow *Calculate the BLow Reference Point*

---

### Description

Computes `BLow`: the current spawning biomass at which it would take `HZN` mean generation times to rebuild to `Bfrac x SBMSY` under zero future catch. Stored in `Hist@Reference@BLow`.

### Usage

```
CalcBLow(Hist, HZN = 2, Bfrac = 0.5, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `Hist` | A hist or om object. If an `om` is supplied it is first converted to a `hist` object. |
| `HZN` | Numeric. Number of mean generation times (see `CalcMGT()`) defining the rebuilding horizon. Default `2`. |
| `Bfrac` | Numeric. Target spawning biomass as a fraction of `SBMSY` (see `SBMSY()`). Default `0.5`. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

For each simulation and stock, the current (last historical year) numbers-at-age are scaled by a factor `k` and projected forward under zero fishing effort for `round(HZN x MGT)` years. `k` is found via `stats::optimize()` such that the projected spawning biomass at the end of that horizon equals `Bfrac x SBMSY`. `BLow` is `k` times the current spawning biomass.

Requires `Hist@Reference@MSY` (see `CalcMSY()`) and enough projection years in the operating model to cover the rebuilding horizon for every simulation/stock; horizons exceeding the available projection years are capped with a warning.

This is substantially more expensive than the other reference points in `CalcRefPoints()` (a full population projection plus a numerical search, repeated per simulation and stock) and is not included by `CalcRefPoints()`'s default reference-point set.

### Value

The input hist object with `Hist@Reference@BLow` populated.

### See Also

`CalcMGT()`, `CalcMSY()`, `CalcRefPoints()`, reference

---

## `CalcEquilibrium` *Calculate Absolute Equilibrium Quantities*

---

### Description

For a given vector of apical fishing mortality values, computes absolute equilibrium abundance and yield by combining per-recruit calculations with stock-recruitment relationship scaling. Results span all complexes defined in the operating model.

### Usage

```
CalcEquilibrium(OM, apicalF = NULL, Years = NULL, Complex = NULL)
```

### Arguments

| | |
|---|---|
| `OM` | An om or hist object. If a hist object is supplied, `Hist@OM` is used. |
| `apicalF` | Numeric vector of apical fishing mortality values at which equilibrium quantities are evaluated. If `NULL` (default), a log-spaced sequence of 50 values is generated automatically. |
| `Years` | Numeric years for which equilibrium quantities are evaluated. Biological and fishery parameters are subset to this year. Must match years in `Years(OM)`. If `NULL` (default), the final historical year is used. |
| `Complex` | Character vector of complex names to evaluate. If `NULL` (default), all complexes are evaluated. |

## Details

**Default apicalF grid:**
When `apicalF = NULL`, a log-spaced sequence of 30 values is generated automatically, spanning from `0.01 × max(M)` to `2 × max(M)` with zero prepended, where `max(M)` is the maximum natural mortality across all stocks.
Log-spacing is used because population dynamics are approximately linear in log(F): yield and biomass change rapidly at low F relative to M but become increasingly insensitive at high F, so equal spacing on the log scale concentrates resolution where the equilibrium curve is most informative.
Supply an explicit vector to override, for example `apicalF = seq(0, 2, by = 0.01)`.

**Complexes:**
Per-recruit and equilibrium quantities are calculated separately for each complex, with `apicalF` defined as the maximum fishing mortality across all stocks within the complex.

**Single year:**
Unlike `CalcPerRecruit()`, this function accepts only a single `Year` value. For multi-year equilibrium curves, call the function separately for each year.

## Value

An equilibrium object containing absolute equilibrium numbers, biomass, spawning biomass, spawning production, removals, and landings evaluated at each value of `apicalF`, with results reported per stock.

## See Also

`CalcPerRecruit()`, `CalcMSY()`, equilibrium

---

# CalcIndexResiduals *Calculate log residuals for standardized index, fitting hyperstability (beta)*

---

## Description

Computes log-scale residuals under the hyperstability model `Observed_t = Efficiency * Simulated_t^Beta * exp(residual_t)` (Harley et al. 2001). `Efficiency` is fit per simulation by log-linear regression; `Beta` estimation (or fixing) is delegated to `EstimateBeta()`.

## Usage

```
CalcIndexResiduals(
  ObservedIndex,
  SimulatedIndex,
  beta = NULL,
  MinPoints = 8,
  FitInd = NULL,
  alpha = 0.05
)
```

### Arguments

| | |
|---|---|
| `ObservedIndex` | Numeric vector of length nYear with observed index values. |
| `SimulatedIndex` | Numeric matrix or array of dimensions nSim x nYear with simulated index values. |
| `beta` | `NULL`, numeric scalar, or length-nSim vector. Passed through to `EstimateBeta()`. |
| `MinPoints` | Integer. Passed through to `EstimateBeta()`. Default `8`. |
| `FitInd` | Integer vector or `NULL`. Year positions to restrict fitting to (e.g. `Index_Obs@Years`). `NULL` (default) uses every non-NA, positive year. |
| `alpha` | Numeric. Passed through to `EstimateBeta()`. Default `0.05`. |

### Details

`NA` or non-positive years are dropped before fitting. See `EstimateBeta()` for when `Beta` is fixed at `1` rather than freely estimated.

### Value

A list with elements:

- `LogResiduals`: matrix nSim x nYear of log residuals (`log(Observed) - fitted log(Observed)`)
- `Beta`: numeric vector length nSim, the fitted or supplied beta values.
- `Efficiency`: numeric vector length nSim, the fitted catchability `q`.
- `SE_Beta`, `CI_Lower`, `CI_Upper`, `R2`, `PValue`, `nPoints`, `Status`: fit diagnostics from `EstimateBeta()`, passed through unchanged.

---

## `CalcInnovationCov` *Calculate Innovation Covariance Matrix for Multivariate AR(1)*

---

### Description

Computes the innovation covariance matrix ($\Sigma_\epsilon$) for a multivariate AR(1) process given the stationary covariance matrix ($\Sigma_Z$) and lag-1 autocorrelation coefficients ($\phi$).

### Usage

```
CalcInnovationCov(Sigma_Z, phi)
```

### Arguments

| | |
|---|---|
| `Sigma_Z` | Numeric covariance matrix (`nStock` x `nStock`) representing the stationary covariance of the process. |
| `phi` | Numeric vector of length `nStock` containing lag-1 autocorrelation coefficients for each stock. |

### Details

The relationship used is:

$$\Sigma_\epsilon = \Sigma_Z - \Phi\Sigma_Z\Phi^T$$

where $\Phi$ is a diagonal matrix of autocorrelation coefficients.

Only stocks with positive stationary variance (active stocks) are included in the computation. The result is embedded back into a full zero matrix, so inactive stocks contribute no innovation variance.

The active submatrix is forced to be symmetric and positive semi-definite. If negative eigenvalues are detected, `Matrix::nearPD()` is used to find the nearest PSD matrix. After correction, the diagonal is capped at the theoretical upper bound $\sigma^2_{Z,i}(1-\phi_i^2)$ to prevent `Matrix::nearPD()` from inflating innovation variance beyond what is consistent with the stationary distribution.

### Value

A numeric covariance matrix (`nStock` x `nStock`) representing the innovation covariance ($\Sigma_\epsilon$). Rows and columns corresponding to inactive stocks (zero diagonal in `Sigma_Z`) are set to zero.

### See Also

`GenerateStockTargeting()`, `FitStockTargeting()`

### Examples

```
Sigma_Z <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
phi <- c(0.7, 0.6)

CalcInnovationCov(Sigma_Z, phi)
```

---

## CalcMGT *Mean Generation Time*

---

### Description

Calculates the mean generation time (MGT), defined as the mean age of a mature female in the population at unfished equilibrium under natural mortality only. MGT is used as the time unit for the $B_{\text{Low}}$ reference point calculation.

### Usage

```
CalcMGT(OM, Years = NULL, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `OM` | An om or hist object. |
| `Years` | Numeric; calendar year(s) at which MGT is evaluated. If `NULL` (default), the final historical year is used. For seasonal models, specify integer calendar years; all seasons within each year are used automatically. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

MGT is computed as:

$$\text{MGT} = \frac{\sum_a a \cdot l_a \cdot m_a}{\sum_a l_a \cdot m_a}$$

where $l_a$ is survivorship to age $a$ under natural mortality only (with $l_1 = 1$) and $m_a$ is maturity-at-age. In seasonal models age classes are in decimal-year units (e.g. 0.25, 0.5, ...) and $M_a$ is the per-season instantaneous rate, so the formula gives MGT in years consistent with an equivalent annual model.

Parameters are taken from the last season of each requested calendar year. If M or maturity vary by year, MGT will vary correspondingly.

### Value

An array with dimensions `[Sim, Stock, Year]` containing MGT in years. Dimensions where values are identical across simulations or years are collapsed by `ReduceDims()`. Like `CalcSPR0()` and `CalcMSY()`, `CalcMGT()` does not itself store the result – assign it to `Hist@Reference@MGT` if needed.

### See Also

`CalcSPR0()`, reference

---

## `CalcMSY` *Calculate MSY Reference Points*

---

### Description

Computes maximum sustainable yield (MSY) biological reference points for all complexes in a hist or om object by identifying the apical fishing mortality that maximises total yield, accounting for the stock-recruitment relationship. Results are stored in `Hist@Reference@MSY` and the updated hist object is returned.

### Usage

```
CalcMSY(
  Hist,
  Years = NULL,
  type = c("Removals", "Landings"),
  parallel = FALSE,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `Hist` | A hist or om object. If an `om` is supplied it is first converted to a `hist` object. |
| `Years` | Integer vector of years for which reference points are evaluated. Biological and fishery parameters are subset to these years. If `NULL` (default), the final historical year is used. |
| `type` | Character. Whether MSY is defined in terms of total removals (landings plus dead discards) or landings only. One of `'Removals'` (default) or `'Landings'`. |
| `parallel` | Logical. If `TRUE`, calculates reference points across simulations in parallel using a `future` plan established by `SetupParallel()`. Falls back to sequential execution with a warning if no parallel plan is active. Default `FALSE`. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

#### Complexes:

Reference points are calculated separately for each complex defined in `Hist@OM@Complexes`. Within each complex:

- `FMSY` is the single apical fishing mortality maximising total yield across all stocks in the complex.
- Fleet-specific F is distributed according to effort-weighted catchability and selectivity.
- Stock-level quantities (`BMSY`, `SBMSY`, etc.) are reported for each stock evaluated at the complex-level `FMSY`.

#### Seasonal models:

For seasonal models (`Seasons > 1`) the reference points are annual quantities: `FMSY` is the annual apical fishing mortality (seasonal fishing mortality at age summed over the seasons of the year, maximum over ages), `MSYLandings`/`MSYDiscards` are annual totals, and `BMSY`/`SBMSY`/`SPMSY` are the biomass or spawning production in the reference season(s) (see `RefSeason()`).

### Value

An a refpointsMSY object.

### See Also

`CalcPerRecruit()`, `CalcSPR0()`, refpointsMSY, `SetupParallel()`

---

`CalcPerRecruit` *Calculate Per-Recruit Quantities*

---

### Description

Extracts biological and fishery parameters from a hist or om object and evaluates per-recruit quantities at a given apical fishing mortality.

### Usage

```
CalcPerRecruit(OM, apicalF = 0.1, Years = NULL, Complex = NULL)
```

### Arguments

| | |
|---|---|
| `OM` | A om or hist object. If an `om` is supplied it is first converted to a `hist` object. |
| `apicalF` | Numeric vector of apical fishing mortality values at which per-recruit quantities are evaluated. Default `0.1`. |
| `Years` | Integer vector of years for which per-recruit quantities are evaluated. Biological and fishery parameters are subset to these years If `NULL` (default), the final historical year is used. |
| `Complex` | Character vector of complex names to evaluate. If `NULL` (default), all complexes are evaluated. |

### Details

**Complexes:**

Per-recruit quantities are calculated separately for each complex, with `apicalF` defined as the maximum fishing mortality across all ages and stocks within the complex. Results are reported per stock and reassembled into a single perrecruit object spanning all stocks. The `Complex` argument can be used to restrict calculations to a subset of complexes.

**Fleet allocation:**

The relative contribution of each fleet to total fishing mortality is computed as the effort-weighted catchability (i.e. `Effort × Efficiency`) normalised across fleets. These proportions are used to distribute `apicalF` across fleets before selectivity is applied.

**Years:**

When `Years` contains multiple values, biological and fishery parameters are subset to those years and the per-recruit calculations are evaluated for each year independently. When `Years = NULL`, only the final historical year is used, giving a single set of per-recruit quantities representative of current conditions.

### Value

A perrecruit object containing numbers-per-recruit, spawning-per-recruit, biomass-per-recruit, removals, and landings evaluated at each value of `apicalF`, with results reported per stock and apical F defined as the maximum F across all stocks within each complex.

### See Also

`CalcSPR0()`, perrecruit

---

## CalcRefPoints *Calculate Per-Recruit and Replacement-Line Reference Points*

---

### Description

Computes `F0.1`, `Fmax`, `Fx%SPR`, `Fmed`, `Fcrash`, and `SPRcrash` biological reference points and stores them in `Hist@Reference`.

### Usage

```
CalcRefPoints(
  Hist,
  Years = NULL,
  SPRTarget = c(0.3, 0.4, 0.5),
  type = c("Removals", "Landings"),
  nF = 100,
  BLow = FALSE,
  HZN = 2,
  Bfrac = 0.5,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `Hist` | A hist or om object. If an `om` is supplied it is first converted to a `hist` object. |
| `Years` | Integer vector of years for which reference points are evaluated. If `NULL` (default), the final historical year is used. |
| `SPRTarget` | Numeric vector of SPR targets (e.g. `0.4` for F40%) at which to evaluate `Fx%SPR`. Default `c(0.3, 0.4, 0.5)`. |
| `type` | Character. Whether yield-per-recruit is defined in terms of total removals or landings only. One of `'Removals'` (default) or `'Landings'`. |
| `nF` | Integer. Number of apical F values in the log-spaced search grid spanning `0` to `Hist@OM@maxF`. Default `100`. |

| | |
|---|---|
| `BLow` | Logical. Also calculate the `BLow` reference point (see `CalcBLow()`)? Default `FALSE`. Substantially more expensive than the other reference points here – a full population projection plus a numerical search, repeated per simulation and stock – so it is kept as a separate opt-in rather than bundled into the default set. |
| `HZN`, `Bfrac` | Passed to `CalcBLow()` when `BLow = TRUE`. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

All reference points except `BLow` are evaluated on the same per-recruit apical-F grid via `CalcPerRecruit()`, reusing a single evaluation rather than re-optimizing per reference point.

- `F0.1`: apical F at which the slope of the yield-per-recruit curve is 10% of the slope at the origin.
- `Fmax`: apical F maximising yield-per-recruit. Undefined (returned as the upper grid bound) for asymptotic selectivity, where yield-per-recruit has no interior maximum.
- `Fx%SPR`: apical F at which spawning-per-recruit relative to unfished equals each value in `SPRTarget`.
- `Fcrash`/`SPRcrash`: the apical F (and corresponding relative SPR) at which the per-recruit replacement line crosses the compensation ratio implied by steepness, evaluated against a fixed reference spawning-per-recruit at the first historical year – matching the fixed-alpha/beta stock-recruitment curve. Only defined for `BevertonHolt` and `Ricker` stock-recruitment models.
- `Fmed`: apical F at which the per-recruit replacement line equals the median historical recruits-per-spawner ratio.
- `BLow` (optional): see `CalcBLow()`.

### Value

The input hist object with `Hist@Reference@F01`, `@FMax`, `@FSPR`, `@FMed`, `@FCrash`, `@SPRcrash`, and (if `BLow = TRUE`) `@BLow` populated.

### See Also

`CalcPerRecruit()`, `CalcMSY()`, `CalcBLow()`, reference, `F01()`, `FMax()`, `FSPR()`, `FMed()`, `FCrash()`, `SPRcrash()`

---

## `calcRefYield` *Calculate Reference Yield*

---

### Description

Calculate Reference Yield

### Usage

```
calcRefYield(x, StockPars, FleetPars, pyears, Ncurr, nyears, proyears)
```

### Arguments

| | |
|---|---|
| x | Integer, the simulation number |
| StockPars | List of Stock Parameters |
| FleetPars | List of Fleet Parameters |
| pyears | The number of years to project forward. Equal to 'nyears' for optimizing for q. |
| Ncurr | Array with current numbers-at-age (dim=c(nsim, maxage+1, nareas)) |
| nyears | Number of historical years |
| proyears | Number of projection years |

### Author(s)

A. Hordyk

---

## CalcResidualStats *Calculate residual statistics for log-space index residuals*

---

### Description

Computes standard deviation and lag-1 autocorrelation of log residuals for a simulated index (`sim x year`).

### Usage

```
CalcResidualStats(LogResiduals, nSeasons = 1)
```

### Arguments

| | |
|---|---|
| LogResiduals | A numeric matrix or array with dimensions `sim x year`. |
| nSeasons | Number of seasons in the array (default 1). |

### Details

Seasons that are NA in every year are treated as seasons the index is never observed in, and are dropped; the seasons they occupy are recorded in `NA_Season` so `GenResiduals()` can reproduce the same pattern in the projection. The remaining values are taken in time order, and `AC` is the lag-1 autocorrelation between consecutive observations. For an index observed in a single season, that is the year-to-year autocorrelation. This matches how `ApplyAC()` propagates the projection residuals, which steps from one observation to the next.

`AC` may be negative, and is returned as estimated.

### Value

A data.frame with columns:

- `AC`: lag-1 autocorrelation between consecutive observations, or `NA` when fewer than two observations are available
- `SD`: standard deviation of residuals
- `NA_Season`: a list length `nSim` with each element containing integer vector indicating which (if any) seasons have NA values for all years

### See Also

`ApplyAC()`, `GenResiduals()`

---

## `CalcSPR0` *Unfished Spawning Production Per Recruit*

---

### Description

Calculates the unfished spawning production per recruit (SPR0) — the denominator of the spawning potential ratio — under equilibrium conditions. SPR0 is computed as the element-wise ratio of unfished spawning production `SP0()` to unfished recruitment `R0()`, with array broadcasting handled by `ArrayDivide()`.

### Usage

```
CalcSPR0(OM, silent = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `OM` | A stock, om, or hist object. For a stock or om object, only the stock components are populated and used, so the OM does not need `Fleet`, `Obs`, or `Imp` objects. A stock object is placed in an OM created with `OM()`, using the arguments in `....` If a hist object is provided (the output of `Simulate()`), it is used directly.<br>SPR0 uses only the spawning timing in `SRR` (`SpawnTimeFrac` and `SpawnLag`), so the `SRR` object may be left unspecified or partially specified: missing `Pars`, `R0`, and `SD` are not required. |
| `silent` | Logical. If `TRUE`, suppresses progress messages during population. Default is `FALSE`. |
| `...` | Arguments passed to `OM()` when `OM` is a stock object, e.g., `nSim`, `nYear`, `pYear`, `CurrentYear`, and `Seasons`. |

### Value

An array with dimensions `[Sim, Stock, Year]` containing the unfished spawning production per recruit. The `Sim` dimension is collapsed by `ReduceDims()` when values are identical across simulations.

### See Also

SPO(), RO(), ArrayDivide()

---

CalcSurvival *Calculate Cumulative Survival-at-Age*

---

### Description

Computes cumulative survival from recruitment to each age class, accounting for natural mortality, optional fishing mortality, spawn timing within a time step, and optional semelparous mortality. The calculation follows:

### Usage

```
CalcSurvival(
  NaturalMortality,
  FishingMortality = NULL,
  PlusGroup = TRUE,
  SpawnTimeFrac = 0,
  Semelparous = FALSE
)
```

### Arguments

| | |
|---|---|
| NaturalMortality | Numeric array of natural mortality-at-age. Either a 2D array (Age × Year) or 3D array (Sim × Age × Year), with named dimensions. |
| FishingMortality | Optional numeric array of fishing mortality-at-age, matching the dimensions of NaturalMortality. If NULL (default), only natural mortality is used. |
| PlusGroup | Logical. If TRUE (default), the plus-group survival is adjusted for ongoing mortality beyond the maximum age class. |
| SpawnTimeFrac | Numeric. Fractional position of spawning within a time step, between 0 (start; default) and 1 (end). Either length 1 (applied to all simulations) or length nSim. |
| Semelparous | Either FALSE (default) to ignore semelparous mortality, or a numeric array of semelparous mortality-at-age matching the dimensions of NaturalMortality. |

### Details

- **Age 1**: S[1] = exp(-Z[1] * SpawnTimeFrac)
- **Age a**: S[a] = S[a-1] * exp(-(Z[a-1] * (1 - f) + Z[a] * f)) * (1 - Semelparous[a-1])

where Z = M + F and f = SpawnTimeFrac.

If PlusGroup = TRUE, the plus-group age class is adjusted to account for ongoing mortality: S[nAge] = S[nAge] / (1 - exp(-Z[nAge])).

### Value

A numeric array of cumulative survival-at-age with the same dimensions and dimnames as `NaturalMortality`. Each value represents the probability of surviving from recruitment to that age class in that year.

---

`CalcTSUnits` *Convert Between Season Count and Time Unit Labels*

---

### Description

Two complementary functions for converting between the integer number of seasons per year and the corresponding time unit label used throughout the package:

### Usage

```
CalcTSUnits(Seasons)

CalcSeasons(Units)
```

### Arguments

| | |
|---|---|
| `Seasons` | Integer. Number of seasons per year. Must be one of `1`, `2`, `4`, `12`, `52`, or `365`. `NULL` is treated as `1` (annual). (`CalcTSUnits()` only.) |
| `Units` | Character. Time unit label. Case-insensitive. Must be one of `"year"`, `"half-year"`, `"quarter"`, `"month"`, or `"week"`. (`CalcSeasons()` only.) |

### Details

- `CalcTSUnits()`: converts a season count to a time unit label string.
- `CalcSeasons()`: converts a time unit label string to a season count.

Supported conversions:

| Seasons | Units |
|---|---|
| 1 | `"year"` |
| 2 | `"half-year"` |
| 4 | `"quarter"` |
| 12 | `"month"` |
| 52 | `"week"` |
| 365 | `"day"` |

### Value

- `CalcTSUnits()`: a character string time unit label.
- `CalcSeasons()`: an integer season count.

### See Also

CalcYears()

### Examples

```
CalcTSUnits(1)    # "year"
CalcTSUnits(4)    # "quarter"
CalcTSUnits(NULL) # "year"

CalcSeasons("year")    # 1
CalcSeasons("Quarter") # 4
```

---

## CalcUnfished_Dynamic *Calculate Dynamic Unfished Population Dynamics*

---

### Description

Calculate dynamic unfished population quantities for all Stock() objects in an operating model. Quantities include unfished number-at-age, total biomass, spawning biomass, and spawning production, evaluated dynamically over historical and projection years.

### Usage

```
CalcUnfished_Dynamic(Hist, IdenticalHist = NULL, silent = FALSE)
```

### Arguments

| | |
|---|---|
| Hist | A hist object, or an om object from which a hist object is built. |
| IdenticalHist | Logical; if TRUE, assumes all simulations are identical and calculates only a single simulation. If NULL (default), determined automatically by checking whether all simulations are identical. |
| silent | Logical; if TRUE, suppress messages during calculation. |

### Value

A popdynamics object containing dynamic unfished Number, Biomass, SBiomass, and SProduction arrays with dimensions Sim × Stock × Year.

### See Also

CalcUnfished_Equilibrium()

---

`CalcUnfished_Equilibrium`
*Calculate Equilibrium Unfished Population Dynamics*

---

## Description

Calculate equilibrium unfished population quantities for all `Stock()` objects in an operating model. Quantities include unfished number-at-age, total biomass, spawning biomass, and spawning production, evaluated under equilibrium unfished conditions.

## Usage

```
CalcUnfished_Equilibrium(OM, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `OM` | An `OM()` or `Hist()` object. |
| `silent` | Logical; if `TRUE`, suppress messages during calculation. |

## Details

If a `Hist()` object is supplied, its embedded operating model is used. The operating model is populated if required prior to calculation.

- Biomass is calculated as the sum over ages of unfished number-at-age multiplied by mean weight-at-age.
- Spawning biomass additionally applies maturity-at-age.
- Spawning production is calculated using fecundity-at-age and may be reassigned across stocks using the `SPFrom` slot of the stocks `SRR()` objects.

## Value

A popdynamics object. `Number` is a list of length `nStock`, each element an array with dimensions `Sim × Age × Year × Area`. `Biomass`, `SBiomass`, and `SProduction` are arrays with dimensions `Sim × Stock × Year` (summed over ages and areas).

## See Also

`CalcUnfished_Dynamic()`

`CalcUnfishedDist` *Calculate the Unfished Spatial Distribution*

## Description

Computes the unfished spatial distribution for each simulation, age class, and year from `Spatial@Movement`. The result is stored in `Spatial@UnfishedDist`.

## Usage

```
CalcUnfishedDist(Spatial, Ages = NULL, Years = NULL)
```

## Arguments

| | |
|---|---|
| `Spatial` | A `Spatial()` object with a populated `Movement` slot. The `Movement` array must have dimensions `Sim × FromArea × ToArea × Age × Year`. |
| `Ages` | An ages object supplying age class labels. If `NULL` (default), `DefaultAges()` is used. |
| `Years` | Numeric vector of year labels. If `NULL` (default), `DefaultYears()` is used. |

## Details

For a single-area model, all values are set to 1. For multi-area models, two modes are available:

- **Age-invariant movement** (`Movement` Age dimension = 1): the stationary (asymptotic) distribution of the single movement matrix is computed and applied uniformly across all age classes using `CalcAsymDist()`.
- **Age-varying movement** (`Movement` Age dimension > 1): a sequential chain propagates the analyst-specified recruit distribution forward through age classes. For age $a$:

$$\boldsymbol{\pi}_a = \boldsymbol{\pi}_{a-1}\mathbf{M}_{a-1}$$

  The plus-group distribution is computed as the stationary distribution of `M[nAge]`, since fish remain in that age class across multiple time steps. The starting point $\boldsymbol{\pi}_1$ is taken from the recruit distribution already stored in `Spatial@UnfishedDist[, , 1, ]`.

## Value

`Spatial` with `Spatial@UnfishedDist` populated as a named `Sim × Area × Age × Year` array.

## See Also

`CalcAsymDist()`, `FitMovement()`, `Spatial()`

---

`CalcUnfishedNumber` *Calculate Equilibrium Unfished Number-at-Age*

---

## Description

Calculates the equilibrium unfished number-at-age

## Usage

```
CalcUnfishedNumber(OM, SP = FALSE)
```

## Arguments

| | |
|---|---|
| `OM` | An `OM()` object |
| `SP` | Logical. Account for `SpawnTimeFrac`? Accounts for spawning timing within a time step to calculate the number-at-age at the time of spawning |

## Details

Equilibrium N-at-Age is calculated from `R0` (which may vary over time) but does NOT account for expected recruitment from the stock-recruit relationship.

Any changes in `R0` due to time-varying biology (Fecundity-at-Age changes over time) should be accounted for in `R0`.

## Value

A named list of length `nStock()` with each element an array with dimensions Sim, Age, and Year

---

`CalcUnfishedSurvival` *Calculate Unfished Survival*

---

## Description

Calculate Unfished Survival

## Usage

```
CalcUnfishedSurvival(
  OM,
  SP = FALSE,
  Years = NULL,
  silent = FALSE,
  Extend = TRUE
)
```

### Arguments

| | |
|---|---|
| OM | An OM() object, a Stock() object, or a list of Stock() objects. |
| SP | Logical. Account for SpawnTimeFrac? Accounts for spawning timing within a time step to calculate the number-at-age at the time of spawning |
| Years | Numeric vector of years to calculate |
| silent | Logical. Print messages? |
| Extend | Logical. Extend the array to include all Sims and Years? |

### Value

A list of length nStock() with an array with the unfished survival for each Stocks

### Examples

```
## Not run:
CalcUnfishedSurvival(OM)

## End(Not run)
```

---

## CALsimp *Simplifies the CAL slot of data object*

---

### Description

A function that condenses the number of catch-at-length bins in a data object

### Usage

```
CALsimp(Data, nbins = 10, simno = 1)
```

### Arguments

| | |
|---|---|
| Data | An object of class 'Data'. |
| nbins | Integer. The target number of catch at length bins |
| simno | Integer. An optional argument to specify the simulation number if writing simulated data |

### Author(s)

T. Carruthers

`Can` *Identify management procedures (MPs) based on data availability*

## Description

Diagnostic tools that look up the slot requirements of each MP and compares to the data available in the Data object.

## Usage

```
Can(Data, timelimit = 1, MPs = NA, dev = FALSE, silent = FALSE)

Cant(Data, timelimit = 1, silent = FALSE)

DLMdiag(
  Data,
  command = c("available", "not available", "needed"),
  reps = 5,
  timelimit = 1,
  funcs1 = NA,
  dev = FALSE,
  silent = FALSE
)

Needed(Data, timelimit = 1, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `Data` | A data-limited methods data object (class Data) |
| `timelimit` | The maximum time (seconds) taken for an MP to undertake 5 reps (this filters out methods that are too slow) |
| `MPs` | Optional list of MP names |
| `dev` | Logical. Run in development mode? |
| `silent` | Logical Display messages? |
| `command` | What to calculate? Character. Options = c("available", "not available", "needed") |
| `reps` | The number of replicates for the MP |
| `funcs1` | A character vector of the MP names (optional) |

## Functions

- `Can()`: Identifies MPs that have the correct data, do not produce errors, and run within the time limit.
- `Cant()`: Identifies MPs that don't have sufficient data, lead to errors, or don't run in time along with a list of their data requirements.

- `DLMdiag()`: Internal function called by `Can` and `Cant`
- `Needed()`: Identifies what data are needed to run the MPs that are currently not able to run given a Data object

### See Also

avail Data-legacy

### Examples

```
CanMPs <- Can(MSEtool::Cobia)
CantMPs <- Cant(MSEtool::Cobia)
Needs <- Needed(MSEtool::Cobia)
```

---

## catch_timeseries *Extract or Assign Catch Time Series*

---

### Description

`Interactions()`, `Landings()`, `Discards()`, and `Removals()` extract total interactions (encounters), retained landings, discards, and removals (landings + discards) respectively.

### Usage

```
Interactions(
  object,
  df = TRUE,
  byAge = FALSE,
  byArea = FALSE,
  byFleet = FALSE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

Landings(
  object,
  df = TRUE,
  byAge = FALSE,
  bySize = FALSE,
  byArea = FALSE,
  byFleet = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)
```

```
Landings(x) <- value

Discards(
  object,
  df = TRUE,
  byAge = FALSE,
  bySize = FALSE,
  byArea = FALSE,
  byFleet = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

Discards(x) <- value

Removals(
  object,
  df = TRUE,
  byAge = FALSE,
  bySize = FALSE,
  byArea = FALSE,
  byFleet = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)
```

## Arguments

| | |
|---|---|
| `object` | A hist, mse, obs, or data object. |
| `df` | Logical. Applies to hist and mse objects only. If `FALSE` the raw array slot is returned. If `TRUE` (default) a tidy `data.frame` is returned. Ignored for obs and data objects. |
| `byAge` | Logical. If `TRUE` the data frame retains the `Age` dimension. Otherwise values are summed over ages. Mutually exclusive with `bySize`. Applies to hist and mse objects only. |
| `byArea` | Logical. If `TRUE` the data frame retains the `Area` dimension. Otherwise values are summed over areas. Applies to hist and mse objects only. |
| `byFleet` | Logical. If `TRUE` (default for `Landings()` and `Discards()`) the data frame retains the `Fleet` dimension. Otherwise values are summed over fleets. Applies to hist and mse objects only. When `bySize = TRUE`, fleets are not required to share a size-class grid (see compdata); `byFleet = FALSE` raises an error if a stock's fleets don't all share the exact same classes, rather than silently summing incompatible bins. |

| | |
|---|---|
| Reduce | Logical. If TRUE (default) simulation dimensions are reduced using ReduceDims() before conversion to a data frame. Applies to hist and mse objects only. |
| IncYear | Logical. Passed to ReduceDims(); controls whether the year dimension is retained during reduction. Default FALSE. Applies to hist and mse objects only. |
| Extend | Logical. If TRUE, rows sharing one value across every simulation (e.g. the historical period, or any deterministic quantity – see Reduce/ReduceDims()) are broadcast to all nSim simulations via ExtendSims(), rather than appearing once with Sim = 1. Default FALSE. Applies to hist and mse objects only. |
| silent | Logical. If FALSE and Extend = FALSE, emits a message when the returned data frame has rows sharing one value across every simulation, pointing at Extend = TRUE. Default TRUE. Applies to hist and mse objects only. |
| bySize | Logical. If TRUE the data frame retains the Size (length-bin) dimension using the *AtSize slots. Otherwise values are summed over size classes. Mutually exclusive with byAge. Not available for Interactions(). Applies to hist and mse objects only. |
| x | An obs or data object. |
| value | A catchobs object (when x is obs) or a catchdata object (when x is data) to assign. |

## Details

When called on a hist or mse object, the functions return simulated catch arrays or tidy data frames as described below.

When called on an obs object, Landings()/Discards() always return the corresponding catchobs slot directly (the observation error structure, not simulated values), regardless of any other argument.

When called on a data object, Landings()/Discards() return the corresponding catchdata slot directly (the observed or simulated values) when neither byAge nor bySize is TRUE. When byAge = TRUE or bySize = TRUE, they instead return a tidy data.frame built from the corresponding *AtAge/*AtSize compdata slot, with columns Year, Fleet, Age or Class, Value, and Variable (no Sim, Stock, Period, or MP, since a data object is a single realization); df is not meaningful in that case and is ignored. For *AtSize, fleets are not required to share a size-class grid (see compdata); byFleet = FALSE raises an error if the object's fleets don't all share the exact same classes, the same as for hist/mse objects.

The assignment forms Landings<- and Discards<- replace the corresponding slot of an obs or data object.

When byAge = FALSE or bySize = FALSE for hist or mse objects, the catch data are in units of biomass (N x Weight), where Weight is the fleet-specific weight-at-age schedule from Fleet(). When byAge = TRUE or bySize = TRUE the values are in units of numbers. When byArea = TRUE the values are always in units of numbers.

When applied to an mse object the historical and projection periods are row-bound and labelled via the Period column; historical rows carry MP = "Historical".

### Value

- For obs: the catchobs object stored in the `Landings` or `Discards` slot.
- For data with `byAge = FALSE` and `bySize = FALSE`: the catchdata object stored in the `Landings` or `Discards` slot.
- For data with `byAge = TRUE` or `bySize = TRUE`: a tidy `data.frame` with columns `Year`, `Fleet`, `Age` or `Class`, `Value`, and `Variable`.
- For hist or mse with `df = FALSE`: the raw array slot (`Interactions`, `Landings`, or `Discards`).
- For hist or mse with `df = TRUE`: a tidy `data.frame` with columns `Sim`, `Stock`, `Year`, `Period`, `MP` (MSE only), and optionally `Age`, `Size`, `Area`, and/or `Fleet`, plus `Value`, `Variable`, and `Units`. `Units` is a biomass unit label (e.g. `"kg"`, `"t"`) when the values are in units of biomass, or the stock's `SRR@Units` scaling factor (e.g. `1`, `1000`) when the values are in units of numbers (see Details).
- Assignment forms return `x` with the named slot replaced by `value`.

### Examples

```
Hist <- Simulate(SingleStockOM)
MSE <- Project(Hist, 'CurrentEffort')

# Tidy data frames
Interactions(Hist)
Landings(MSE)
Discards(MSE)
Removals(MSE)

# Retain fleet, age, and area structure
Landings(MSE, byFleet = TRUE, byAge = TRUE, byArea = TRUE)

# Direct slot access for obs and data objects
obs <- Obs(Landings = CatchObs(CV = 0.2))
Landings(obs)
Landings(obs) <- CatchObs(CV = 0.3)
```

---

## `Catchability` *Catchability Constructor and Accessors*

---

### Description

Construct and manipulate a catchability object defining fishing gear or vessel efficiency for a `Fleet()` object.

### Usage

```
Catchability(Efficiency = NULL, qCV = NULL, qInc = NULL, Misc = list())

Catchability(x) <- value

Efficiency(x)

Efficiency(x) <- value

qCV(x)

qCV(x) <- value

qInc(x)

qInc(x) <- value
```

### Arguments

| | |
|---|---|
| `Efficiency` | Numeric, array, or `NULL`. Gear or vessel catchability coefficient (`q`) relating fishing effort to fishing mortality (encounters, see `Interactions()`). Accepted forms:<br>• `NULL` (default): creates an empty catchability object. `PopulateCatchability()` initialises `Efficiency` to 1 across all simulations and years.<br>• Scalar numeric (e.g. `0.01`): constant efficiency applied across all simulations and years.<br>• Numeric array with dimensions `Sim x Year` and named dimnames. The `Sim` dimension may be length 1 (replicated internally) or match `nSim`. The `Year` dimension may be length 1 or span a subset of years; `PopulateCatchability()` extends it to cover all historical and projected years.<br>If `Efficiency` is a fleet object, the `Catchability` slot of that fleet is returned (see Details). |
| `qCV` | Numeric or `NULL`. Coefficient of variation for lognormal stochastic variation in catchability. Applied to projected years only (see Details). Default `NULL`. Included primarily for backwards compatibility; users may find it simpler to supply a fully specified `Efficiency` array directly. |
| `qInc` | Numeric or `NULL`. Annual percentage increase in catchability (e.g. `2` = 2% per year; negative values indicate declining efficiency). Applied to projected years only (see Details). Default `NULL`. Included primarily for backwards compatibility; users may find it simpler to supply a fully specified `Efficiency` array directly. |
| `Misc` | List. Miscellaneous additional inputs. Default `list()`. |
| `x` | A catchability object for accessor and replacement functions, or an indicesobs object for `qInc()` and `qInc<-`. |
| `value` | The replacement value for the corresponding slot. |

## Details

A catchability object defines the efficiency with which a fleet converts fishing effort into fishing mortality (encounters/interactions, see `Interactions()`).

A constant scalar is sufficient for most applications; time-varying or simulation-varying efficiency can be specified via a full `Sim x Year` array.

**Efficiency Array Format:**

When supplied as a scalar, `Efficiency` is wrapped internally into a `1 x 1` array and replicated across all simulations and years during `PopulateCatchability()`, producing constant efficiency. When supplied as an array, the `Sim` dimension may be length 1 or match `nSim`, and the `Year` dimension is extended to cover all historical and projected years if it does not already do so.

When `Efficiency` is `NULL`, `PopulateCatchability()` initialises it to 1 across all simulations and years, equivalent to supplying `Efficiency = 1`.

**Projected-Year Adjustments: `qInc` and `qCV`:**

`qInc` and `qCV` modify efficiency in projected years only and have no effect on historical years.

`qInc` specifies a percentage annual increase compounded over projection years: efficiency in projection time step `t` is scaled by `(1 + qInc / 100)^(t - t_H)`, where `t - t_H` is the time in years since the last historical time step (a fraction of a year for each season of a seasonal model). Negative values model declining gear efficiency over the projection period.

`qCV` introduces lognormal inter-annual variation in efficiency during projection years. Each projected year's efficiency is multiplied by a mean-1 lognormal deviate with the specified coefficient of variation.

Both parameters are included primarily for backwards compatibility. For most applications, supplying a fully specified `Efficiency` array gives more direct control over time-varying catchability across both historical and projected periods.

**Attaching to a Fleet:**

A catchability object can be attached to a `Fleet()` with `Catchability(Fleet) <- MyCatchability` and retrieved with `Catchability(Fleet)`.

## Value

- `Catchability()` returns a catchability object. If `Efficiency` is a fleet object, the `Catchability` slot of that fleet is returned.
- `Catchability<-` returns `x` with the `Catchability` slot replaced by `value`.
- `Efficiency()`, `qCV()`, `qInc()` return the corresponding slot from the catchability object `x`. `qInc()` also accepts an indicesobs object.
- Their replacement forms return `x` with the corresponding slot updated.

## See Also

- catchability for the class definition and slot-level documentation.
- `Fleet()` for the enclosing fleet constructor.

- `Effort()` for the effort object that is scaled by efficiency to produce fishing mortality.
- `PopulateCatchability()` for how efficiency arrays are extended and stochastic adjustments are applied.

Other fleet: `Bioeconomic()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## Examples

```
## ---- Empty Catchability object ----
# An empty Catchability object is created by default when constructing a Fleet.
# PopulateCatchability() initialises Efficiency to 1 across all simulations
# and years when the Efficiency slot is NULL.
cq <- Catchability()
cq

## ---- Constant efficiency (scalar) ----
# A scalar is the most common form. It is treated as constant across all
# simulations and years: equivalent to supplying a full nSim x nYear array
# of that value.
hist_years <- 2000:2024
proj_years <- 2025:2034
nSim       <- 48

pop <- function(cq, ...) {
  Populate(cq, nSim = nSim, HistYears = hist_years, ProjYears = proj_years, ...)
}

cq_scalar <- Catchability(Efficiency = 0.01)
Efficiency(pop(cq_scalar))

## ---- Time-varying efficiency (array) ----
# A Sim x Year array allows efficiency to vary across years and simulations.
# Here efficiency increases linearly from 0.005 to 0.02 over the historical
# period (one row, replicated across simulations internally).
eff_array <- array(
  seq(0.005, 0.02, length.out = length(hist_years)),
  dim      = c(1, length(hist_years)),
  dimnames = list(Sim = 1, Year = hist_years)
)
cq_array <- Catchability(Efficiency = eff_array)
Efficiency(pop(cq_array))

## ---- Projected-year trend with qInc ----
# qInc applies a compounded annual percentage change to efficiency in
# projected years only. Historical efficiency is unaffected.
# Here a 2% annual increase is applied: efficiency in projection year t
# is scaled by (1 + 2/100)^t relative to the terminal historical value.
# Negative values model declining efficiency (e.g. gear deterioration).
cq_inc <- Catchability(Efficiency = 0.01, qInc = 2)
Efficiency(pop(cq_inc))  # efficiency rises in proj_years; hist_years unchanged
```

```
## ---- Projected-year stochasticity with qCV ----
# qCV introduces lognormal inter-annual variation in efficiency during
# projected years only, via a mean-1 lognormal deviate with the specified
# coefficient of variation. Historical efficiency is unaffected.
# For most applications, a fully specified Efficiency array gives more
# direct control than using qInc or qCV. The two can also be combined.
cq_cv <- Catchability(Efficiency = 0.01, qCV = 0.1)
Efficiency(pop(cq_cv, seed = 42))  # stochastic in proj_years; hist_years constant

## ---- Accessor and replacement functions ----
cq <- Catchability(Efficiency = 0.01, qCV = 0.1, qInc = 1)

Efficiency(cq)
qCV(cq)
qInc(cq)

Efficiency(cq) <- 0.02
qInc(cq)       <- -1   # declining efficiency in projection

## ---- Attaching to a Fleet object ----
f <- Fleet(Name = "Trawl")
Catchability(f) <- Catchability(Efficiency = 0.01)
f |> Catchability() |> Efficiency()
```

---

`catchability-class` *The* `catchability` *S4 Class*

---

### Description

Defines gear or vessel efficiency for use in a fleet object. Objects are typically created via the `Catchability()` constructor, which documents all parameters in detail.

### Slots

`Efficiency` `numeric`, array, or `NULL`. Gear efficiency (`q`). See `Catchability()`.

`qCV` `numeric` or `NULL`. Coefficient of variation for stochastic variation in catchability during projected years. See `Catchability()`.

`qInc` `numeric` or `NULL`. Annual percentage increase in catchability during projected years. See `Catchability()`.

`Misc` `list`. Miscellaneous additional inputs. Used internally.

### See Also

- `Catchability()` for the constructor and full parameter documentation.
- fleet for the enclosing fleet object.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

`CatchAndSurveyObs` *Catch-and-Survey Observation Model*

## Description

An obs object representing a moderate data scenario combining reported landings with a fishery-independent survey index. No effort, CPUE, or composition data are simulated.

## Usage

```
CatchAndSurveyObs
```

## Format

An obs object. Populated slots:

- `Landings` (catchobs): reported landings with CV in `[0.10, 0.25]` and multiplicative bias in `[0.80, 1.00]`, capturing moderate reporting uncertainty with a tendency toward under-reporting.
- `Survey` (indicesobs): fishery-independent index assuming biomass-proportional selectivity, with CV in `[0.15, 0.30]` and autocorrelation in `[0.0, 0.2]`.

Empty slots (no data simulated): `Effort`, `Discards`, `CPUE`, `LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, `DiscardsAtSize`.

`CatchAndSurveyObs` represents a stock for which total catch is reported and a survey provides a relative abundance index, but finer-scale data (effort, CPUE, age or length compositions) are unavailable.

## See Also

obs, `Obs()`, `CatchObs()`, `IndicesObs()`, CommercialFleetObs, AgeStructuredObs, LengthStructuredObs, DataRichObs

Other obs-examples: `AgeStructuredObs`, `CommercialFleetObs`, `DataRichObs`, `LengthStructuredObs`

## Examples

```
CatchAndSurveyObs
Landings(CatchAndSurveyObs)
Survey(CatchAndSurveyObs)
```

`CatchData` *CatchData Constructor*

## Description

Construct a catchdata object storing observed landed or discarded catch and associated uncertainty.

## Usage

```
CatchData(
  Name = NULL,
  Value = NULL,
  CV = NULL,
  Units = NULL,
  Ref = NULL,
  RefCV = NULL
)
```

## Arguments

| | |
|---|---|
| `Name` | `character` or `NULL`. Fleet names, length `nFleet`. Default `NULL`. |
| `Value` | `array` or `NULL`. Observed catch values with dimensions `[nYear x nFleet]`. Default `NULL`. |
| `CV` | `array` or `NULL`. Coefficients of variation for the catch observations, matching the dimensions of `Value`. Default `NULL`. |
| `Units` | `character` or `NULL`. Units of catch measurement, one element per fleet: `"Biomass"`, `"Number"`, or a mass or count unit (e.g. `"t"`, `"kg"`, `"n"`). Values are not rescaled, so catch in biomass must be in the OM's `Weight` units. Default `NULL`. See `CheckCatch()` for the consequences of `Landings` and `Discards` having mismatched units. |
| `Ref` | `array` or `NULL`. Reference catch values (e.g., a historical baseline), matching the dimensions of `Value`. Default `NULL`. |
| `RefCV` | `array` or `NULL`. Coefficients of variation for the reference values, matching the dimensions of `Ref`. Default `NULL`. |

## Details

### Usage:

All arguments are optional; an empty object is valid and slots are populated later (e.g., when `Simulate()` generates historical data).

### Attaching to a Data Object:

```
Landings(data) <- CatchData(Value = myArray, CV = myCVArray)
Discards(data) <- CatchData(Value = myArray, CV = myCVArray)
```

### Value

A catchdata object.

### See Also

- catchdata for the class definition.
- data and `Data()` for the enclosing data object.
- `CatchObs()` for the observation error structure that generates catchdata objects during simulation.

Other data: `CompData()`, `EffortData()`, `IndicesData()`, `catchdata-class`, `compdata-class`, `effortdata-class`, `indicesdata-class`

### Examples

```
# Empty object
cd <- CatchData()

# With values
yrs <- 2000:2020
fleets <- c("Trawl", "Longline")
val <- array(runif(length(yrs) * length(fleets)),
             dim = c(length(yrs), length(fleets)),
             dimnames = list(Year = yrs, Fleet = fleets))
cd <- CatchData(Name  = fleets,
                Value = val,
                CV    = array(0.1, dim = dim(val), dimnames = dimnames(val)),
                Units = c("Biomass", "Biomass"))
```

---

## `catchdata-class` *The* `catchdata` *S4 Class*

---

### Description

Stores time series observations of landed or discarded catch and associated uncertainty. Used in the `Landings` and `Discards` slots of a data object. Objects are typically created via `CatchData()`, which documents all parameters in detail.

### Slots

`Name` `character` or `NULL`. Fleet names, length `nFleet`.

`Value` `array` or `NULL`. Observed catch values with dimensions `[nYear x nFleet]`. See `CatchData()`.

`CV` `array` or `NULL`. Coefficients of variation matching the dimensions of `Value`. See `CatchData()`.

`Units` `character` or `NULL`. Units of catch measurement per fleet: `"Biomass"`, `"Number"`, or a mass or count unit (e.g. `"t"`, `"kg"`, `"n"`). See `CatchData()` and `CheckCatch()` for the consequences of `Landings` and `Discards` having mismatched units.

`Ref array` or `NULL`. Reference catch values (e.g., a historical baseline), matching the dimensions of `Value`. See `CatchData()`.

`RefCV array` or `NULL`. Coefficients of variation for the reference values, matching the dimensions of `Ref`. See `CatchData()`.

### See Also

`CatchData()` for the constructor and full parameter documentation. data for the enclosing data object.

Other data: `CatchData()`, `CompData()`, `EffortData()`, `IndicesData()`, `compdata-class`, `effortdata-class`, `indicesdata-class`

---

## `CatchFrac` *Fleet Allocation of TAC for OM Objects*

---

### Description

Access or replace the `CatchFrac` and `FleetAllocation` slots of an om object, which together control how catch is split among fleets: `CatchFrac` as a historical calibration target, `FleetAllocation` as the split applied to a stock/complex-level TAC during projection. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
CatchFrac(x)

CatchFrac(x) <- value

FleetAllocation(x)

FleetAllocation(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a named list of length 0 or `nStock(OM)` (`CatchFrac`) / the number of stock complexes (`FleetAllocation`), each element an `nSim` by `nFleet` matrix (or a single row recycled across sims) with rows summing to 1. `NULL` clears it. |

### Details

**`CatchFrac`:**

Only relevant when there is more than one fleet and a historical `Depletion@Final` target is set for at least one stock. In that case, catchability is calibrated during conditioning so that the modelled catch split across fleets in the final historical year reproduces `CatchFrac`. If left unspecified for a stock, `CatchFrac` is instead derived from relative Effort times Catchability in the final historical year, and used as-is (no calibration target).

`FleetAllocation`:

Controls how a stock/complex-level TAC is split among fleets during projection - both TAC advice returned by an MP and `InterimAdvice`. For each stock/complex, resolved in this order:

1. Use `FleetAllocation` if specified.
2. Fall back to the deprecated `Allocation` slot if specified (retained only so that objects saved before the `FleetAllocation` rename still carry their data; not user-facing, set `FleetAllocation` directly instead).
3. Fall back to `CatchFrac` if specified.
4. Derive from the mean relative removals (landings + discards) over the last five historical years.

## Value

- `CatchFrac()`/`FleetAllocation()` return the value of the corresponding slot.
- `CatchFrac<-()`/`FleetAllocation<-()` return `x` with the corresponding slot updated.

## See Also

- `OM()` for the constructor, om for the class definition.
- `HistoricalWeight()`/`SeasonalAllocation()` for how a periodically-set TAC/Effort is further split across seasons.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
om <- OM(nSim = 10)
FleetAllocation(om) <- list(Stock1 = matrix(c(0.7, 0.3), 10, 2, byrow = TRUE))
FleetAllocation(om)
```

---

`CatchObs` *CatchObs Constructor and Accessors*

---

## Description

Construct a catchobs object defining observation error for landed or discarded catch, or access and replace individual slots.

### Usage

```
CatchObs(
  CV = NULL,
  Bias = NULL,
  Error = NULL,
  Years = NULL,
  Units = NULL,
  Ref = NULL,
  Misc = list()
)
```

### Arguments

| | |
|---|---|
| `CV` | `numeric` or `NULL`. Coefficient of variation of the lognormal catch observation error. Accepted forms:<br>• `NULL` (default): no observation error is generated unless `Error` is supplied directly.<br>• Length-1 scalar: a single CV applied uniformly across all simulations and years when generating the `Error` array internally.<br>• Length-2 vector `c(mean, sd)`: `CV` is itself drawn stochastically — a unique CV per simulation is generated before being used to generate the `Error` array.<br>• Length-`nSim` named array: used directly, one CV per simulation.<br>When conditioning on real data, `CV` is replaced by the simulation-specific standard deviation of historical log-residuals. |
| `Bias` | `numeric` or `NULL`. Multiplicative observation bias. Accepted forms:<br>• `NULL` (default): bias defaults to `1` (unbiased) internally.<br>• Positive scalar: interpreted as the CV of a lognormal distribution; a unique bias is drawn per simulation as `rlnorm(nSim, mconv(1, Bias), sdconv(1, Bias))`.<br>• Length-2 vector `c(lower, upper)` (when `nSim != 2`): treated as uniform bounds; a unique bias per simulation is drawn via `runif(nSim, lower, upper)`.<br>• Length-`nSim` vector or named array: used directly, one bias per simulation.<br>When conditioning on real data, `Bias` is replaced by the mean ratio of simulated to observed catch across historical years, per simulation. |
| `Error` | `numeric` array or `NULL`. Pre-specified lognormal observation error multipliers with dimensions `nSim x nYear` and named dimnames (`Sim`, `Year`). When supplied, `Error` is used directly and `CV` is ignored for error generation. Values should be positive multipliers centred near 1. Default `NULL`, in which case `Error` is generated from `CV` internally. When conditioning on real data, any user-supplied `Error` is replaced. |
| `Years` | `numeric` vector or `NULL`. Calendar years over which to condition the observation error when real catch data are provided. When `NULL` (default), |

all historical years in the Data object are used. Has no effect when no real data are supplied.

Units — character(1) or NULL. Units of catch measurement: "Biomass", "Number", or a mass or count unit (e.g. "t", "kg", "n"). Determines whether simulated catch is aggregated as biomass or numbers when generating data. Values are not rescaled, so catch in biomass is in the OM's Weight units. Default NULL (treated as "Biomass").

Ref — numeric or NULL. Reference catch value. Accepted forms:

- NULL (default): no reference value.
- Positive scalar: interpreted as the CV of a lognormal distribution; a unique reference value is drawn per simulation as rlnorm(nSim, mconv(1, Ref), sdconv(1, Ref)) internally.
- Length-nSim vector or named array: used directly.

Misc — list. Miscellaneous additional objects. Default list().

## Details

### Observation Error Generation:

The error applied to simulated catch is Bias[sim] * Error[sim, year]. The Error array (nSim x nYear) is obtained in one of three ways, in order of precedence:

1. **User-supplied Error**: stored directly; CV is not used.
2. **Stochastic from CV**: if Error is NULL and CV is specified, Internally the model generates a lognormal array where each cell is rlnorm(1, mconv(1, CV[sim]), sdconv(1, CV[sim])).
3. **Conditioned from real data**: Simulate() estimates CV, Bias, and Error from historical residuals; any user-supplied values are replaced.

Bias is expanded internally: a scalar input is treated as the CV of a lognormal from which one bias per simulation is drawn; a length-nSim input is used directly.

Ref is expanded internally using the same convention as Bias.

### Conditioning on Real Data:

When real catch data are provided (via OM@Data), Simulate() conditions the observation model: CV and Bias are estimated from the historical residuals between simulated and observed catch, and Error is populated to cover both historical and projection years. Any user-supplied CV, Bias, and Error values are replaced.

### Attaching to an Obs Object:

```
Landings(obs) <- CatchObs(CV = 0.2)
Discards(obs) <- CatchObs(CV = 0.3, Bias = 1.1)
```

## Value

- CatchObs() returns a catchobs object.
- Landings() and Discards(), when called on an obs or data object, return the corresponding catchobs or catchdata slot directly (no data frame conversion).
- Landings<- and Discards<- return x with the slot replaced.

### See Also

- catchobs for the class definition.
- `Obs()` for the enclosing observation model constructor.
- `Simulate()` for conditioning the observation model on real data.

Other obs: `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

### Examples

```
# Uniform 20% CV - same CV used for all simulations
co <- CatchObs(CV = 0.2)

# Stochastic CV - unique CV per simulation drawn from lognormal(mean=0.2, sd=0.05)
co <- CatchObs(CV = c(0.2, 0.05))

# Bias as a CV - unique bias per simulation drawn from lognormal(mean=1, sd=0.1)
co <- CatchObs(CV = 0.2, Bias = 0.1)

# Pre-specified error array - bypasses CV entirely
err <- array(rlnorm(48 * 20), dim = c(48, 20),
             dimnames = list(Sim = 1:48, Year = 2001:2020))
co <- CatchObs(Error = err)

# Attach to an obs object
obs <- Obs(Landings = CatchObs(CV = 0.2),
           Discards = CatchObs(CV = 0.3, Bias = 0.05))
```

---

## `catchobs-class` *The* `catchobs` *S4 Class*

---

### Description

Defines the observation error structure for landed or discarded catch data. Used in the `Landings` and `Discards` slots of an obs object. Objects are typically created via `CatchObs()`, which documents all parameters in detail.

### Slots

`CV numeric` or `NULL`. Coefficient of variation of catch observation error. See `CatchObs()`.

`Error numeric` array or `NULL`. Realised lognormal observation error multipliers (`nSim x nYear`). Populated internally during conditioning or simulation; see `CatchObs()`.

`Bias numeric` or `NULL`. Multiplicative observation bias. See `CatchObs()`.

`Years numeric` or `NULL`. Calendar years over which the observation error is conditioned. See `CatchObs()`.

`Units` `character` or `NULL`. Units of catch: `"Biomass"`, `"Number"`, or a mass or count unit (e.g. `"t"`, `"kg"`, `"n"`). See `CatchObs()`.

`Ref` `numeric` array or `NULL`. Reference catch values, one per simulation. See `CatchObs()`.

`Misc` `list`. Miscellaneous additional objects.

### See Also

- `CatchObs()` for the constructor and full parameter documentation.
- obs for the enclosing observation model object.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

---

## `CheckCatch` *Check catch data availability and units*

---

### Description

Warns if none of the catch slots in `slot_names` (by default `Landings` and `Discards`) contain data, or if `Landings` and `Discards` are not in the same units as each other. Missing `Discards` alone is not a problem: TACs are then set from `Landings`.

### Usage

```
CheckCatch(Data, incUnits = TRUE, slot_names = c("Landings", "Discards"))
```

### Arguments

| | |
|---|---|
| `Data` | A data object. |
| `incUnits` | Logical. If `TRUE` (default), an additional warning is issued when `Landings` and `Discards` units do not match each other (only checked when both are included in `slot_names`). |
| `slot_names` | Slots to check. Defaults are `Landings` and `Discards` |

### Details

If no catch data are available, and a finite TAC value cannot be produced, the MP will likely fall back to setting effort to last historical effort - ie same as `CurrentEffort()` (the default for empty `Advice()` objects).

If `Landings` and `Discards` are not in the same units (e.g. one in `Biomass` and the other in `Number`, or one in `"t"` and the other in `"kg"`), summing them to produce a removals TAC is not meaningful. The unit-mismatch check is only performed when both `'Landings'` and `'Discards'` are included in `slot_names`.

Within a projection, warnings are issued only for the first simulation on the MP's first call (`Data@Misc$FirstAdvice`), so they appear once per MP.

### Value

`NULL` invisibly, called for its side-effect warnings.

---

`CheckDuplicate` *Check for duplicated MPs names*

---

### Description

Custom MPs cannot have the same names of MPs in MSEtool and related packages

### Usage

```
CheckDuplicate(MPs)
```

### Arguments

| | |
|---|---|
| `MPs` | Character vector of MP names |

### Value

An error if duplicated MP names, otherwise nothing

---

`CheckMPs` *Check that specified MPs are valid and will run on MSEtool::SimulatedData*

---

### Description

Check that specified MPs are valid and will run on MSEtool::SimulatedData

### Usage

```
CheckMPs(MPs = NA, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `MPs` | Character vector of MP names |
| `silent` | Logical. Report messages? |

### Value

MP names

checkMSE *Utility functions for MSE objects*

## Description

Utility functions for MSE objects

## Usage

```
checkMSE(MSEobj)

addMPs(MSEobjs)

joinMSE(MSEobjs = NULL)

joinHist(Hist_List)

updateMSE(MSEobj, save.name = NULL)
```

## Arguments

| | |
|---|---|
| MSEobj | A MSE object |
| MSEobjs | A list of MSE objects |
| Hist_List | A list of objects of class Hist |
| save.name | Character string. Optional file name to save the updated MSE object to disk. |

## Value

An object of class MSE

A new object of class Hist

## Functions

- checkMSE(): Check that an MSE object includes all slots in the latest version of DLMtool
- addMPs(): Adds additional MPs to an MSE object by combining multiple MSE objects that have identical historical OM values but different MPs.
- joinMSE(): Joins two or more MSE objects together across simulations. MSE objects must have identical number of historical years, and projection years.
- joinHist(): Join objects of class Hist. Does not join slot OM
- updateMSE(): Updates an existing MSE object (class MSE) from a previous version of the MSEtool to include slots new to the latest version. Also works with Stock, Fleet, Obs, Imp, and Data objects. The new slots will be empty, but avoids the 'slot doesn't exist' error that sometimes occurs. Returns an object of class matching class(MSEobj)

### Author(s)

A. Hordyk

### See Also

`addMMPs()`

joinData

---

`CheckOM` *Check OM object is complete*

---

### Description

For an om object, reports which `Stock`, `Fleet`, `Obs`, and `Imp` components are missing or incomplete, as a nested list matching the `[[stock/complex]][[fleet]]` structure of `OM@Fleet`/`OM@Obs`/`OM@Imp`. `Stock`/`Fleet` list required slots that `PopulateOM()` cannot fill on its own; `Obs`/`Imp` have no such required slots (every unset slot legitimately means "no observation"/"perfect implementation").

### Usage

```
CheckOM(OM, msg = TRUE, stop_if_missing = TRUE)
```

### Arguments

| | |
|---|---|
| `OM` | An object of class `OM` |
| `msg` | Logical. Display messages? |
| `stop_if_missing` | Logical. Stop with error is values are missing and there is no default? |

### Value

For an om `OM`: a list with elements `MissingStock`, `MissingFleet`, `MissingObs`, `MissingImp` (each `NULL` if nothing is missing).

### Examples

```
CheckOM(SingleStockOM)
```

---

`CheckPackage` *Check that a required package is installed*

---

## Description

Checks if a required package and (if applicable) package version are installed and if not, provides an error with installation instructions

## Usage

```
CheckPackage(pkg, version = NULL, pkg.path = NULL)
```

## Arguments

| | |
|---|---|
| `pkg` | Character string with the name of the package |
| `version` | Optional. Character string specifying required version number |
| `pkg.path` | Optional. Character string specifying the install command. See `Examples` |

## Examples

```
# Already installed
CheckPackage("MSEtool")

## Not run:
# Not installed
CheckPackage("MadeUp")

# Needs updating
CheckPackage("MSEtool", "99")

# Update and specify installation path
CheckPackage("MSEtool", "99", "pak::pgk_install('blue-matter/MSEtool')")

## End(Not run)
```

---

`Choose` *Manually map parameters for the historical period of operating model*

---

## Description

Interactive plots to specify trends and variability in fishing effort, fleet selectivity, and natural mortality for the operating model.

### Usage

```
ChooseEffort(Fleet, Years = NULL)

ChooseM(OM, type = c("age", "length"), x = NULL, y = NULL)

ChooseSelect(Fleet, Stock, FstYr = NULL, SelYears = NULL)
```

### Arguments

| | |
|---|---|
| `Fleet` | A fleet object. |
| `Years` | An optional vector of years. Should be nyears long. |
| `OM` | An object of class 'OM' |
| `type` | A character string - is M to be mapped by 'age' or 'length'? |
| `x` | Optional vector for x-axis |
| `y` | Optional vector for y-axis |
| `Stock` | Optional Stock object. If provided, average length-at-maturity is included on plot for reference. |
| `FstYr` | Optional value for first historical year. If empty, user must specify the year in console. |
| `SelYears` | Optional vector of values for each year where selectivity pattern changed. If empty, user must specify the years in console (comma separated). |

### Details

| | |
|---|---|
| `ChooseEffort` | Interactive plot which allows users to specify the relative trajectory and variability in the historica |
| `ChooseM` | Interactive plot which allows users to specify M by age or size class |
| `ChooseSelect` | Input the first historical year, and all years where selectivity pattern changed (separated by comm |

### Value

`ChooseEffort` and `ChooseSelect` return a Fleet object while `ChooseM` returns an OM object.

### Author(s)

A. Hordyk

`CollapseSeasons` *Collapse a Seasonal Operating Model to an Annual Operating Model*

## Description

[**Experimental**]

## Usage

```
CollapseSeasons(OM, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `OM` | A seasonal `OM()` object (`OM@Seasons > 1`). Multi-area (`nArea > 1`) OMs are not yet supported. |
| `silent` | `logical(1)`. If `TRUE`, suppresses informational console messages. Defaults to `FALSE`. |

## Details

Converts an Operating Model with sub-annual seasonal resolution (`OM@Seasons > 1`) into an annual Operating Model (`Seasons = 1`) that approximates the same annual-scale fishery dynamics.

## Value

An updated `OM()` object with `Seasons = 1` and re-derived biology/fishery arrays at annual resolution.

### Method:

Seasons are included in the `Year` dimension (as fractional-year timesteps) and the `Age` dimension (as fractional-age classes) of every populated array. So "collapsing seasons" means re-deriving both dimensions at annual resolution, following a cohort's actual path through the `Seasons` sub-annual steps of a calendar year (age `a`, `a + 1/Seasons`, ..., `a + (Seasons-1)/Seasons`):

- **Natural mortality** and **fecundity** are summed
- **Length-** and **weight-at-age**, and **maturity-at-age** are read directly off the populated seasonal array at the exact integer-age, first-season-of-year cell
- **Seasonal** `R0` is summed to annual `R0`
- **Fleet dynamics** (effort, catchability, selectivity, retention, discard mortality, weight-at-age-in-the-catch) follow the same F-weighted-average reconstruction used by `CombineFleets()`, but weighting across a cohort's season-steps within a year instead of across fleets. In a year where every season has zero F, the at-age curves weight the season-steps by catchability, or equally if catchability is also zero.

- **Observation and implementation models** (`OM@Obs`, `OM@Imp`) keep the first season of each year from their `Year`-indexed arrays (e.g. errors, sample sizes, compliance) and the integer ages from their `Age`-indexed arrays

This is an approximation, not an exact transformation: the population dynamics apply mortality and fishing simultaneously within each season (`exp(-(M+F))` per timestep), so an annual model cannot exactly reproduce `Seasons` sequential simultaneous-mortality events with a single annual one. The approximation is expected to be close whenever seasonal heterogeneity in `F` and `M` is mild.

### See Also

`CombineFleets()` for the analogous fleet-aggregation function.

---

## `CombineFleets` *Combine Multiple Fleets into a Single Fleet*

---

### Description

Combines several fleets into a new aggregated fleet within an Operating Model, preserving the overall fishery dynamics that would result from the disaggregated fleets.

### Usage

```
CombineFleets(OM, FleetList = NULL, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `OM` | An `OM()` object. |
| `FleetList` | A named list of character vectors. Each element names the fleets to merge, and the element's name becomes the name of the new combined fleet. Multiple entries combine multiple sets of fleets simultaneously. Defaults to `NULL`, which combines every fleet in `OM` into one, named `"Combined"`. |
| `silent` | `logical(1)`. If `TRUE`, suppresses informational console messages. Defaults to `FALSE`. |

### Value

An updated `OM()` object in which each set of fleets named in `FleetList` has been replaced by a single aggregated fleet across all stocks.

For each entry in `FleetList`, the first named fleet is replaced in-place by the combined fleet and the remaining fleets are dropped. Combination preserves aggregate fishing mortality, selectivity, retention, discard mortality, and weight-at-age.

**Combination equations:**
Let $F_f^{apical}$ be the apical fishing mortality for fleet $f$, and $s_f(a)$, $r_f(a)$, $d_f(a)$ be selectivity, retention, and discard mortality at age $a$.

**F-at-age (interaction)**

$$F_{combined}(a) = \sum_f F_f^{apical} \cdot s_f(a)$$

**Apical F and effort**

$$F_{combined}^{apical} = \max_a F_{combined}(a)$$

the peak of the summed age curve. $q$ for the combined fleet is taken from the first fleet, and effort is recovered as $F_{combined}^{apical}/q$.

Because effort has no stock dimension (one effort series drives every stock a fleet interacts with) this reconstruction can only define effort for the first stock processed. For every other stock, effort is reused from that first stock's reconstruction, and the difference is instead absorbed into that stock's own $q$, so $q_{combined,stock} = F_{combined,stock}^{apical}/\text{Effort}_{ref}$ still reproduces that stock's true apical F exactly.

**Selectivity**

$$s_{combined}(a) = F_{combined}(a) \,/\, \max_a F_{combined}(a)$$

**Retention**

$$r_{combined}(a) = \frac{\sum_f F_f^{apical} \cdot s_f(a) \cdot r_f(a)}{F_{combined}(a)}$$

**Discard mortality** (age-specific F-weighted average on the instantaneous-rate scale)

$$m_{combined}(a) = 1 - \exp\left(-\frac{\sum_f F_{combined,f}(a) \cdot (-\log(1 - d_f(a)))}{F_{combined}(a)}\right)$$

**WeightFleetSelected** (age-specific interaction-F-weighted average)

$$W_{combined}^{sel}(a) = \frac{\sum_f F_{combined,f}(a) \cdot W_f^{sel}(a)}{F_{combined}(a)}$$

**WeightFleetRetained** (age-specific retained-F-weighted average)

$$W_{combined}^{ret}(a) = \frac{\sum_f F_{combined,f}^{retain}(a) \cdot W_f^{ret}(a)}{F_{combined}^{retain}(a)}$$

**Years with zero F**

In a year and simulation where every combined fleet has zero F, the selectivity, retention, discard mortality, and weight-at-age curves above use each fleet's catchability $q_f$ in place of $F_f^{apical}$, or equal weights if every $q_f$ is also zero.

**Fleet settings:**

Weights $w_f$ are each fleet's share of the summed apical F in each simulation and historical year (equal shares where the total is zero).

- `Effort@Targeting` and `Effort@StockTargetingLambda` are the $w_f$-weighted means of the fleets' values.
- `Effort@Distribution` is the $w_f$-weighted mean of the fleets' fixed distributions, and is derived dynamically unless every fleet has one.
- `Effort@Units` are the first fleet's units, since effort is recovered from its catchability.

- `Closure`: an area is open to the combined fleet where any of its fleets is open.
- Projected catchability (from `qInc`/`qCV`) is $q_{combined}(y) = q_{combined}(y_c) \sum_f w_f(y_c)\, q_f(y)/q_f(y_c)$, where $y_c$ is the last historical year.
- Fleets must share `Effort@Mode`, and bag-limit settings (`TripsScalar`, `AnglerPerTrip`, `Theta`) cannot be combined; either is an error.

The combined fleet uses the first fleet's `Obs()` and `Imp()` objects.

**Data:**

For each stock's `Data()` object:

- `Landings`, `Discards`, and `Effort` values are summed across the combined fleets in each year, and are `NA` in years with no data for any of them. `CV` is $\sqrt{\sum_f (CV_f \cdot x_f)^2}/\sum_f x_f$, and `Ref`/`RefCV` are combined in the same way. Catch units must match; effort recorded in different units is dropped for the combined fleet.
- `CPUE` and `Survey` indices named after combined fleets stay in their slot, and each gets its own `Obs()` entry that carries the fleet's index observation settings and, unless the index has its own selectivity, the fleet's selectivity-at-age from before combining. An index keeps its fleet's name, with a `" CPUE"` or `" Survey"` suffix where that name is taken by a fleet or another index.
- `LandingsAtAge`/`DiscardsAtAge` counts are summed across the combined fleets. `LandingsAtSize`/`DiscardsA` counts are summed when every fleet shares the same size-class bins (within floating-point tolerance), and dropped otherwise. Counts are `NA` in years with no sample for any of the fleets.

Catch, effort, and composition data held by only some fleets of a group are dropped for the combined fleet, since the combined total is unknown. Each drop is recorded as an assumption in the stock's `Data` object (see `Log()`).

**Allocations:**

`FleetAllocation()` and `CatchFrac()` shares are summed across the combined fleets. For each complex with a `SeasonalAllocation()`, the combined fleet's seasonal shares are the `FleetAllocation`-weighted mean of its fleets' shares. `HistoricalWeight()` is combined the same way for complexes where `SeasonalAllocation` is derived. If `FleetAllocation` is not set, the fleets' values must be identical. `EffortAllocation()` is reset, and derived again in `Simulate()`.

**Interim advice:**

`InterimAdvice()` rows for the combined fleets are replaced by one row per timestep for the new fleet. `Fleet = NA` rows are unchanged. Within each calendar year x Complex x Type, every fleet in a group must have one row for each of the same timesteps, or no rows so that a `Fleet = NA` total covers the combined fleet.

- `"TAC"` rows must share `TACType` and `TACUnit`. `Mean` is the sum of the fleets' means and `CV` is $\sqrt{\sum_f (CV_f \cdot Mean_f)^2}/\sum_f Mean_f$, the CV of a sum of independent draws. `Max` is the sum of the fleets' bounds, where a fleet with `CV = NA`/`0` or `Mean = 0` is bounded by its `Mean`, and is `NA` if any other fleet has no bound.
- Relative `"Effort"` rows (`EffType = "Rel"`) must have the same `Mean`, `CV` and `Max` for every fleet, and are kept as a single row.

- Absolute `"Effort"` rows cannot be combined, since effort units depend on each fleet's catchability.

Rows that cannot be combined are an error. In that case, set `InterimAdvice(OM) <- NULL` before combining and supply rows for the new fleet afterwards.

---

`CombineMMP` *Create a blank MP recommendations object (class Rec) of the right dimensions*

---

### Description

Create a blank MP recommendations object (class Rec) of the right dimensions

### Usage

```
CombineMMP(temp, nareas)
```

### Arguments

| | |
|---|---|
| `temp` | A list of nsim simulations. |
| `nareas` | The number of areas. |

### Author(s)

T. Carruthers

---

`CombineMSE` *Combine Multiple* `mse` *Objects Across Simulations*

---

### Description

Combines a list of mse objects into a single `MSE` object, by concatenating every simulation-indexed array along the `Sim` dimension.

### Usage

```
CombineMSE(MSE_List, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `MSE_List` | A list of mse objects. |
| `silent` | Logical. Suppress the summary message. Default `FALSE`. |

### Details

All elements of `MSE_List` must share the same MPs (same names, same order), the same stock names, and the same historical/projection years.

### Value

A single mse object spanning all simulations in `MSE_List`.

### See Also

`CombineOMs()`, `CombineFleets()`

---

`CombineOMs` *Combine Multiple Operating Models into One*

---

### Description

Combine multiple `om` objects into a single unified operating model. Each element of `OM_List` represents a distinct stock or stock complex (e.g., sex-structured populations). The resulting object spans the full union of historical and projection years across all inputs.

### Usage

```
CombineOMs(
  OM_List,
  Name = "Combined OM",
  FillEffort = list(nYears = 3, SD = 0.1, Mean = NULL, Values = NULL),
  FillEfficiency = list(nYears = 3, SD = 0.1, Mean = NULL, Values = NULL),
  StandardizeEffort = TRUE,
  record_assumption = TRUE,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `OM_List` | Named list of `om` objects. All elements must share the same `nSim`, `Seasons`, and fleet names in identical order. See `FleetNames()`. |
| `Name` | `character(1)`. Name assigned to the combined OM. Default `"Combined OM"` |
| `FillEffort` | List controlling effort forward-filling. See `Details`. |
| `FillEfficiency` | List controlling efficiency forward-filling. See `Details`. |
| `StandardizeEffort` | `logical(1)`. If `TRUE` (default), calls `StandardizeEffort()` after combining, which equalises effort across stocks for each fleet and back-calculates stock targeting weights. |
| `record_assumption` | `logical(1)`. Passed to `StandardizeEffort()`. Record the effort-standardization as an `"assumption"` in `OM@Log`? Default `TRUE`. Ignored if `StandardizeEffort = FALSE`. |
| `silent` | `logical(1)`. If `TRUE`, suppresses informational messages. Default `FALSE`. |

## Details

### Processing steps:

1. Validate that all OMs share `nSim`, `Seasons`, and fleet names.
2. Standardize each OM via `PopulateOM()`.
3. Construct unified historical and projection year ranges: historical years are the union across all OMs; projection years are those strictly greater than the maximum historical year.
4. Extend every stock and fleet object to the unified year range. Missing early years are back-filled; missing later historical years are forward-filled using `FillEffort` and `FillEfficiency`.
5. Concatenate stocks and fleets.
6. Optionally standardize effort across stocks, which populates `@StockTargeting`

Correlated projection recruitment deviations are not generated here – they are generated automatically (and lazily) the next time the combined `OM` is populated, e.g. via `PopulateOM()` or `Simulate()`. See `GenMultiStockRecDevs()` and `OM@Control$CorrelatedRecDevs`.

### Fill Controls:

Both `FillEffort` and `FillEfficiency` accept:

- `nYears` *(integer)*: Number of trailing historical years used to compute a reference mean
- `SD` *(numeric)*: Lognormal standard deviation applied during stochastic filling
- `Mean` *(numeric)*: Optional override (length 1 or `nSim`)
- `Values` *(array)*: Optional `nSim x n_fill_years` matrix used directly

If `Values` is supplied, all other parameters are ignored.

### Metadata:

The following slots are copied verbatim from `OM_List[[1]]`: `Agency`, `Author`, `Email`, `Region`, `Latitude`, `Longitude`, `Sponsor`, `nSim`, `Seasons`, `DataLag`, `Interval`, `nReps`, `pStar`, `maxF`, `Seed`, `Control`. Ensure the first OM contains the desired global configuration.

## Value

An `om` object with:

- `@Stock`: concatenation of all stocks across input OMs, in list order.
- `@Fleet`: concatenation of all fleet lists across input OMs.
- `@nYear`: length of the unified historical year range.
- `@pYear`: number of projection years strictly after the latest historical year.
- `@CurrentYear`: the latest historical year across all input OMs.
- Administrative metadata (`Agency`, `Author`, `nSim`, `Seed`, etc.) copied from `OM_List[[1]]`.

## See Also

`GenMultiStockRecDevs()`

### Examples

```
## Not run:
combined <- CombineOMs(
  OM_List = list(StockA = om_a, StockB = om_b),
  Name = "Combined OM",
  FillEffort = list(nYears = 5, SD = 0.05),
  FillEfficiency = list(nYears = 5, SD = 0.05)
)

## End(Not run)
```

---

## CommercialFleetObs *Commercial-Fleet Observation Model*

---

### Description

An obs object representing a fleet-monitored data scenario in which landings, discards, effort, and a commercial CPUE index are all observed, but no fishery-independent survey or composition data are available.

### Usage

```
CommercialFleetObs
```

### Format

An obs object. Populated slots:

- `Landings` (catchobs): reported landings with CV in `[0.05, 0.15]` and a positive bias in `[1.00, 1.15]`, reflecting some misreporting.
- `Discards` (catchobs): estimated discards with CV in `[0.20, 0.40]` and bias in `[0.80, 1.20]`, capturing the greater uncertainty in at-sea discard estimation.
- `Effort` (effortobs): vessel-days from logbooks with CV in `[0.02, 0.08]` and near-unbiased reporting `[0.95, 1.05]`.
- `CPUE` (indicesobs): standardised commercial CPUE with CV in `[0.15, 0.25]` and autocorrelation in `[0.0, 0.3]`.

Empty slots (no data simulated): `Survey`, `LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, `DiscardsAtSize`.

### See Also

obs, `Obs()`, `CatchObs()`, `EffortObs()`, `IndicesObs()`, CatchAndSurveyObs, AgeStructuredObs, LengthStructuredObs, DataRichObs

Other obs-examples: `AgeStructuredObs`, `CatchAndSurveyObs`, `DataRichObs`, `LengthStructuredObs`

### Examples

```
CommercialFleetObs
Effort(CommercialFleetObs)
CPUE(CommercialFleetObs)
```

---

`CompareBAM` *Compare BAM and OM Output*

---

### Description

Compares key population time series between BAM output and a simulated operating model (OM), reporting the mean absolute relative error (MARE) for recruits, total numbers, total biomass, and landings and discards by fleet.

### Usage

```
CompareBAM(
  Stock,
  OM = NULL,
  plot = FALSE,
  thresh = 1,
  save_plots = TRUE,
  outdir = "figures/diagnostics/BAM",
  width = NULL,
  height = NULL
)
```

### Arguments

| | |
|---|---|
| `Stock` | Character string matching a stock in `bamExtras`, a list of BAM output objects containing elements `rdat` and `dat`, or an object of class `BAMdata` as returned by `GetBAMOutput()`. |
| `OM` | Optional OM or `hist` object. If `NULL` (default), one is constructed internally via `ImportBAM()`. |
| `plot` | Logical. Plot the timeseries for the `OM` and `BAM` output? If `plot=FALSE` (default) the plots will only be printed if `MARE>thresh` in some years. |
| `thresh` | Numeric. MARE threshold (as a percentage) above which a comparison is flagged, printed, and plotted. Default `1%`. |
| `save_plots` | Logical. If `TRUE` (default), diagnostic plots are written as PNG files to `file.path(outdir, Stock)`, independently of whether they are also printed to the active device (see `plot`). |
| `outdir` | Character. Base directory for saved plots. Defaults to `"figures/diagnostics/BAM"`. |

`width`, `height` Numeric. Width/height (inches) passed to `ggplot2::ggsave()`. If `NULL` (default), each plot is sized automatically from its number of facet panels: `Recruits`, `Number`, and `Biomass` (single-panel) use 6 x 4; `Landings`/`Discards`, which facet by fleet, scale up accordingly. Set either to a number to use that fixed size for every saved plot instead.

### Details

Series with MARE exceeding `thresh` are plotted automatically.

### Value

Invisibly returns a named list with elements `Stock`, `Recruits`, `Number`, `Biomass`, `Landings`, and `Discards`. Each of these (other than `Stock`) is a list with elements `df` (long-format data.frame of OM and BAM values by year), `MARE` (absolute relative error by year), and `plot` (the `ggplot` diagnostic plot, if generated - see `plot`, `thresh`, and `save_plots`).

### See Also

`ImportBAM()`, `GetBAMOutput()`

---

## `CompareiSCAM` *Compare iSCAM and OM Output*

---

### Description

Compares key population time series between an iSCAM MPD fit and a simulated operating model, reporting the mean absolute relative error (MARE) for recruits, spawning biomass, and landings. Follows the same shape as `CompareBAM()`/`CompareSS()`, sharing the same underlying engine.

### Usage

```
CompareiSCAM(
  iSCAMdir,
  OM = NULL,
  sim = 1,
  plot = FALSE,
  thresh = 1,
  save_plots = FALSE,
  outdir = "figures/diagnostics/iSCAM",
  width = NULL,
  height = NULL,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| iSCAMdir | Character string giving the path to a directory containing iSCAM output files, or a list already returned by load.iscam.files(). |
| OM | Optional om or hist object. If NULL (default), one is constructed internally via ImportiSCAM(). |
| sim | Integer. Simulation index used to select the OM simulation row. Default 1 (iSCAM's MPD fit is a single point estimate, so there is only ever one iSCAM-side series to compare against). |
| plot | Logical. Plot the timeseries for the OM and iSCAM output? If FALSE (default) plots are only printed if MARE exceeds thresh. |
| thresh | Numeric. MARE threshold (percentage) above which a comparison is flagged and plotted. Default 1. |
| save_plots | Logical. If TRUE, diagnostic plots are written as PNG files to file.path(outdir, Stock). Default FALSE. |
| outdir | Character. Base directory for saved plots. Defaults to "figures/diagnostics/iSCAM". |
| width, height | Numeric. Passed to ggplot2::ggsave(); NULL (default) sizes automatically. |
| silent | Logical; suppress informational output. Default FALSE. |

### Value

Invisibly returns a named list with elements Stock, Recruits, Biomass, and Landings, each (other than Stock) a list with elements df, MARE, and plot – the same shape as CompareBAM(). Note: unlike CompareBAM()/CompareSS(), there is no Number/Discards comparison, since ImportiSCAM() only imports a single aggregate fleet and iSCAM's MPD output does not separately report discards.

### See Also

ImportiSCAM(), CompareBAM(), CompareSS()

---

## CompareSS *Compare SS3 and OM fishery dynamics*

---

### Description

Generates plots of timeseries of total Number, total Biomass, Landings, and Discards for each stock/fleet for users to visually compare the fishery dynamics from the two models.

### Usage

```
CompareSS(
  SSDir,
  Hist,
  sim = 1,
  plot = FALSE,
  thresh = 1,
  save_plots = FALSE,
  outdir = "figures/diagnostics/SS3",
  width = NULL,
  height = NULL,
  silent = FALSE,
  ...
)

CompareSS_Number(SSDir, Hist, sim = 1, silent = FALSE, ...)

CompareSS_Biomass(SSDir, Hist, sim = 1, silent = FALSE, ...)

CompareSS_Landings(SSDir, Hist, sim = 1, silent = FALSE, ...)

CompareSS_Discards(SSDir, Hist, sim = 1, silent = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `SSDir` | Character string giving the path to a directory containing SS3 output files, or a list returned by `ImportSSReport()`. |
| `Hist` | A `hist-class` or `om-class` object. If an `om-class` object is supplied to `CompareSS()`, `Simulate()` is called automatically before comparison. |
| `sim` | Integer. Simulation index used to select both the SS3 report element and the OM simulation row. Default `1`. Only used where `ImportSSReport()` is applied to multiple SS3 output directories. |
| `plot` | Logical. Plot the timeseries for the `OM` and `SS3` output? If `plot = FALSE` (default) the plots are only printed if MARE exceeds `thresh` in some years – see `CompareBAM()` for the same convention. |
| `thresh` | Numeric. Mean absolute relative error (MARE, as a percentage) threshold above which a series is flagged and plotted. Default `1`. |
| `save_plots` | Logical. If `TRUE`, diagnostic plots are written as PNG files to `file.path(outdir, Stock)`. Default `FALSE`. |
| `outdir` | Character. Base directory for saved plots. Defaults to `"figures/diagnostics/SS3"`. |
| `width, height` | Numeric. Width/height (inches) passed to `ggplot2::ggsave()`. If `NULL` (default), sized automatically from the number of facet panels – see `CompareBAM()`. |
| `silent` | Logical. Passed to `ImportSSReport()` and `Simulate()` to suppress console output. Default `FALSE`. |
| `...` | Additional arguments forwarded to `ImportSSReport()`. |

### Details

The fishery dynamics generated by `Simulate()` should match those from the `SS3` output. If there are significant differences, this indicates either a problem with the OM import process or a mismatch in the population dynamics models.

### Value

- `CompareSS()` invisibly returns a named list with elements `Stock`, `Recruits`, `Number`, `Biomass`, `Landings`, and `Discards`, each (other than `Stock`) a list with elements `df`, `MARE`, and `plot` – the same shape as `CompareBAM()`.
- `CompareSS_Number()` and `CompareSS_Biomass()` each print a `ggplot2::ggplot()` object and return the combined `data.frame` of OM and SS3 values invisibly, with columns `Year`, `Value`, `Stock`, and `Model`.
- `CompareSS_Landings()` and `CompareSS_Discards()` each print a `ggplot2::ggplot()` object and return the combined `data.frame` invisibly, with columns `Year`, `Fleet`, `Value`, and `Model`. Values are in kg (SS3 `retain(B)` / `dead(B)` timeseries converted from metric tons; OM from `Landings()` / `Discards()`).

### See Also

`ImportSSReport()`, `Simulate()`, `Number()`, `Biomass()`, `Landings()`, `Discards()`, `CompareBAM()`

---

## `CompData` *CompData Constructor*

---

### Description

Construct a compdata object storing age or size composition observations of catch samples.

### Usage

```
CompData(
  Name = NULL,
  Value = NULL,
  Classes = NULL,
  Units = NULL,
  Log = list(),
  Misc = list()
)
```

### Arguments

| | |
|---|---|
| `Name` | `character` or `NULL`. Fleet names, length `nFleet`. Default `NULL`. |
| `Value` | `array` or `NULL`. Composition counts with dimensions `[nYear x nFleet x nClass]`. Values represent counts scaled to sum to `SampleSize` within each year and fleet — see `CompObs()` for the simulation model. Default `NULL`. |

| | |
|---|---|
| `Classes` | `numeric`, `list`, or `NULL`. For age compositions, a numeric vector of class values shared by every fleet, length matching `dim(Value)[3]`. For size compositions, a `list` of numeric vectors, one per fleet (named to match `Name`), since fleets are not required to share a size-class grid – see compdata. Default `NULL`. |
| `Units` | `character` or `NULL`. Units of the class variable (e.g., `"years"`, `"cm"`, `"mm"`). Default `NULL`. |
| `Log` | `list`. Named list for diagnostic and audit logging. Default `list()`. |
| `Misc` | `list`. Named list for additional metadata. Default `list()`. |

## Details

### Usage:

All arguments are optional; an empty object is valid and slots are populated later (e.g., when `Simulate()` generates historical data).

### Value Dimensions and Units:

`Value` stores counts, compositions are normalised to sum to `SampleSize` within each year-fleet combination. The simulation model that generates `Value` is defined in `CompObs()`.

### Attaching to a Data Object:

```
LandingsAtAge(data)  <- CompData(Value = myArray, Classes = ages)
DiscardsAtAge(data)  <- CompData(Value = myArray, Classes = ages)

# Size compositions: Classes is a list, one vector per fleet, since
# fleets need not share a size-class grid.
LandingsAtSize(data) <- CompData(Value = myPaddedArray,
                                 Classes = list(Trawl = bins1, Longline = bins2))
DiscardsAtSize(data) <- CompData(Value = myPaddedArray,
                                 Classes = list(Trawl = bins1, Longline = bins2))
```

## Value

A compdata object.

## See Also

- compdata for the class definition.
- data and `Data()` for the enclosing data object.
- `CompObs()` for the observation error structure that generates compdata objects during simulation.

Other data: `CatchData()`, `EffortData()`, `IndicesData()`, `catchdata-class`, `compdata-class`, `effortdata-class`, `indicesdata-class`

### Examples

```
# Empty object
cd <- CompData()

# With values
yrs    <- 2000:2020
ages   <- 1:10
fleets <- c("Trawl", "Longline")
val <- array(0L,
             dim = c(length(yrs), length(fleets), length(ages)),
             dimnames = list(Year = yrs, Fleet = fleets, Age = ages))
cd <- CompData(Name    = fleets,
               Value   = val,
               Classes = ages,
               Units   = "years")
```

---

## compdata-class *The* compdata *S4 Class*

---

### Description

Stores age or size composition observations of catch samples. Used in the `LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, and `DiscardsAtSize` slots of a data object. Objects are typically created via `CompData()`, which documents all parameters in detail.

### Slots

`Name character` or `NULL`. Fleet names, length `nFleet`.

`Value array` or `NULL`. Composition counts with dimensions `[nYear x nFleet x nClass]`. For age compositions the third dimension indexes a class grid shared by every fleet (see `Classes`). For size compositions the third dimension is a *fleet-relative position*, not a shared bin: fleets are not required to use the same size-class grid, so `nClass` is the largest class count among the object's fleets, and a fleet with fewer classes has unused (`NA`) cells at the end of its slice. See `CompData()`.

`Classes numeric`, `list`, or `NULL`. For age compositions, a numeric vector of class values (ages in years) shared by every fleet, length matching `dim(Value)[3]`. For size compositions, a `list` of numeric vectors, one per fleet (named to match `Name`), each giving that fleet's own lower bounds of size bins in the appropriate unit; bin `k` of fleet `fl` spans `[Classes[[fl]][k], Classes[[fl]][k+1])`, and `length(Classes[[fl]])` may be less than `dim(Value)[3]` (the unused tail of that fleet's slice in `Value`). See `CompData()`.

`Units character` or `NULL`. Units of the class variable (e.g., `"years"`, `"cm"`, `"mm"`), shared by every fleet. See `CompData()`.

`Log list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

`Misc list`. Named list for additional composition-level metadata.

### See Also

- `CompData()` for the constructor and full parameter documentation.
- data for the enclosing data object.

Other data: `CatchData()`, `CompData()`, `EffortData()`, `IndicesData()`, `catchdata-class`, `effortdata-class`, `indicesdata-class`

---

`ComplexOM` *Stock Complex Example Operating Model*

---

### Description

An example om object representing two stocks managed jointly as a single complex, each exploited by one fleet. The `Complexes` slot groups both stocks so that data and management advice are aggregated across the complex. Intended to illustrate stock complex construction. Pairings of stocks, fleets, and observation models are for illustration only and should not be taken as ecologically meaningful. See also SingleStockOM, TwoFleetOM, and MultiStockOM for structural variants.

### Usage

```
ComplexOM
```

### Format

An om object with the following slots populated:

- `Name`: `"Stock Complex - Single Fleet"`.
- `nSim`: `8`. Number of stochastic simulations.
- `nYear`: `20`. Number of historical years.
- `pYear`: `30`. Number of projection years.
- `Stock`: a length-2 list of `[[AlbacoreExStock, ButterfishExStock]]`.
- `Fleet`: a length-2 list, each containing a length-1 list of AsympExFleet. Structure is `[[stock]][[fleet]]`. Both stocks are fished by the same fleet type.
- `Obs`: a length-1 list containing a length-1 list of DataRichObs. Structure is `[[complex]][[fleet]]`. A single observation model applies to the whole complex, since data are aggregated across both stocks before being observed.
- `Complexes`: `list(StockComplex = 1:2)`. Both stocks are assigned to a single complex named `"StockComplex"`, so data and management advice are aggregated across stocks 1 and 2.
- `Imp`: a length-1 list containing a length-1 list of FullComplianceImp. Structure is `[[complex]][[fleet]]`. Perfect TAC/ effort implementation and full immediate size-regulation compliance.

## Details

### Stocks:

Two biologically contrasting stocks are included:

- AlbacoreExStock: long-lived, slow-growing, low natural mortality.
- ButterfishExStock: short-lived, fast-growing, high natural mortality.

See AlbacoreExStock and ButterfishExStock for full parameter details.

### Stock Complex:

Setting `Complexes = list(StockComplex = 1:2)` instructs the operating model to treat both stocks as a single management unit. Catch and index data are aggregated across the complex when generating management advice, while population dynamics are simulated independently for each stock.

### Fleet:

AsympExFleet is applied to both stocks, representing a single fleet type with asymptotic length-based selectivity. See AsympExFleet for full parameter details.

### Observation Model:

Because both stocks are aggregated into a single complex, the `Obs` slot is indexed by complex and fleet rather than by individual stock. A single DataRichObs object applies to the complex as a whole: landings, discards, effort, a commercial CPUE index, a spawning-biomass survey, and annual age- and length-composition samples from landed and discarded fish. A fully-monitored scenario is appropriate here since joint management of a stock complex relies on comprehensive, aggregated data across its member stocks. See DataRichObs for full parameter details.

### Implementation Error:

FullComplianceImp models perfect implementation: `TAC@Mean = 1/ SD = 0` and `Effort@Mean = 1/SD = 0` mean realised TAC/effort always equal advice exactly, and `Size@Compliance = 1` means any size-regulation advice (a change to `Advice@Retention/Advice@Selectivity`) is adopted immediately by the whole fleet. See `Imp()` for details, and OverageImp, UnderageImp, PartialSizeComplianceImp for contrasting examples.

## See Also

SingleStockOM, TwoFleetOM, MultiStockOM for structural variants. AlbacoreExStock, ButterfishExStock, AsympExFleet, DataRichObs, FullComplianceImp for the component objects. `OM()`, om, `runMSE()`, `PopulateOM()`, `ExampleMPs()`

Other om: `CatchFrac()`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
ComplexOM
ComplexOM@Complexes
nStock(ComplexOM)
```

```
## Not run:
Hist <- runMSE(ComplexOM, MPs = ExampleMPs())

## End(Not run)
```

---

`CompObs` *CompObs Constructor and Accessors*

---

## Description

Construct a compobs object defining the observation error structure for age or length composition data, or access and replace composition observation slots of an obs object.

## Usage

```
CompObs(
  SampleSize = NULL,
  ESS = NULL,
  Theta = NULL,
  Years = NULL,
  Shift = NULL,
  Misc = list()
)
```

## Arguments

| | |
|---|---|
| `SampleSize` | `numeric` or `NULL`. Nominal sample size (number of fish aged or measured per year), used as the count argument of the final multinomial draw. `NULL` (default) uses `ESS` as the sample size; if both are `NULL`, no composition data are generated for this data type. Accepted input forms:<br>• Scalar: constant across all simulations and years.<br>• Length-2 vector `c(lower, upper)`: bounds of a Uniform distribution from which one value per simulation is drawn, constant across years.<br>• Named matrix with dimensions `Sim` and/or `Year`: specifying values at change-point years, expanded to `[nSim x nYear]` by `PopulateCompObs()`. See Specifying Biological and Fleet Schedules for the full matrix input convention. |
| `ESS` | `numeric` or `NULL`. Effective sample size, which scales the Dirichlet-Multinomial concentration vector (see Details). Controls stochastic variability in the composition draw independently of `SampleSize`. Accepted input forms are identical to those for `SampleSize` above. `NULL` (default) uses `SampleSize` as the effective sample size. Typically `ESS <= SampleSize`; values of `ESS < SampleSize` produce overdispersion relative to a pure multinomial with `SampleSize` draws. |

`Theta` `numeric` or `NULL`. Dirichlet-Multinomial dispersion parameter in `(0, 1]`. `Theta = 1` (default) recovers the standard multinomial. Values less than 1 produce overdispersed compositions, with `Theta -> 0` giving maximum overdispersion. Accepted input forms are identical to those for `SampleSize` above.

`Years` `numeric` vector or `NULL`. Calendar years of observed composition data to use during conditioning. Only relevant when real composition data are supplied in `OM@Data`; has no effect during simulation-only runs. `NULL` (default) uses all available historical years during conditioning.

`Shift` `numeric` or `NULL`. Systematic per-bin offset on the log-concentration scale applied to the Dirichlet concentration vector before drawing, capturing directional bias between observed and OM-predicted compositions. The concentration for bin $b$ is scaled as $\alpha'_b = \text{ESS} \times q_b \times \exp(\text{Shift}_b)$, so `Shift = 0` leaves a bin unchanged, positive values inflate its concentration, and negative values deflate it. Because the transformation is log-multiplicative, `Shift` cannot produce invalid (negative) concentrations regardless of magnitude. `NULL` (default) applies no shift. Accepted input forms:

- Scalar: constant offset across all simulations, years, and bins.
- Vector of length `nBin`: bin-specific offset, constant across simulations and years.
- Named matrix with any subset of `Sim`, `Year`, and `Bin` dimensions, with change-point years: expanded to `[nSim x nYear x nBin]` by `PopulateCompObs()`.
- Full `[nSim x nYear x nBin]` array. In conditioning mode, `Shift` is populated internally from the mean per-bin log-concentration residual across historical years and should not be set by the user.

`Misc` `list`. Reserved for internal use. Default `list()`.

## Details

### Composition generation model:

Let $\mathbf{q}$ be the OM-predicted composition vector (length `nBin`, sums to 1) for a given simulation, year, and fleet. The observation error model proceeds as follows:

**Step 1 — Base concentration vector**

$$\alpha = \text{ESS} \times \mathbf{q}$$

**Step 2 — Apply log-concentration offset (if `Shift` non-`NULL`)**

$$\alpha'_b = \alpha_b \times \exp(\text{Shift}[\text{sim}, \text{year}, b])$$

When `Shift` is `NULL`, $\alpha' = \alpha$.

**Step 3 — Dirichlet draw**

$$\mathbf{p}^* \sim \text{Dirichlet}(\alpha'/\text{Theta})$$

where `Theta` $\in (0, 1]$ controls overdispersion. `Theta = 1` recovers the standard Dirichlet with concentration $\alpha'$. Smaller values of `Theta` shrink the total concentration and increase variance.

**Step 4 — Multinomial draw**

$$\text{obs} \sim \text{Multinomial}(\text{SampleSize}, \mathbf{p}^*)$$

**Conditioning mode:**

When real composition data are supplied via `OM@Data`, `Simulate()` compares observed compositions with OM-predicted compositions across the historical years specified in `Years`, and populates `Shift` (mean per-bin log-concentration residual) as well as estimating an appropriate `ESS` and `Theta`. Users should not set these slots manually when conditioning on real data.

Note: when observed compositions differ substantially from OM-predicted compositions (e.g. a length distribution shifted far left or right, or with markedly different spread), the resulting `Shift` values will be large. This is treated as observation error rather than model mis-specification — an assumption that should be evaluated carefully. See the technical manual for a full discussion.

**Attaching to an Obs object:**

```
obs <- Obs()
LandingsAtAge(obs)  <- CompObs(SampleSize = 200, ESS = 50)
LandingsAtSize(obs) <- CompObs(SampleSize = 150, ESS = 40, Theta = 0.5)
```

## Value

- `CompObs()` returns a compobs object.
- `LandingsAtAge()`, `DiscardsAtAge()`, `LandingsAtSize()`, `DiscardsAtSize()` return the corresponding compobs slot from an obs object `x`.
- Their replacement forms return `x` with the corresponding slot updated and the object revalidated.

## See Also

- compobs for the class definition and slot descriptions.
- `PopulateCompObs()` for the population function.
- `Simulate()` for conditioning the observation model on real data.
- `Obs()` for the enclosing observation model constructor.

Other obs: `CatchObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

### Examples

```
# Default multinomial draw: 200 fish sampled, ESS = 50
co <- CompObs(SampleSize = 200, ESS = 50)

# With Dirichlet-Multinomial overdispersion (Theta = 0.5)
co_dm <- CompObs(SampleSize = 200, ESS = 50, Theta = 0.5)

# Stochastic sample size drawn from Uniform(150, 250) across simulations
co_stoch <- CompObs(SampleSize = c(150, 250), ESS = 50)

# Bin-specific shift (e.g. 7 age classes): inflate older ages
co_shift <- CompObs(SampleSize = 200, ESS = 50, Shift = c(-2, -1, 0, 0, 1, 2, 3))

# Attach to an obs object
obs <- Obs(
  LandingsAtAge  = CompObs(SampleSize = 200, ESS = 50),
  LandingsAtSize = CompObs(SampleSize = 150, ESS = 40, Theta = 0.5)
)
```

---

## compobs-class *The* compobs *S4 Class*

---

### Description

Defines the observation error structure for age or length composition data (catch-at-age, catch-at-length). Used in the `LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, and `DiscardsAtSize` slots of an obs object. Objects are typically created via `CompObs()`, which documents all parameters and accepted input forms in detail.

### Details

**Composition generation model:**

During simulation, `PopulateCompObs()` applies the following steps:

1. `Shift` **non-**`NULL`: user-specified (or conditioned) per-bin log-concentration offsets are exponentiated and applied to the OM-predicted concentration vector:

$$\alpha'_b = \text{ESS} \times q_b \times \exp(\text{Shift}_b)$$

2. `Shift` `NULL` (default): pure Dirichlet-Multinomial draw from the OM-predicted composition with concentration scaled by `ESS` and dispersion `Theta`:

$$\alpha' = \text{ESS} \times \mathbf{q}$$

**Note on large compositional shifts:**

When observed compositions differ substantially from OM-predicted compositions (e.g. a length distribution shifted far left or right, or with markedly different spread), conditioning will produce large values in `Shift`. These are treated as observation error rather than model mis-specification — an assumption that should be evaluated carefully. See the technical manual for a full discussion of this assumption and its implications for simulation performance.

### Slots

`SampleSize numeric` or `NULL`. Nominal sample size (number of fish aged or measured) applied when drawing the final multinomial sample. Accepted input forms are documented in `CompObs()`. After population by `PopulateCompObs()`, stored as a named `[nSim x nYear]` array. `NULL` (default) suppresses composition data generation for this data type; `EmptyObject()` returns `TRUE` when `SampleSize` is `NULL`.

`ESS numeric` or `NULL`. Effective sample size, which scales the Dirichlet-Multinomial concentration vector and therefore controls stochastic variability in the composition draw independently of `SampleSize`. Accepted input forms are the same as for `SampleSize`; see `CompObs()`. After population, stored as a named `[nSim x nYear]` array. `NULL` (default) uses `SampleSize` as the effective sample size.

`Theta numeric` or `NULL`. Dirichlet-Multinomial dispersion parameter in `(0, 1]`. Values less than 1 produce overdispersed compositions relative to a standard multinomial; `Theta = 1` (default) recovers the standard multinomial. Accepted input forms are the same as for `SampleSize`; see `CompObs()`. After population, stored as a named `[nSim x nYear]` array.

`Years numeric` vector or `NULL`. Calendar years of observed composition data to be used during the conditioning step. `NULL` (default) uses all available historical years during conditioning.

`Shift numeric` or `NULL`. Systematic per-bin offset on the log-concentration scale applied to the Dirichlet concentration vector before drawing, capturing directional bias between observed and OM-predicted compositions. After population by `PopulateCompObs()`, stored as a named `[nSim x nYear x nBin]` array. `NULL` (default) applies no shift. In conditioning mode, populated internally from the mean per-bin log-concentration residual across historical years.

`Misc list`. Reserved for internal use.

### See Also

- `CompObs()` for the constructor and full parameter documentation.
- `PopulateCompObs()` for the population function.
- `Simulate()` for conditioning the observation model on real data.
- obs for the enclosing observation model object.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

---

## `ConstrainTAC` *Constrain a TAC Change Between Management Cycles*

---

### Description

Applies minimum/maximum fractional change limits to a proposed TAC modifier, then clips the resulting TAC to an absolute range.

### Usage

```
ConstrainTAC(PrevTAC, Mod, DeltaDown, DeltaUp, TACRange)
```

### Arguments

| | |
|---|---|
| `PrevTAC` | Positive number. Previous TAC (see `LastTAC()`). |
| `Mod` | Positive number. Proposed modifier (e.g. `1.2` = 20% increase). |
| `DeltaDown`, `DeltaUp` | Numeric vector, length 2 (`c(min, max)`). Minimum and maximum allowed fractional change, downward and upward respectively. |
| `TACRange` | Numeric vector, length 2 (`c(min, max)`). Absolute bounds on the TAC. |

### Value

A numeric TAC.

### See Also

`IndexRate()`, `Advice()`

### Examples

```
# Proposed 50% increase, capped at the 20% max upward change
ConstrainTAC(PrevTAC = 100, Mod = 1.5,
             DeltaDown = c(0.01, 0.3), DeltaUp = c(0.01, 0.2),
             TACRange = c(0, 1000))

# Proposed 5% decrease, within the minimum-change band: TAC held constant
ConstrainTAC(PrevTAC = 100, Mod = 0.95,
             DeltaDown = c(0.1, 0.3), DeltaUp = c(0.1, 0.2),
             TACRange = c(0, 1000))

# Allowed change would exceed the absolute TAC range: clipped to the max
ConstrainTAC(PrevTAC = 100, Mod = 1.15,
             DeltaDown = c(0.01, 0.3), DeltaUp = c(0.01, 0.2),
             TACRange = c(0, 110))
```

---

## `Converge` *Check Convergence*

---

### Description

Have I undertaken enough simulations (nsim)? Has my MSE converged on stable (reliable) performance metrics?

### Usage

```
Converge(
  MSEobj,
  PMs = c("Yield", "P10", "AAVY"),
  maxMP = 15,
  thresh = 0.5,
  ref.it = 20,
  inc.leg = FALSE,
  all.its = FALSE,
  nrow = NULL,
  ncol = NULL,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An MSE object of class `'MSE'` |
| `PMs` | A character vector of names of the PM methods or a list of the PM methods |
| `maxMP` | Maximum number of MPs to include in a single plot |
| `thresh` | The convergence threshold. Maximum root mean square deviation over the last `ref.it` iterations |
| `ref.it` | The number of iterations to calculate the convergence statistics. For example, a value of 20 means convergence diagnostics are calculated over last 20 simulations |
| `inc.leg` | Logical. Should the legend be displayed? |
| `all.its` | Logical. Plot all iterations? Otherwise only (nsim-ref.it):nsim |
| `nrow` | Numeric. Optional. Number of rows |
| `ncol` | Numeric. Optional. Number of columns |
| `silent` | Hide the messages printed in console? |

### Details

Performance metrics are plotted against the number of simulations. Convergence diagnostics are calculated over the last `ref.it` (default = 20) iterations. The convergence diagnostics are:

1. Is the order of the MPs stable over the last `ref.it` iterations?
2. Is the average difference in performance statistic over the last `ref.it` iterations < `thresh`?

By default three commonly used performance metrics are used:

1. Average Yield Relative to Reference Yield
2. Probability Spawning Biomass is above 0.1BMSY
3. Probability Average Annual Variability in Yield is < 20 per cent

Additional or alternative performance metrics objects can be supplied. Advanced users can develop their own performance metrics.

### Value

A table of convergence results for each MP

### Author(s)

A. Hordyk

### Examples

```
## Not run:
MSE <- runMSE()
Converge(MSE)

## End(Not run)
```

---

`Convert` *Convert legacy S4 objects to new S4 object classes*

---

### Description

Converts legacy openMSE S4 objects to their equivalent new S4 class representations. Dispatches to the appropriate conversion function based on the class of `x`.

### Usage

```
Convert(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | A legacy S4 object of class OM-legacy, MOM-legacy, Stock-legacy, Fleet-legacy, Obs-legacy, Imp-legacy, or Data-legacy. |
| `...` | Additional named arguments passed to the underlying conversion function (`ConvertOM()`, `ConvertMOM()`, `ConvertStock()`, `ConvertFleet()`, `ConvertObs()`, `ConvertImp()`, or `ConvertData()`). |

### Details

Not all slots from Obs-legacy objects exist in the new obs objects. Slots without a direct equivalent are silently dropped.

Legacy `Data` objects do not contain discard data. It is assumed that `Data@Cat` represents landings only, with no discards. If `Data@Cat` represents total removals, the resulting data object must be updated manually after conversion.

### Value

A new S4 object of the corresponding updated class:

- OM-legacy → om
- MOM-legacy → om
- Stock-legacy → stock
- Fleet-legacy → fleet
- Obs-legacy → obs
- Imp-legacy → imp
- Data-legacy → data

### See Also

`ConvertOM()`, `ConvertMOM()`, `ConvertStock()`, `ConvertFleet()`, `ConvertObs()`, `ConvertImp()`, `ConvertData()`

### Examples

```
# Stock
stock <- Convert(Herring)

# Fleet
fleet <- Convert(Generic_Fleet)

# Obs
obs <- Convert(Generic_Obs)

# Imp
imp <- Convert(Perfect_Imp)

# OM
om <- Convert(testOM)

# MOM
om <- Convert(Albacore_TwoFleet)

# Data
data <- Convert(China_rockfish)
```

---

## `ConvertData` *Convert Between Legacy* `Data` *and New* `data` *Classes*

---

### Description

Converts a legacy Data-legacy object to the current data, or a data object (or list of data objects, one per simulation) to a legacy Data-legacy object. Direction is determined automatically from the class of `x`.

### Usage

```
ConvertData(x, Seasons = 1, sim = 1, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | A Data-legacy object, a data object, or a `list` of data objects (one per simulation, to be combined into a single multi-sim legacy `Data` object via `joinData()`). |
| `Seasons` | Integer. Number of seasons per year. Only used when converting legacy `Data` -> `data`. Default `1`. |
| `sim` | Integer. Simulation index to extract from the legacy object. Only used when converting legacy `Data` -> `data`. Default `1`. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

**Legacy `Data` -> `data`:**

Legacy `Data` objects do not distinguish between landings and discards. `Data@Cat` is mapped to `Landings` only; `Discards` will be empty in the resulting data object. If `Data@Cat` represents total removals, the `Discards` slot must be populated manually after conversion.

`Ind`, `SpInd`, `VInd`, and every column of `AddInd` become indices in `data@Survey`. The tracked biomass component (total, spawning, vulnerable) is recorded in `Survey@Misc$IndexType`, and `AddIunits` sets the `Units` of each `AddInd` index (`Ind`, `SpInd`, and `VInd` are `"Biomass"`). `FMSY_M` is converted to an absolute `Reference@FMSY` via `FMSY_M * Mort`. Slots with no structural equivalent anywhere in `data` (`ML`, `Lc`, `Lbar`, `Abun`, `SpAbun`, `BMSY_B0`, `Bref`, `Rec`, `AvC`, `Dt`, `Ref`, `Ref_type`) are stashed under `data@Misc`/`data@Reference@Misc` rather than dropped.

**`data` -> Legacy `Data`:**

A data object holds a single realization (no `Sim` dimension), so the result always has `nsim = 1` - callers of a legacy MP must use `x = 1`. Multi-fleet time series and composition data are aggregated to the single-fleet legacy shape (landings + discards summed across fleets for `Cat`; effort summed across fleets; age/size composition summed across fleets, with size-composition bins rebinned onto a common grid first). When multiple `CPUE`/`Survey` indices are present, one is selected for `Data@Ind` (`Survey` preferred over `CPUE`, then most complete, then longest span). Slots with no equivalent in `data` (see `Data@Misc$UnmappedSlots` on the result) are left `NA`. Every lossy or approximated mapping is recorded in `Data@Log` (see `Log()`).

### Value

- Legacy `Data` -> `data`: a data object populated from simulation `sim` of `x`.
- `data` (or list of `data`) -> legacy `Data`: a Data-legacy object with `nsim` equal to `1` (single `data` object) or `length(x)` (list of `data` objects).

### See Also

`Convert()`, `ConvertOM()`, `ConvertMOM()`, `joinData()`

### Examples

```
## Not run:
Datalegacy <- readRDS("MyLegacyData.rds")
data_new <- ConvertData(Datalegacy, sim = 1)
Data_again <- ConvertData(data_new)

## End(Not run)
```

---

## `ConvertFleet` *Convert a Legacy Fleet Object to the Current* `fleet` *Class*

---

### Description

Converts a legacy Fleet-legacy object to the current fleet by mapping each component to its corresponding new S4 sub-object.

### Usage

```
ConvertFleet(Fleet, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `Fleet` | A Fleet-legacy object to convert. An OM-legacy object with a legacy `Fleet` slot is also accepted, in which case the `Fleet` slot is extracted and converted. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

**Slot Mapping:**

The following legacy slots are mapped to their current equivalents:

| Legacy slot | New location |
|---|---|
| `Name` | `fleet@Name` |
| `EffYears`, `EffLower`, `EffUpper`, `Esd` | `Effort@Effort` (data frame) |
| `qcv`, `qinc` | `Catchability@qCV`, `@qInc` |
| `L5`, `LFS`, `Vmaxlen` | `Selectivity@Pars`, `@Model = DoubleNormal` |
| `LR5`, `LFR`, `Rmaxlen` | `Retention@Pars`, `@Model = DoubleNormalRetention` |

**Effort:**

Historical effort is constructed as a data frame with columns `Year`, `Lower`, `Upper`, and `CV`, passed to `Effort()`. The legacy `Esd` slot contains a two-element vector of lower and upper bounds for a uniform distribution of effort variability. Only `Esd[1]` is currently used as a scalar CV; `Esd[2]` is silently dropped.

**Selectivity:**

`L5`, `LFS`, and `Vmaxlen` map to the `DoubleNormal()` double-normal selectivity-at-length model. `Vmaxlen < 1` produces a dome-shaped selectivity curve. See `SelectivityModels()` for the full set of available selectivity models.

**Retention:**

`LR5`, `LFR`, and `Rmaxlen` map to the `DoubleNormalRetention()` double-normal retention-at-length model. `Rmaxlen < 1` produces a dome-shaped retention curve. See `RetentionModels()` for the full set of available retention models.

**Slots Not Converted:**

The following slots have no equivalent in the legacy Fleet-legacy and retain their default values in the converted object:

- `DiscardMortality`: new functionality; no legacy equivalent.
- `Closure`: new functionality; no legacy equivalent.
- `Targeting`: new functionality; no legacy equivalent.
- `WeightFleetRetained`/`WeightFleetSelected`: new functionality; no legacy equivalent.

## Value

A fleet object.

## See Also

`Convert()`, `ConvertOM()`, `ConvertMOM()`, `ConvertStock()`, `ConvertObs()`, `ConvertImp()`, `Effort()`, `Catchability()`, `Selectivity()`, `Retention()`, `SelectivityModels()`, `RetentionModels()`

## Examples

```
## Not run:
fleet_legacy <- readRDS("MyLegacyFleet.rds")
fleet_new <- ConvertFleet(fleet_legacy)

## End(Not run)
```

---

`ConvertImp` *Convert a Legacy Imp Object to a New imp Class*

---

## Description

Converts a legacy Imp-legacy object to the current imp by mapping implementation error parameters to their corresponding new S4 slots.

## Usage

```
ConvertImp(Imp, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `Imp` | An Imp-legacy object to convert. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

## Value

An imp object with `TAC`, `Effort`, and `Size` implementation error slots populated.

## See Also

`Convert()`, `ConvertOM()`, `ConvertMOM()`, `ConvertObs()`

## Examples

```
## Not run:
Implegacy <- readRDS("MyLegacyImp.rds")
imp_new <- ConvertImp(Implegacy)

## End(Not run)
```

---

`ConvertMOM` *Convert Legacy MOM Object to New om Class*

---

## Description

Converts a legacy MOM-legacy object to the current om, copying attributes, updating year vectors, and optionally populating the object via `PopulateOM()`.

### Usage

```
ConvertMOM(
  MOM,
  Author = "",
  CurrentYear = NULL,
  Seasons = 1,
  Populate = TRUE,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| MOM | A MOM-legacy object to convert. |
| Author | Character. Author of the OM. Default "". |
| CurrentYear | Numeric. Last historical calendar year. If NULL (default), taken from MOM@Fleets[[1]][[1]]@CurrentYr; if that is less than 1000, defaults to the current system year. |
| Seasons | Integer. Number of seasons per year. Default 1. |
| Populate | Logical. If TRUE (default), calls PopulateOM() on the converted object. |
| silent | Logical. If TRUE, suppresses progress messages. Default FALSE. |

### Value

An om object.

### See Also

Convert(), ConvertOM(), ConvertStock(), ConvertFleet(), ConvertObs(), ConvertImp(), PopulateOM()

### Examples

```
## Not run:
MOMlegacy <- readRDS("MyLegacyMOM.rds")
om_new <- ConvertMOM(MOMlegacy)

## End(Not run)
```

---

## ConvertObs *Convert a Legacy Obs Object to the Current* obs *Class*

---

### Description

Converts a legacy Obs-legacy object to the current obs by mapping observation error parameters to their corresponding sub-object slots.

### Usage

```
ConvertObs(Obs, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `Obs` | An Obs-legacy object to convert. An om object with a legacy `Obs` slot is also accepted, in which case the `Obs` slot is extracted and converted. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

#### Slot Mapping:

The following legacy slots are mapped to their current equivalents:

| Legacy slot | New slot |
|---|---|
| `Eobs` | `Effort@CV` |
| `Ebiascv` | `Effort@Bias` |
| `Cobs` | `Landings@CV` and `Discards@CV` |
| `Cbiascv` | `Landings@Bias` and `Discards@Bias` |
| `Iobs` | `CPUE@CV` and `Survey@CV` |

Note that `Discards@CV` and `Discards@Bias` are set to the same values as `Landings` (`Cobs` and `Cbiascv`) because legacy objects do not distinguish between landings and discards observation error.

#### Slots Not Yet Converted:

The following legacy slots do not have a corresponding slot in the current obs and are silently dropped. They will be mapped once the lifehistoryobs and exploitationobs placeholders are fully implemented:

- Life-history bias CVs: `Linfbiascv`, `tObiascv`, `Kbiascv`, `LenMbiascv`, `Mbiascv`
- Selectivity bias CVs: `LFCbiascv`, `LFSbiascv`
- Reference point bias CVs: `FMSY_Mbiascv`, `BMSY_B0biascv`
- Composition data: `CAA_nsamp`, `CAA_ESS`, `CAL_nsamp`, `CAL_ESS` (mapped to `LandingsAtAge` and `LandingsAtSize` once implemented)

### Value

An obs object.

### See Also

`Convert()`, `ConvertOM()`, `ConvertMOM()`, `ConvertImp()` for related legacy conversion functions.

### Examples

```
## Not run:
obs_legacy <- readRDS("MyLegacyObs.rds")
obs_new <- ConvertObs(obs_legacy)

## End(Not run)
```

---

`ConvertOM` *Convert Legacy OM Object to New om Class*

---

### Description

Converts a legacy OM-legacy object to the current om, copying attributes, updating year vectors, and optionally populating the object via `PopulateOM()`.

### Usage

```
ConvertOM(
  OM,
  Author = "",
  CurrentYear = NULL,
  Seasons = 1,
  Populate = TRUE,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `OM` | An OM-legacy object to convert. |
| `Author` | Character. Author of the OM. Default `""`. |
| `CurrentYear` | Numeric. Last historical calendar year. If `NULL` (default), taken from `OM@CurrentYr`; if that is less than 1000, defaults to the current system year. |
| `Seasons` | Integer. Number of seasons per year. Default `1`. |
| `Populate` | Logical. If `TRUE` (default), calls `PopulateOM()` on the converted object. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

Maximum length slots (`Vmaxlen` and `Rmaxlen`) are updated to reflect the maximum length class rather than `Linf` as in legacy objects. This changes the selectivity and retention curves if `Vmaxlen` or `Rmaxlen` are less than 1. As an alternative, users can set the `MeanAtLength` or `MeanAtAge` arrays directly in the `Selectivity()` and `Retention()` objects.

### Value

An om object.

### See Also

`Convert()`, `ConvertMOM()`, `ConvertStock()`, `ConvertFleet()`, `ConvertObs()`, `ConvertImp()`, `PopulateOM()`

### Examples

```
## Not run:
OMlegacy <- readRDS("MyLegacyOM.rds")
om_new <- ConvertOM(OMlegacy)

## End(Not run)
```

---

## `ConvertStock` *Convert a Legacy Stock Object to the Current* `stock` *Class*

---

### Description

Converts a legacy Stock-legacy object to the current stock by mapping each component to its corresponding new S4 sub-object.

### Usage

```
ConvertStock(Stock, Seasons = 1, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `Stock` | A Stock-legacy object to convert. An OM-legacy object with a legacy `Stock` slot is also accepted, in which case the `Stock` slot is extracted and converted. |
| `Seasons` | Numeric. Number of seasons in a year. Default `1`. |
| `silent` | Logical. If `TRUE`, suppresses progress messages. Default `FALSE`. |

### Details

**Slot Mapping:**

The following legacy slots are mapped to their current equivalents:

| Legacy slot | New location |
|---|---|
| `Name`, `Common_Name`, `Species` | `stock@Name`, `@CommonName`, `@Species` |
| `maxage` | `Ages@MaxAge` (via `Ages()`, `MinAge = 0`) |
| `Linf`, `K`, `t0`, `Linfsd`, `Ksd`, `LenCV` | `Length@Pars`, `@CVatAge` |
| `a`, `b` | `Weight@Pars$alpha`, `@Pars$beta` |

| | |
|---|---|
| M, Msd | NaturalMortality@Pars |
| L50, L50_95 | Maturity@Pars, @Model = MaturityAtLength |
| ageM, age95 | Maturity@Pars, @Model = MaturityAtAge (used only when L50 is absent) |
| h | SRR@Pars$h |
| R0 | SRR@R0 |
| Perr | SRR@SD |
| AC | SRR@AC |
| SRrel | SRR@Model (1 → "BevertonHolt", 2 → "Ricker") |
| Size_area_1 | Spatial@RelativeSize |
| Prob_staying | Spatial@ProbStaying |
| Frac_area_1 | Spatial@UnfishedDist |
| D | Depletion@Final |

**Maturity Priority:**

When both length-based (L50, L50_95) and age-based (ageM, age95) maturity parameters are present, the length-based parameters take priority and Model is set to MaturityAtLength. Age-based parameters are used only when L50 is absent or NULL.

**Spatial Structure:**

If all three spatial parameters (Size_area_1, Prob_staying, Frac_area_1) equal 0.5, an empty spatial object is returned. This is treated by the model as a single well-mixed area with no spatial structure.

## Value

A stock object.

## See Also

Convert(), ConvertOM(), ConvertMOM(), ConvertFleet(), ConvertObs(), ConvertImp()

## Examples

```
## Not run:
stock_legacy <- readRDS("MyLegacyStock.rds")
stock_new <- ConvertStock(stock_legacy)

## End(Not run)
```

---

# Cos_thresh_tab *Current default thresholds for COSEWIC satisficing*

---

## Description

Current default thresholds for COSEWIC satisficing

### Usage

```
Cos_thresh_tab(Ptab1)
```

### Arguments

| | |
|---|---|
| `Ptab1` | A COSEWIC performance table made by COSEWIC_tab() |

### Author(s)

T. Carruthers

---

## cparscheck — *Internal function for checking that the* `OM@cpars` *is formatted correctly*

---

### Description

Internal function for checking that the `OM@cpars` is formatted correctly

### Usage

```
cparscheck(cpars)
```

### Arguments

| | |
|---|---|
| `cpars` | a list of model parameters to be sampled (single parameters are a vector `nsim` long, first dimension of matrices and arrays must be `nsim`) |

### Value

either an error and the length of the first dimension of the various cpars list items or passes and returns the number of simulations in `cpars`

### Author(s)

T. Carruthers

`Cplot` *Plot the median biomass and yield relative to last historical year*

## Description

Compare median biomass and yield in first year and last 5 years of projection

## Usage

```
Cplot(
  MSEobj,
  MPs = NA,
  lastYrs = 5,
  point.size = 2,
  lab.size = 4,
  axis.title.size = 12,
  axis.text.size = 10,
  legend.title.size = 12
)
```

## Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `MPs` | Optional vector of MPs to plot |
| `lastYrs` | Numeric. Last number of years to summarize results. |
| `point.size` | Size of the points |
| `lab.size` | Size of labels |
| `axis.title.size` | Axis title size |
| `axis.text.size` | Axis text size |
| `legend.title.size` | Legend title size |

## Examples

```
## Not run:
MSE <- runMSE()
Cplot(MSE)

## End(Not run)
```

`CR2h` *Convert Between Steepness and Compensation Ratio*

## Description

Convert between the Beverton-Holt stock-recruitment steepness (h) and compensation ratio (CR) parameterizations.

## Usage

```
CR2h(CR)

h2CR(h)
```

## Arguments

| | |
|---|---|
| `CR` | Numeric. Compensation ratio (see Details). |
| `h` | Numeric. Steepness value (see Details). Must be in the range (0.2, 1), exclusive. |

## Details

**Steepness (h)** is the fraction of unfished recruitment obtained when the spawning biomass is reduced to 20% of its unfished level. It is bounded between 0.2 (highly depensatory) and 1.0 (density-independent recruitment).

**Compensation ratio (CR)** is the ratio of recruits-per-spawner at the origin of the stock-recruitment curve relative to recruits-per-spawner at unfished equilibrium. Higher values indicate stronger density-dependent compensation. CR is bounded between 1 (no compensation) and infinity.

The relationships between the two parameters are:

$$h = \frac{CR}{CR + 4} \qquad CR = \frac{4h}{1 - h}$$

## Value

Numeric. `CR2h()` returns steepness (h) bounded between 0.2 and 1; `h2CR()` returns the compensation ratio (CR).

## Examples

```
CR2h(5)
h2CR(0.7)
```

---

`Data` *Constructor and Accessor for* `Data`

---

## Description

Creates a new data object, or extracts the `Data` or `PPD` slot from an existing om, hist, or mse class object.

## Usage

```
Data(
  Name = "New Data Object",
  CommonName = NULL,
  Species = NULL,
  Agency = NULL,
  Author = NULL,
  Email = NULL,
  Region = NULL,
  Latitude = NULL,
  Longitude = NULL,
  Years = NULL,
  YearLH = NULL,
  Seasons = 1,
  nArea = 1,
  LifeHistory = NULL,
  Exploitation = NULL,
  Reference = NULL,
  Effort = NULL,
  Landings = NULL,
  Discards = NULL,
  CPUE = NULL,
  Survey = NULL,
  LandingsAtAge = NULL,
  DiscardsAtAge = NULL,
  LandingsAtSize = NULL,
  DiscardsAtSize = NULL,
  Advice = NULL,
  Misc = list()
)

Data(x) <- value
```

## Arguments

| | |
|---|---|
| `Name` | Either a character string naming the new data object, or an existing om, hist, or mse object from which to extract data. Defaults to `"New Data Object"`. |

| | |
|---|---|
| `CommonName` | Optional character string. Common name of the stock. |
| `Species` | Optional character string. Scientific name of the species. |
| `Agency` | Optional character string. Name of the managing agency. |
| `Author` | Optional character string. Name of the data author. |
| `Email` | Optional character string. Contact email for the author. |
| `Region` | Optional character string. Geographic region of the stock. |
| `Latitude` | Optional numeric. Latitude of the stock. |
| `Longitude` | Optional numeric. Longitude of the stock. |
| `Years` | Vector of calendar years covered by the data. **Required** |
| `YearLH` | The last historical calendar year; always a whole integer, even when `Seasons > 1` and `Years` holds sub-annual decimal steps. Separates the historical period from the projection period. Defaults to `floor(max(Years))`. |
| `Seasons` | A positive integer giving the number of seasons per year. Defaults to `1`. |
| `nArea` | A positive integer giving the number of spatial areas. Defaults to `1`. |
| `LifeHistory` | Optional. An object of class lifehistorydata . |
| `Exploitation` | Optional. An object of class exploitationdata. |
| `Reference` | Optional. An object of class referencedata. |
| `Effort` | Optional. An object of class effortdata. |
| `Landings` | Optional. An object of class catchdata for landed catch. |
| `Discards` | Optional. An object of class catchdata for discarded catch. |
| `CPUE` | Optional. An object of class indicesdata for catch-per-unit- effort indices. |
| `Survey` | Optional. An object of class indicesdata for fishery- independent survey indices. |
| `LandingsAtAge` | Optional. An object of class compdata for age composition of landings. |
| `DiscardsAtAge` | Optional. An object of class compdata for age composition of discards. |
| `LandingsAtSize` | Optional. An object of class compdata for size composition of landings. |
| `DiscardsAtSize` | Optional. An object of class compdata for size composition of discards. |
| `Advice` | Optional. An object of class advicedata containing TAC and related advice. |
| `Misc` | A named list for any additional user-defined data. Defaults to `list()`. |
| `x` | An om object, for use with `Data<-`. |
| `value` | A data object, or a list of data objects for multi-stock operating models, to assign to the `Data` slot. |

### Details

When `Name` is an om or hist object, the function returns the corresponding `@Data` slot. When `Name` is an mse object, the `@PPD` slot is returned instead. Otherwise, a new `data` object is constructed from the supplied arguments.

All slot arguments default to `NULL`, in which case an empty sub-object of the appropriate class is initialised automatically:

- `LifeHistory` → lifehistorydata
- `Exploitation` → exploitationdata
- `Reference` → referencedata
- `Effort` → effortdata
- `Landings`, `Discards` → catchdata
- `CPUE`, `Survey` → indicesdata
- `LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, `DiscardsAtSize` → compdata
- `Advice` → advicedata

### Value

A data object, or when `Name` is an mse object, a list of data objects from the `@PPD` slot. `Data<-` returns `x` with the `Data` slot replaced by `value`.

### See Also

data, `LastTAC()`, `LastHistYearInd()`, `ProjectionYear()`

---

## `data-class` *The* `data` *S4 Class*

---

### Description

The `data` class stores observed or simulated fishery data used by an `OM()` object. Data may include life-history information, exploitation patterns, catches, indices, age and size compositions, and management advice.

### Details

Objects should be created with the `Data()` constructor, which initialises all sub-object slots to empty objects of the appropriate class when not supplied.

### Slots

`Name` Optional character string. Name of the data object.

`CommonName` Optional character string. Common name of the stock.

`Species` Optional character string. Scientific name of the species.

`Agency` Optional character string. Name of the managing agency.

`Author` Optional character string. Name(s) of the data author(s).

`Email` Optional character string. Contact email(s) for the author(s).

`Region` Optional character string. Geographic region of the stock.

`Latitude` Optional numeric. Latitude of the stock in decimal degrees.

`Longitude` Optional numeric. Longitude of the stock in decimal degrees.

`Years` Numeric vector of calendar years covered by the data.

`YearLH` Numeric. The last historical calendar year (always a whole integer, even when `Seasons > 1` and `Years` holds sub-annual decimal steps), separating the historical period from the projection period. Defaults to `floor(max(Years))` when not supplied to `Data()`.

`Seasons` Positive integer. Number of seasons per year. Defaults to `1`.

`nArea` Positive integer. Number of spatial areas. Defaults to `1`.

`LifeHistory` An object of class lifehistorydata containing biological parameters such as growth, maturity, and natural mortality.

`Exploitation` An object of class exploitationdata containing selectivity, retention, and discard mortality parameters.

`Reference` An object of class referencedata containing biological reference points such as unfished biomass and MSY-based quantities.

`Effort` An object of class effortdata containing fishing effort time series.

`Landings` An object of class catchdata containing landed catch time series.

`Discards` An object of class catchdata containing discarded catch time series.

`CPUE` An object of class indicesdata containing catch-per-unit-effort indices.

`Survey` An object of class indicesdata containing fishery-independent survey indices.

`LandingsAtAge` An object of class compdata containing age composition of landed catch.

`DiscardsAtAge` An object of class compdata containing age composition of discarded catch.

`LandingsAtSize` An object of class compdata containing size composition of landed catch.

`DiscardsAtSize` An object of class compdata containing size composition of discarded catch.

`Advice` An object of class advicedata containing TAC recommendations and related management advice.

`Misc` A named list for any additional user-defined data. Defaults to `list()`.

`Log` `list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

### See Also

`Data()`

---

`Data-legacy-class` *Class* `'Data'`

---

## Description

An object for storing fishery data for analysis

## Slots

`Name` The name of the Data object. Single value. Character string

`Common_Name` Common name of the species. Character string

`Species` Scientific name of the species. Genus and species name. Character string

`Region` Name of the general geographic region of the fishery. Character string

`LHYear` The last historical year of the simulation (before projection). Single value. Positive integer

`MPrec` The previous recommendation of a management procedure. Vector of length nsim. Positive real numbers

`Units` Units of the catch/absolute abundance estimates. Single value. Character string

`MPeff` The current level of effort. Vector of length nsim. Positive real numbers

`nareas` Number of fishing areas. Vector of length nsim. Non-negative integer

`MaxAge` Maximum age. Vector nsim long. Positive integer

`Mort` Natural mortality rate. Vector nsim long. Positive real numbers

`CV_Mort` Coefficient of variation in natural mortality rate. Vector nsim long. Positive real numbers

`vbLinf` Maximum length. Vector nsim long. Positive real numbers

`CV_vbLinf` Coefficient of variation in maximum length. Vector nsim long. Positive real numbers

`vbK` The von Bertalanffy growth coefficient K. Vector nsim long. Positive real numbers

`CV_vbK` Coefficient of variation in the von Bertalanffy K parameter. Vector nsim long. Positive real numbers

`vbt0` Theoretical age at length zero. Vector nsim long. Non-positive real numbers

`CV_vbt0` Coefficient of variation in age at length zero. Vector nsim long. Positive real numbers

`wla` Weight-Length parameter alpha. Vector nsim long. Positive real numbers

`CV_wla` Coefficient of variation in weight-length parameter a. Vector nsim long. Positive real numbers

`wlb` Weight-Length parameter beta. Vector nsim long. Positive real numbers

`CV_wlb` Coefficient of variation in weight-length parameter b. Vector nsim long. Positive real numbers

`steep` Steepness of stock-recruitment relationship. Vector nsim long. Value in the range of one-fifth to 1

`CV_steep` Coefficient of variation in steepness. Vector nsim long. Positive real numbers

`sigmaR` Recruitment variability. Vector nsim long. Positive real numbers

`CV_sigmaR` Coefficient of variation in recruitment variability. Vector nsim long. Positive real numbers

`L50` Length at 50 percent maturity. Vector nsim long. Positive real numbers

`CV_L50` Coefficient of variation in length at 50 per cent maturity. Vector nsim long. Positive real numbers

`L95` Length at 95 percent maturity. Vector nsim long. Positive real numbers

`LenCV` Coefficient of variation of length-at-age (assumed constant for all age classes). Vector nsim long. Positive real numbers

`LFC` Length at first capture. Vector nsim long. Positive real numbers

`CV_LFC` Coefficient of variation in length at first capture. Vector nsim long. Positive real numbers

`LFS` Shortest length at full selection. Vector nsim long. Positive real numbers

`CV_LFS` Coefficient of variation in length at full selection. Vector nsim long. Positive real numbers

`Vmaxlen` Vulnerability of individuals at asymptotic length. Vector nsim long. Real number between 0 and 1.

`Year` Years that corresponding to catch and relative abundance data. Vector nyears long. Positive integer

`Cat` Total annual catches. Matrix of nsim rows and nyears columns. Non-negative real numbers

`CV_Cat` Coefficient of variation in annual catches. Matrix nsim rows and either 1 or nyear columns. Positive real numbers. Note: built-in MPs use only the first value of `CV_Cat` for all years.

`Effort` Annual fishing effort. Matrix of nsim rows and nyears columns. Non-negative real numbers

`CV_Effort` Coefficient of variation in annual effort. Matrix nsim rows and either 1 or nyear columns. Positive real numbers. Note: built-in MPs use only the first value of `CV_Effort` for all years.

`Ind` Relative total abundance index. Matrix of nsim rows and nyears columns. Non-negative real numbers

`CV_Ind` Coefficient of variation in the relative total abundance index. Matrix nsim rows and either 1 or nyear columns. Positive real numbers. Note: built-in MPs use only the first value of `CV_Ind` for all years

`SpInd` Relative spawning abundance index. Matrix of nsim rows and nyears columns. Non-negative real numbers

`CV_SpInd` Coefficient of variation in the relative spawning abundance index. Matrix nsim rows and either 1 or nyear columns. Positive real numbers.

`VInd` Relative vulnerable abundance index. Matrix of nsim rows and nyears columns. Non-negative real numbers

`CV_VInd` Coefficient of variation in the relative vulnerable abundance index. Matrix nsim rows and either 1 or nyear columns. Positive real numbers.

`AddInd` Optional additional indices. Array of dimensions `nsim`, n additional indices, and `nyears` (length `Year`).

`CV_AddInd` Coefficient of variation for additional indices. Array of same dimensions as `AddInd`

`AddIndV` Vulnerability-at-age schedules for the additional indices. Array with dimensions: `nsim`, n additional indices, `MaxAge+1`.

`AddIunits` Units for the additional indices - biomass (1; default) or numbers (0). Numeric vector length n.ind.

`AddIndType` Index calculated from total stock (1, default), spawning stock (2), or vulnerable stock (3). Numeric vector of length n.ind

`Rec` Recent recruitment strength. Matrix of nsim rows and nyears columns. Non-negative real numbers

`CV_Rec` Log-normal CV for recent recruitment strength. Matrix nsim rows and either 1 or nyear columns. Positive real numbers. Note: built-in MPs use only the first value of `CV_Rec` for all years.

`ML` Mean length time series. Matrix of nsim rows and nyears columns. Non-negative real numbers

`Lc` Modal length of catches. Matrix of nsim rows and nyears columns. Positive real numbers

`Lbar` Mean length of catches over Lc. Matrix of nsim rows and nyears columns. Positive real numbers

`Vuln_CAA` Optional vulnerability-at-age schedule for catch-at-age samples. Used to condition OM for closed-loop simulation testing. Replaces the fleet selectivity schedule in the OM used to generate CAA samples. Matrix with dimensions `nsim` x `MaxAge+1`.

`CAA` Catch at Age data (numbers). Array of dimensions nsim x nyears x MaxAge+1. Non-negative integers

`Vuln_CAL` Optional vulnerability-at-length schedule for catch-at-length samples. Used to condition OM for closed-loop simulation testing. Replaces the fleet selectivity schedule in the OM used to generate CAL samples. Matrix with dimensions `nsim` x `length(CAL_mids)`.

`CAL_bins` The values delimiting the length bins for the catch-at-length data. Vector. Non-negative real numbers

`CAL_mids` The values of the mid-points of the length bins. Optional, calculated from `CAL_bins` if not entered. Vector. Non-negative real numbers.

`CAL` Catch-at-length data. An array with dimensions nsim x nyears x length(CAL_mids). Non-negative integers. By default the CAL data will be the retained lengths (i.e, not including discards). If `OM@control$CAL =="removals"` then the CAL data will include all removals (retained + discards).

`Dep` Stock depletion SSB(current)/SSB(unfished). Vector nsim long. Fraction.

`CV_Dep` Coefficient of variation in current stock depletion. Vector nsim long. Positive real numbers

`Abun` An estimate of absolute current vulnerable abundance. Vector nsim long. Positive real numbers

`CV_Abun` Coefficient of variation in estimate of absolute current stock size. Vector nsim long. Positive real numbers

`SpAbun` An estimate of absolute current spawning stock abundance. Vector nsim long. Positive real numbers

`CV_SpAbun` Coefficient of variation in estimate of absolute spawning current stock size. Vector nsim long. Positive real numbers

`FMSY_M` An assumed ratio of FMSY to M. Vector nsim long. Positive real numbers

`CV_FMSY_M` Coefficient of variation in the ratio in FMSY/M. Vector nsim long. Positive real numbers

`BMSY_B0` The most productive stock size relative to unfished. Vector nsim long. Fraction

`CV_BMSY_B0` Coefficient of variation in the position of the most productive stock size relative to unfished. Vector nsim long. Positive real numbers

`Cref` Reference or target catch level (eg MSY). Vector of length nsim. Positive real numbers

`CV_Cref` Log-normal CV for reference or target catch level. Vector of length nsim. Positive real numbers

`Bref` Reference or target biomass level (eg BMSY). Vector of length nsim. Positive real numbers

`CV_Bref` Log-normal CV for reference or target biomass level. Vector of length nsim. Positive real numbers

`Iref` Reference or target relative abundance index level (eg BMSY / B0). Vector of length nsim. Positive real numbers

`CV_Iref` Log-normalCV for reference or target relative abundance index level. Vector of length nsim. Positive real numbers

`t` The number of years corresponding to AvC and Dt. Single value. Positive integer

`AvC` Average catch over time t. Vector nsim long. Positive real numbers

`CV_AvC` Coefficient of variation in average catches over time t. Vector nsim long. Positive real numbers

`Dt` Depletion over time t SSB(now)/SSB(now-t+1). Vector nsim long. Fraction

`CV_Dt` Coefficient of variation in depletion over time t. Vector nsim long. Positive real numbers

`Ref` A reference management level (eg a catch limit). Single value. Positive real number

`Ref_type` Type of reference management level (eg 2009 catch limit). Single value. Character string

`Log` A record of events. Single value. Character string

`params` A place to store estimated parameters. An object. R list

`PosMPs` The methods that can be applied to these data. Vector. Character strings

`TAC` The calculated catch limits (function TAC). An array with dimensions PosMPs x replicate TAC samples x nsim. Positive real numbers

`Sense` The results of the sensitivity analysis (function Sense). An array with dimensions PosMPs x sensitivity increments. Positive real numbers

`MPs` The methods that were applied to these data. Vector. Character strings

`OM` A table of operating model conditions. R table object of nsim rows. Real numbers

`Obs` A table of observation model conditions. R table object of nsim rows. Real numbers

`Misc` Other information for MPs. An object. R list

### Objects from the Class

Objects can be created by calls of the form `new('Data', stock)`

### Author(s)

T. Carruthers and A. Hordyk

### Examples

```
newdata<-new('Data')
```

---

## Data_xl *Read in Data object from Excel spreadsheet*

---

### Description

A function to read in Data object from an Excel spreadsheet with tabs named following specific convention.

### Usage

```
Data_xl(fname, stkname, fpath = "", saveCSV = FALSE)
```

### Arguments

| | |
|---|---|
| `fname` | Name of the Excel spreadsheet file. Must include file extension. |
| `stkname` | Name of the Stock. |
| `fpath` | Full file path, if file is not in current working directory |
| `saveCSV` | Do you also want to the Data parameters to a CSV file? |

### Details

The Excel spreadsheet must have tabs named with the following convention. For example if `stkname` is 'myFish', the Data parameters are in a tab named 'myFishData'.

### Value

A object of class Data

### Author(s)

A. Hordyk

### Examples

```
## Not run:
OM <- OM_xl(fname='OMTables.xlsx', stkname='myFish')

## End(Not run)
```

---

`Data2csv` *Converts a Data object into a .csv data file*

---

### Description

A function that writes a correctly formatted .csv file from a MSEtool Data object

### Usage

```
Data2csv(Data, file = NULL, simno = 1, overwrite = F, keepNAs = T)
```

### Arguments

| | |
|---|---|
| `Data` | An object of class 'Data'. |
| `file` | Character string. The name of the location and file you wish to create (e.g. "C:/temp/mydata.csv") |
| `simno` | Integer. An optional argument to specify the simulation number if writing simulated data |
| `overwrite` | Boolean. Should existing data files be automatically overwritten. |
| `keepNAs` | Boolean. Should slots with NAs still be written to the data file. |

### Author(s)

T. Carruthers

## DataDir *Directory of the data in the installed package on your computer*

### Description

A way of locating where the package was installed so you can find example data files and code etc.

### Usage

```
DataDir(stock = NA)
```

### Arguments

| | |
|---|---|
| stock | Character string representing the name of a .csv file e.g. 'Snapper', 'Rockfish' |

### Value

The file path to the object

### Author(s)

T. Carruthers

### Examples

```
## Not run:
tilefish_location <- DataDir("Gulf_blue_tilefish")
tilefish_Data <- new("Data", tilefish_location)

## End(Not run)
```

## DataHelpers *Data Helper Functions*

### Description

A collection of helper functions for extracting key values from a `Data` object, including TAC advice, and year indices for historical and projection time steps.

### Usage

```
LastTAC(Data, TACType = c("Removals", "Landings"))

LastHistYearInd(Data)

ProjectionYear(Data)

SeasonOfYear(x, Seasons = NULL, Year = NULL)
```

### Arguments

| | |
|---|---|
| `Data` | An object of class data. |
| `TACType` | Character. Catch returned when no TAC has been set: `'Removals'` (default; landings plus discards) or `'Landings'`. |
| `x` | A data object, or a numeric vector of years. |
| `Seasons` | Integer. Seasons per year. Required when `x` is not a `data` object; ignored otherwise. |
| `Year` | Numeric. The year to query. Defaults to the most recent year in the resolved years vector. |

### Value

- `LastTAC()`: A numeric scalar giving the last TAC, or the most recent annual catch of type `TACType` if no TAC is available.
- `LastHistYearInd()`: An integer giving the index of the last historical year in `Data@Years`.
- `ProjectionYear()`: A positive integer giving the current projection year index.
- `SeasonOfYear()`: A list with elements `Seasons` (the number of seasons) and `SeasonInd` (the season index, `1..Seasons`, of `Year`).

### Functions

- `LastTAC()`: Get the last TAC (Total Allowable Catch) from the advice slot. If no TAC has been set, returns the catch of type `TACType` (summed over fleets, ignoring `NA`) in the most recent year; for seasonal data (`Seasons > 1`), the most recent complete calendar year (see `AnnualData()`).
- `LastHistYearInd()`: Get the index of the last historical year within `Data@Years`. This is the position of the final year that does not exceed `Data@YearLH`, and is typically used to subset or index time series vectors aligned to `Data@Years`.
- `ProjectionYear()`: Get the index of the current projection time step relative to `Data@YearLH`. Returns `1` in the first projection year, incrementing by one each subsequent timestep
- `SeasonOfYear()`: Get the season of a given (or, by default, the current/most recent) year. `x` may be a `data` object (`Seasons` is then taken from `x@Seasons` and `Year` defaults to the most recent element of `x@Years`) or a plain numeric vector of years, in which case `Seasons` must be supplied.

### See Also

DataTrim()

---

DataInit *Initialize Data Input Files*

---

### Description

Creates template for the Data input file (Excel or CSV) and Data documentation file (Markdown) in the working directory or the directory specified by the `dir` argument

### Usage

```
DataInit(name = "Data", ext = c("xlsx", "csv"), overwrite = FALSE, dir = NULL)
```

### Arguments

| | |
|---|---|
| `name` | Name of the data input files. Default is 'Data'. Use 'Example' to create populated example Data Input and Data Documentation files. |
| `ext` | Optional file extension for input file. 'xlsx' (default) or 'csv' |
| `overwrite` | Logical. Overwrite existing files? |
| `dir` | Optional directory path to create the Data files. Default is 'getwd()" |

### Value

Nothing. Creates template data files in the working directory.

### Author(s)

A. Hordyk

### Examples

```
## Not run:
DataInit("Example") # populated example
DataInit("myData") # empty template

## End(Not run)
```

`DataLag` *Data Lag for OM Objects*

## Description

Access or replace the `DataLag` slot of an om object: the number of additional whole years that the `Data` seen by an MP is lagged behind the timestep at which its advice takes effect. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

## Usage

```
DataLag(x)

DataLag(x) <- value
```

## Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a non-negative integer, in years. |

## Details

Use `ManagementScheduleTable()` to see which data year is used at each management year for a given OM.

In a real fishery, management advice is never based on fully up to date data. Even without any extra delay, an MP can only ever see data through the timestep immediately before the one at which its advice takes effect, since a timestep's data cannot be complete until that timestep has ended. `DataLag` adds a further, deliberate delay on top of that unavoidable one-timestep gap: `DataLag = 0` is that minimum unavoidable gap, and each extra year of `DataLag` pushes the data the MP sees one further whole year into the past.

**Management years and when advice takes effect:**

A "management year" is a projection year in which the MP is actually called to produce new advice. Which years are management years is controlled by `Interval`: with `Interval = 1` every projection year is a management year, with `Interval = 2` every second year is, and so on. In a year that is not a management year, the TAC or Effort from the most recent management year simply carries forward unchanged. Whatever advice is produced for a management year takes effect starting in that same calendar year, not the year after. Before `MPStartYear` (if it is set), the MP is not called at all; advice instead comes from `InterimAdvice`.

**Which data year the MP sees:**

The MP is given data up to and including the timestep immediately before the current management timestep, shifted back by a further `DataLag` whole years. This applies uniformly from the very first management year onward, reaching back into the operating model's historical data if needed, and it does not depend on `Interval`: the one-timestep

minimum gap is always relative to the single timestep before the current management timestep, not to whenever the MP was previously called.

For an annual OM (`Seasons = 1`), a timestep is a calendar year, so this is simply: with a last historical year of 2026, `MPStartYear = NULL` (so MPs start in 2027), `Interval = 1`, and `DataLag = 1`, the very first management year (2027) is based on data up to 2025 (2026 is the year immediately before 2027, and shifting that back one further year for the lag lands on 2025).

If `MPStartYear` is set, the years before it are "interim" years in which `InterimAdvice()` is used instead of the MP, but the operating model still simulates real catch and effort for those years and records data for them (f they are missing in the OM Data). Once the MP starts being called at `MPStartYear`, `DataLag` counts back from the timestep before that management year in exactly the same way, reaching into the interim period's data rather than treating the operating model's original historical period as the most recent data available.

**Seasonal OMs:**

For a seasonal OM (`Seasons > 1`), `DataLag` is still always specified in whole years; internally it is converted to `DataLag * Seasons` timesteps. For example, in a quarterly OM (`Seasons = 4`), a management timestep of quarter 1, 2028 with `DataLag = 1` uses data up to quarter 4, 2026 (quarter 4, 2027 is the timestep immediately before quarter 1, 2028, and shifting that back one further year lands on quarter 4, 2026).

**Per-MP override:**

An MP can declare `attr(mp, 'DataLag') <- value` to use that lag in place of the OM's `DataLag`. The status-quo MPs `CurrentCatch()`, `CurrentLandings()`, and `AverageCatch()` set it to `0`.

`DataLag` combines with `Interval` and `MPStartYear` to determine the full management timeline. `ManagementScheduleTable()` shows, for a given OM, which years are management years and which data year `DataLag` calls for at each one.

## Value

- `DataLag()` returns the value of the `DataLag` slot.
- `DataLag<-()` returns `x` with the `DataLag` slot updated.

## See Also

- `OM()` for the constructor, om for the class definition.
- `ManagementScheduleTable()` for previewing the resulting data/management year schedule.
- `Interval()`, `MPStartYear()` for the other slots controlling the management timeline.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM(DataLag = 1)
DataLag(om)
DataLag(om) <- 2

# Preview which years are management years, and which data year each uses
om <- OM(CurrentYear = 2026, DataLag = 1, Interval = 1)
ManagementScheduleTable(om)
```

---

## DataRichObs *Data-Rich Observation Model*

---

### Description

An obs object representing a fully observed scenario in which all data types are available: landings, discards, fishing effort, a commercial CPUE index, a fishery-independent survey, and both age- and length-composition samples from landed and discarded fish.

### Usage

```
DataRichObs
```

### Format

An obs object with all slots populated:

- `Landings` (catchobs): precisely reported landings with CV in `[0.03, 0.08]` and near-unbiased reporting `[0.98, 1.02]`, reflecting well-monitored logbook compliance.
- `Discards` (catchobs): observer-estimated discards with CV in `[0.10, 0.20]` and bias in `[0.90, 1.10]`.
- `Effort` (effortobs): electronically monitored vessel-days with CV in `[0.02, 0.05]` and negligible bias `[0.98, 1.02]`.
- `CPUE` (indicesobs): logbook-derived standardised CPUE with CV in `[0.10, 0.20]` and mild autocorrelation `[0.0, 0.2]`.
- `Survey` (indicesobs): annual spawning-biomass survey with CV in `[0.08, 0.15]` and negligible autocorrelation `[0.0, 0.1]`.
- `LandingsAtAge` (compobs): stratified port-sampling age compositions of landed fish. Nominal sample size `[300, 500]`; ESS `[80, 150]`; `Theta` in `[0.6, 1.0]`.
- `DiscardsAtAge` (compobs): observer age samples from discarded fish. Nominal sample size `[150, 250]`; ESS `[50, 100]`; `Theta` in `[0.5, 0.9]`.
- `LandingsAtSize` (compobs): dockside length-composition samples from landed fish. Nominal sample size `[400, 700]`; ESS `[100, 180]`; `Theta` in `[0.6, 1.0]`.
- `DiscardsAtSize` (compobs): observer-measured length compositions from discarded fish. Nominal sample size `[200, 350]`; ESS `[60, 120]`; `Theta` in `[0.5, 0.9]`.

### See Also

obs, `Obs()`, `CatchObs()`, `EffortObs()`, `IndicesObs()`, `CompObs()`, CatchAndSurveyObs, CommercialFleetObs, AgeStructuredObs, LengthStructuredObs

Other obs-examples: `AgeStructuredObs`, `CatchAndSurveyObs`, `CommercialFleetObs`, `LengthStructuredObs`

### Examples

```
DataRichObs
Survey(DataRichObs)
LandingsAtAge(DataRichObs)
DiscardsAtSize(DataRichObs)
```

---

## DataSlots *DataSlots*

---

### Description

Dataframe with details of slots in Dat object

### Usage

```
DataSlots
```

### Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 101 rows and 4 columns.

---

## DataTrim *Trim Data Object by Final Year*

---

### Description

Truncate a `Data` object so that all time-dependent arrays are trimmed to the specified final year.

### Usage

```
DataTrim(Data, Year)
```

### Arguments

| | |
|---|---|
| `Data` | A `data` class object containing year-indexed information. |
| `Year` | A single numeric value specifying the final year to retain.<br>Arrays that are not indexed by `Year` are left unchanged. If the supplied year is already the maximum year in the object, the input object is returned unchanged. The `Advice` object is preserved from the original `Data` object.<br>• `Year` must be numeric and of length one.<br>• `Year` must be contained in `Years(Data)`. |

### Value

A `Data` object where all year-dependent slots include only years less than or equal to `Year`.

### See Also

- `Years()` for extracting available years

---

`DecE_Dom` *Fleet class objects*

---

### Description

Example objects of class Fleet

### Usage

```
DecE_Dom

DecE_HDom

DecE_NDom

FlatE_Dom

FlatE_HDom

FlatE_NDom

Generic_DecE

Generic_FlatE

Generic_Fleet

Generic_IncE
```

`IncE_HDom`

`IncE_NDom`

`Low_Effort_Non_Target`

`Target_All_Fish`

`Targeting_Small_Fish`

### Format

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

An object of class `Fleet` of length 1.

### Examples

```
avail("Fleet")
```

---

## `DefaultAges` *Default Ages and Years*

---

### Description

Convenience functions that return a sensible default when their argument is `NULL`. Used internally to allow optional `Ages` and `Years` arguments throughout the package without requiring callers to always supply them.

### Usage

```
DefaultAges(Ages = NULL)

DefaultYears(Years = NULL)
```

### Arguments

| | |
|---|---|
| `Ages` | An ages object, or `NULL` (default). |
| `Years` | Numeric vector of years, or `NULL` (default). |

### Details

- `DefaultAges()`: returns `Ages` unchanged if supplied, otherwise creates an ages object with `MaxAge = 10` via `Ages()`.
- `DefaultYears()`: returns `Years` unchanged if supplied, otherwise generates an annual sequence from 1950 to five years beyond the current calendar year.

### Value

- `DefaultAges()`: an ages object.
- `DefaultYears()`: a numeric vector of integer years.

### See Also

`Ages()`, ages

### Examples

```
DefaultAges()
DefaultYears()
```

---

## `DefaultSlickPMs` *Default Performance Metrics for* `MSE2Slick()`

---

### Description

The default `PMs` list used by `MSE2Slick()` (and, unless overridden, by every one of its `BoxplotPMs`/`QuiltPMs`/`SpiderPMs`/`TradeoffPMs` panel-specific arguments) when none is supplied: four metrics, chosen to give a reasonably complete first look at a set of MPs without the caller having to assemble a `PMs` list themselves.

### Usage

```
DefaultSlickPMs()
```

### Details

- PM_Yield(): mean projected catch, on its natural (biomass) scale; in the Spider it is relative to the highest value among the MPs.
- PM_AAVY(): average annual variability in the TAC, i.e. how much the TAC changes between management cycles; also natural-scale.
- PM_Status(): the joint probability that a stock/complex is neither overfished nor experiencing overfishing (P(SB > SBMSY & F < FMSY)).
- PM_Safety() with Lim = 0.4: the probability that SB/SBMSY never drops below 40% at any point in the projection.

Pass your own list to MSE2Slick(PMs = ...) to override this entirely, or start from DefaultSlickPMs() and append/replace entries (see MSE2Slick()'s Details section).

### Value

A list of performance-metrics, in the shape MSE2Slick()'s PMs argument expects.

### See Also

MSE2Slick(), PM

---

## DeleteLogs *Delete the Log Directory*

---

### Description

Delete the Log Directory

### Usage

```
DeleteLogs(Dir = file.path(getwd(), "Log"))
```

### Arguments

| | |
|---|---|
| Dir | Directory where the logs are kept. |

### Value

Nothing

`Depletion` *Depletion Constructor and Accessors*

## Description

Construct a depletion object defining the initial and/or final depletion assumptions for a stock, or access and replace the `Depletion` slot of a stock and its individual slots. `Depletion` is optional — see *Default Behaviour* below.

## Usage

```
Depletion(Initial = NULL, Final = NULL, Reference = "B0")

Initial(x)

Initial(x) <- value

Final(x)

Final(x) <- value

Depletion(x) <- value
```

## Arguments

| | |
|---|---|
| `Initial` | `numeric` or `NULL`. Depletion relative to `Reference` at the start of the historical period. Accepted formats:<br>• `NULL` (default): stock is assumed unfished at the start of the historical period. No adjustment to recruitment deviations.<br>• Scalar: same depletion for all simulations.<br>• Length-2 bounds vector: depletion sampled from `Uniform(lower, upper)` independently for each simulation.<br>• Length-`nSim` vector: one value per simulation. When provided, the historical period starts from a fished equilibrium with biomass relative to `Reference` in the first time step equal to the specified value (see *Initial Depletion*). When `Initial` is a stock object, `Depletion()` acts as a pass-through accessor and returns `x@Depletion`. |
| `Final` | `numeric` or `NULL`. Target depletion relative to `Reference` in the terminal historical time step. Accepts the same formats as `Initial`. When populated, the `Efficiency` parameter in `Catchability()` is optimised to achieve this depletion level, overwriting any existing `Efficiency` values. When `NULL` (default), no optimisation occurs and `Catchability()` must be populated directly with valid values. |
| `Reference` | `character(1)`. Reference biomass used to scale depletion. Currently implemented options:<br>• `"B0"` (default): total unfished equilibrium biomass. |

- `"SB0"`: spawning biomass at unfished equilibrium. The options `"BMSY"`, `"SBMSY"`, `"SP0"`, and `"SPMSY"` are reserved for future use and currently pass validation but are not implemented.

`x` A depletion object for slot accessors, or a stock object for `Depletion<-`.

`value` For `Depletion<-`: a depletion object. For `Initial<-` and `Final<-`: a numeric scalar, length-2 bounds vector, or length-`nSim` vector.

## Details

### Default Behaviour When Omitted:

`Depletion` is optional. When both `Initial` and `Final` are `NULL` (the default), the object has no effect on model behaviour:

- The stock is assumed unfished at the start of the historical period.
- Depletion in the terminal year is determined by the interaction between `Stock()` and `Fleet()` parameters; `Catchability()` must be populated directly in this case.

### Initial Depletion:

When `Initial` is supplied, the historical period starts from the equilibrium age structure, spatial distribution and recruitment produced by constant fishing with the first historical year's biology, selectivity, retention, discard mortality, and fleet and seasonal effort pattern (`Effort x Efficiency`). If no fleet has effort in the first year, the effort pattern of the first year with effort is used. Equilibrium recruitment follows the stock-recruit relationship. The fishing mortality is scaled so that biomass (`Reference = "B0"`) or spawning biomass (`"SB0"`) in the first time step, after applying the initial recruitment deviations, equals `Initial` times the unfished equilibrium value in that time step. The solved multiplier on the fishing pattern is stored in `Depletion@Misc$InitialEffortScale`. Values must be in (0, 1]:

```
# Fixed initial depletion across all simulations
dep <- Depletion(Initial = 0.8)

# Stochastic initial depletion - sampled from Uniform(0.6, 0.9)
dep <- Depletion(Initial = c(0.6, 0.9))

# One value per simulation
dep <- Depletion(Initial = runif(48, 0.6, 0.9))
```

### Final Depletion:

When `Final` is supplied, the operating model optimises `Efficiency` in `Catchability()` so that depletion in the terminal historical year matches the target. Any existing `Efficiency` values are overwritten. When `Final` is `NULL`, `Efficiency` is used as supplied:

```
# Force terminal depletion to 0.4 relative to B0
dep <- Depletion(Final = 0.4)

# Stochastic terminal depletion
dep <- Depletion(Final = c(0.3, 0.5))
```

```
# Both initial and final depletion specified
dep <- Depletion(Initial = 0.9, Final = 0.4)

# Relative to spawning biomass at unfished equilibrium
dep <- Depletion(Final = 0.4, Reference = "SB0")
```

**Pass-Through Access from a Stock:**

When `Initial` is a stock object, `Depletion()` returns the `Depletion` slot directly:

```
Depletion(my_stock)             # returns my_stock@Depletion
Depletion(my_stock) <- my_dep   # replaces my_stock@Depletion
```

**Slot Accessors:**

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Initial(dep)   <- 0.8
Final(dep)     <- 0.4
```

## Value

- `Depletion()` returns a depletion object. If `Initial` is a stock, returns `x@Depletion`.
- `Depletion<-` returns the stock `x` with the `Depletion` slot replaced and the object re-validated.
- `Initial()` and `Final()` return the value of the corresponding slot from `x` (a `nSim`-length array after `Populate()`, or `NULL`).
- `Initial<-` and `Final<-` return `x` with the named slot updated and the object re-validated.

## See Also

- depletion for the class definition and slot-level documentation.
- `Catchability()` for the fleet catchability object whose `Efficiency` parameter is optimised when `Final` is populated.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.

Other depletion: `depletion-class`

## Examples

```
# Default (Depletion omitted): stock starts unfished, terminal depletion
# determined by Fleet and Catchability parameters.
dep <- Depletion()

# Fixed initial depletion - same value for every simulation
dep <- Depletion(Initial = 0.8)

# Stochastic initial depletion - sampled from Uniform(0.6, 0.9) per simulation
dep <- Depletion(Initial = c(0.6, 0.9))
```

```
# Force terminal-year depletion to 0.4 relative to B0.
# Catchability Efficiency is optimised to achieve this.
dep <- Depletion(Final = 0.4)

# Both initial and final, relative to spawning biomass at unfished
# equilibrium (SB0) instead of the default total biomass (B0)
dep <- Depletion(Initial = 0.9, Final = 0.4, Reference = "SB0")

# Slot accessors
Initial(dep)                # 0.9
Initial(dep) <- c(0.7, 0.9)   # now stochastic
Final(dep)   <- 0.3

# Attaching to a Stock (pass-through access)
stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
Depletion(stk) <- dep
Initial(Depletion(stk))

# After Populate(), Initial and Final become nSim-length arrays
pop_dep <- Populate(dep, nSim = 48)
Initial(pop_dep)
```

---

`depletion-class` *The* `depletion` *S4 Class*

---

## Description

Defines the initial and final depletion assumptions for a stock object, expressed relative to a reference biomass. Objects are typically created via `Depletion()`, which documents all parameters and validates inputs. This object is optional — see *Default Behaviour* in `Depletion()` for what happens when it is omitted.

## Details

Direct construction via `methods::new()` is not recommended; use `Depletion()` instead, which validates inputs.

Both `Initial` and `Final` may remain `NULL` after `Populate()` — `NULL` indicates the slot is not in use rather than an uninitialised state. When both are `NULL`, the `depletion` object has no effect on model behaviour.

## Slots

`Initial` `array` or `NULL`. Depletion relative to `Reference` in the first historical time step, as a `nSim`-length array after `Populate()`. Before population, may be `NULL` (stock assumed dynamic unfished at the start of the historical period), a scalar, a length-2 bounds vector, or a length-`nSim` vector. See `Depletion()` for full details.

`Final array` or `NULL`. Target depletion relative to `Reference` in the terminal historical time step, as a `nSim`-length array after `Populate()`. When populated, the `Efficiency` parameter in `Catchability()` is optimised to achieve this depletion level, overwriting any existing `Efficiency` values. When `NULL`, no optimisation occurs and `Catchability()` must be populated directly. See `Depletion()` for full details.

`Reference character(1)`. Reference biomass used to scale depletion values. Currently implemented options are `"B0"` (total unfished biomass, default) and `"SB0"` (spawning biomass at unfished equilibrium). The options `"BMSY"`, `"SBMSY"`, `"SP0"`, and `"SPMSY"` are accepted by the validator but are reserved for future use.

`Misc list`. Used internally.

### See Also

- `Depletion()` for the constructor and accessor functions.
- `Catchability()` for the fleet catchability object whose `Efficiency` parameter is optimised when `Final` is populated.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.

Other depletion: `Depletion()`

---

## `DFO_bar` *Department of Fisheries and Oceans stock status bar plot*

---

### Description

A plot of biomass relative to BMSY over projected years

### Usage

```
DFO_bar(MSEobj, yres = 10)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An MSE object of class MSE produced by DLMtool function runMSE |
| `yres` | Integer: the year interval over which to calculate B/BMSY in future years |

### Author(s)

T. Carruthers

## DFO_hist *Department of Fisheries and Oceans historical plot*

### Description

A plot of current and historical stock status by simulation according to the stock status zones and reference points of DFO. http://www.dfo-mpo.gc.ca/reports-rapports/regs/sff-cpd/precaution-eng.htm

### Usage

```
DFO_hist(OM, panel = T, nsim = 48)
```

### Arguments

| | |
|---|---|
| OM | An operating model object of class OM |
| panel | should the plots be separate or in two panels? |
| nsim | how many simulations should be plotted (over-ridden by OM@nsim where cpars is specified) |

### Author(s)

T. Carruthers

## DFO_plot *Department of Fisheries and Oceans trade-off plot*

### Description

A plot of mean biomass relative to BMSY and fishing mortality rate relative to FMSY over the final 5 years of the projection http://www.dfo-mpo.gc.ca/reports-rapports/regs/sff-cpd/precaution-eng.htm

### Usage

```
DFO_plot(MSEobj, zero_origin = T)
```

### Arguments

| | |
|---|---|
| MSEobj | An MSE object of class MSE produced by MSEtool function runMSE |
| zero_origin | Logical: should plots have a zero-zero origin? |

### Author(s)

T. Carruthers

## DFO_plot2 *Department of Fisheries and Oceans default plot 2*

### Description

A preliminary plot for returning trade-offs plots and performance table for probability of obtaining half reference (FMSY) yield and probability of biomass dropping below 50 per cent BMSY

### Usage

```
DFO_plot2(MSEobj, nam = NA, panel = T, Bcut = 50, Ycut = 50)
```

### Arguments

| | |
|---|---|
| MSEobj | An object of class MSE |
| nam | Title of plot |
| panel | Should the plots be organized in many panels in a single figure |
| Bcut | The cutoff biomass for satisficing (relative to BMSY) |
| Ycut | the cutoff yield for satisficing (relative to reference yield) |

### Value

A table of performance metrics.

### Author(s)

T. Carruthers

## DFO_proj *Department of Fisheries and Oceans projection plot*

### Description

A projection plot of MP performance by simulation according to the stock status zones and reference points of DFO. http://www.dfo-mpo.gc.ca/reports-rapports/regs/sff-cpd/precaution-eng.htm

### Usage

```
DFO_proj(MSEobj, maxplot = 6)
```

### Arguments

| | |
|---|---|
| MSEobj | An operating model object of class MSE |
| maxplot | The maximum number of MPs to be plotted per figure |

### Author(s)

T. Carruthers

---

`DFO_quant` *Department of Fisheries and Oceans biomass quantile plot*

---

### Description

A plot of biomass relative to BMSY quantiles over projected years

### Usage

```
DFO_quant(
  MSEobj,
  maxcol = 6,
  qcol = rgb(0.4, 0.8, 0.95),
  lcol = "dodgerblue4",
  curyr = 2018,
  quants = c(0.05, 0.25, 0.75, 0.95),
  addline = T,
  forreport = T
)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An MSE object of class MSE produced by DLMtool function runMSE |
| `maxcol` | Integer how many columns for panel plots? |
| `qcol` | A color, the quantile coloration |
| `lcol` | A color, the mean B/BMSY line |
| `curyr` | The current calendar year |
| `quants` | A vector 2 long for the quantiles e.g. 0.1 and 0.9 for the 10th and 90th quantiles |
| `addline` | Should two individual example simulations be added to the plot? |
| `forreport` | Logical. Is it for a report? If true, one plot of six MPs in a row will be provided one after another. |

### Author(s)

T. Carruthers

## DFO_report *Create a standard DFO MSE report*

### Description

Provides performance plots typical in the assessment of Canadian fish stocks.

### Usage

```
DFO_report(
  MSEobj,
  output_file = NA,
  author = "Author not specified",
  title = NA,
  maxMPs = 15
)
```

### Arguments

| | |
|---|---|
| MSEobj | An object of class MSE |
| output_file | The directory and filename you wish to use for the report e.g. "C:/temp/myMSEreport.html" |
| author | The person who made this report |
| title | The title of the report |
| maxMPs | Maximum number of MPs to plot |

### Author(s)

T. Carruthers

## DFO_spider *DFO performance spider plot (top three MPs)*

### Description

DFO performance spider plot (top three MPs)

### Usage

```
DFO_spider(MSEobj)
```

### Arguments

| | |
|---|---|
| MSEobj | An object of class MSE produced by MSEtool::runMSE() |

### Author(s)

T. Carruthers

## DFO_tab *Create a standard DFO performance table*

### Description

P_Cr_S is the probability of being in the critical zone in the first 10 projected years P_Ct_S is the probability of being in the cautious zone in the first 10 projected years P_H_S is the probability of being in the healthy zone in the first 10 projected years POF_S is the probability of overfishing in the first 10 projected years STY is the mean yield relative to FMSY management over the first 10 projected years P_Cr_L is the probability of being in the critical zone in the last 10 projected years P_Ct_L is the probability of being in the cautious zone in the last 10 projected years P_H_L is the probability of being in the healthy zone in the last 10 projected years POF_L is the probability of overfishing in the last 10 projected years LTY is the mean yield relative to FMSY management over the last 10 projected years AAVY is the average annual variability in yield over the whole projection phrased as a CV percentage P_Reb is the probability the stock has rebuilt to over BMSY in 2 mean generation times

### Usage

```
DFO_tab(MSEobj, maxMPs = 15, rnd = 0)
```

### Arguments

| | |
|---|---|
| MSEobj | An object of class MSE |
| maxMPs | Integer: the maximum number of top ranking MPs to include in the table (ranked by long term yield) |
| rnd | The number of significant figures for rounding. |

### Author(s)

T. Carruthers

## DFO_tab_formatted *A formatted version of the standard DFO performance plot, color coded by thresholds*

### Description

Crit_S is the probability of being in the critical zone in the first 10 projected years Caut_S is the probability of being in the cautious zone in the first 10 projected years Health_S is the probability of being in the healthy zone in the first 10 projected years OvFish_S is the probability of overfishing in the first 10 projected years Yield_S is the mean yield relative to FMSY management over the first 10 projected years Crit is the probability of being in the critical zone in the last 10 projected years Caut is the probability of being in

the cautious zone in the last 10 projected years Health is the probability of being in the healthy zone in the last 10 projected years OvFish is the probability of overfishing in the last 10 projected years Yield is the mean yield relative to FMSY management over the last 10 projected years AAVY is the average annual variability in yield over the whole projection phrased as a CV percentage Reb is the probability the stock has rebuilt to over BMSY in 2 mean generation times

### Usage

```
DFO_tab_formatted(
  Ptab1,
  thresh = c(30, 50, 40, 60, 50, 20, 40, 50, 60, 50, 30, 50),
  ret_thresh = F
)
```

### Arguments

| | |
|---|---|
| `Ptab1` | A DFO performance table made by DFO_tab() |
| `thresh` | A vector of thresholds for each column Health, Yield and Reb are 'greater than threshold' conditions |
| `ret_thresh` | Logical: if true just the threshold levels are returned |

### Author(s)

T. Carruthers

---

## `dim` *Dimensions of an Object*

---

### Description

Retrieve or set the dimension of an object.

### Usage

```
dim(x)
```

### Arguments

| | |
|---|---|
| `x` | an R object, for example a matrix, array or data frame.<br>This is a wrapper for `base::dim`. The only difference is that this function prints the dimension names (if applicable) |

### Examples

```
MyArray <- array(1:6,
  dim = c(6, 2),
  dimnames = list(
    Age = 1:6,
    Year = c(2025, 2026)
  )
)
dim(MyArray)
```

---

## dimension-utils *Add or Drop Named Dimensions of an Array*

---

### Description

Utilities to add or drop named dimensions from arrays.

### Usage

```
AddDimension(array, name, val = 1, pos = NULL)

DropDimension(array, name, warn = TRUE)
```

### Arguments

| | |
|---|---|
| `array` | An array with named dimensions. |
| `name` | A single character string naming the dimension to add (`AddDimension`) or a character vector of dimension names to drop (`DropDimension`). |
| `val` | Value(s) to assign as the dimname of the new dimension in `AddDimension`. Defaults to `1`. |
| `pos` | Integer giving the position at which to insert the new dimension in `AddDimension`. Defaults to the last position. |
| `warn` | Logical; if `TRUE`, warn when dropping a dimension of length $> 1$. |

### Details

#### AddDimension

Adds a new dimension to an array at position `pos` with the given `name` and `val` as its dimname values. If the dimension name already exists, the array is returned unchanged.

#### DropDimension

Drops one or more named dimensions from an array. All requested dimensions must exist and at least one dimension must remain. Dimensions of length $> 1$ are sliced at their first index, with an optional warning.

### Value

An array with modified dimensions and updated `dimnames`.

### Examples

```
# ---- Setup Array ----
Arr <- array(1:24, dim = c(2, 3, 4),
               dimnames = list(
                 Sim  = 1:2,
                 Age  = 0:2,
                 Year = 2020:2023
               ))

dim(Arr)

# ---- AddDimension ----

# Append an Area dimension at the end (default pos)
AddDimension(Arr, 'Area') |> dim()

# Append an Area dimension with two values
AddDimension(Arr, 'Area', val = 1:2) |> dim()

# Adding a dimension that already exists returns the array unchanged
AddDimension(Arr, 'Sim') |> dim()

# ---- DropDimension ----

# Drop a length-1 dimension (no warning)
Arr <- array(1:12, dim = c(1, 3, 4),
           dimnames = list(Sim = 1, Age = 0:2, Year = 2020:2023))

DropDimension(Arr, 'Sim')

# Drop a length > 1 dimension (warns by default)
DropDimension(Arr, 'Age')

# Suppress the warning
DropDimension(Arr, 'Age', warn = FALSE)

# Drop multiple dimensions at once
DropDimension(Arr, c('Sim', 'Year'), warn = FALSE)
```

---

## DiscardMortality *Discard Mortality*

---

### Description

Construct and manipulate a discardmortality object defining the proportion of discarded catch that dies, for use in a `Fleet()` object. Discard mortality is optional; if not specified, all discarded fish are assumed to survive.

### Usage

```
DiscardMortality(
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  Classes = NULL,
  Misc = list()
)

DiscardMortality(x) <- value
```

### Arguments

| | |
|---|---|
| `MeanAtAge` | Numeric, numeric vector, or array, or `NULL`. Mean discard mortality at age. Values must be between 0 (all discards survive) and 1 (all discards die). Accepted forms:<br>• `NULL` (default): creates an empty discardmortality object. `PopulateDiscardMortality()` sets discard mortality to 0 for all age and length classes.<br>• Scalar: constant mortality applied to all ages. Wrapped internally to a `1 x nAge x 1` array.<br>• Numeric vector of length `nAge`: mortality specified per age class. Wrapped internally to a `1 x nAge x 1` array.<br>• Numeric array with dimensions `Sim x Age x Year` and named dimnames.<br>If `MeanAtAge` is a fleet object, the `DiscardMortality` slot of that fleet is returned.<br>Unlike `Selectivity()` and `Retention()`, `DiscardMortality` does not support model-based population via `Pars` and `Model`. Values must be supplied directly as arrays or derived from `MeanAtLength`. |
| `MeanAtLength` | Numeric array or `NULL`. Mean discard mortality at length with dimensions `Sim x Class x Year`. If provided, takes precedence over `MeanAtAge`; `MeanAtAge` is derived from it via the age-length key during `PopulateDiscardMortality()`. A numeric vector is also accepted and converted to a `1 x nClass x 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`. |
| `Classes` | Numeric vector or `NULL`. Length class midpoints corresponding to the second dimension of `MeanAtLength`. Default `NULL`. |
| `Misc` | List. Miscellaneous additional inputs. Default `list()`. |
| `x` | A discardmortality object, or a fleet, `FleetList`, `StockFleetList`, or om object for `DiscardMortality<-`. |
| `value` | For `DiscardMortality<-`: a discardmortality object when `x` is a fleet; a flat list of discardmortality objects when `x` is a `FleetList`; a nested stock-fleet list of discardmortality objects when `x` is a `StockFleetList` or om. |

## Details

Discard mortality defines the proportion of discarded catch that dies. Fish that are selected but not retained (as defined by `Retention()`) are treated as discards; `DiscardMortality` determines how many of those fish die as a result of the capture-and-release process.

Unlike `Selectivity()` and `Retention()`, discard mortality does not support model-based population via `Pars` and `Model`. Values must be supplied directly as `MeanAtAge` or `MeanAtLength` arrays. If neither is supplied, `PopulateDiscardMortality()` defaults to 0 (all discards survive) for all age and length classes.

### Array Format:

`MeanAtAge` may be a scalar, a vector of length `nAge`, or a full `Sim x Age x Year` array. Scalars and vectors are wrapped internally to arrays with a single simulation and year, then extended to cover all simulations and years during population. If `MeanAtLength` is supplied, it takes precedence and `MeanAtAge` is derived from it using the age-length key.

### Attaching to a Fleet:

A discardmortality object can be attached to a `Fleet()` with `DiscardMortality(Fleet) <- MyDiscardMortality` and retrieved with `DiscardMortality(Fleet)`.

Individual slots may be accessed or modified using `MeanAtAge()`, `MeanAtLength()`, and `Classes()`.

## Value

- `DiscardMortality()` returns a discardmortality object. If `MeanAtAge` is a fleet object, the `DiscardMortality` slot of that fleet is returned.
- `DiscardMortality<-` returns `x` with the `DiscardMortality` slot replaced by `value`. When `x` is an om or `StockFleetList`, each fleet's slot is updated from the corresponding element of the nested list `value`. When `x` is a `FleetList`, each fleet's slot is updated from the corresponding element of the flat list `value`.

## See Also

- discardmortality for the class definition and slot-level documentation.
- `Fleet()` for the enclosing fleet constructor.
- `Selectivity()`, `Retention()` for related fleet components.
- `PopulateDiscardMortality()` for population details.

Other fleet: `Bioeconomic()`, `Catchability()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## Examples

```
## ---- Default: zero discard mortality ----
# An empty DiscardMortality object causes PopulateDiscardMortality() to set
# discard mortality to 0 for all age and length classes (all discards survive).
# This is the default when no DiscardMortality object is supplied to Fleet().
dm_empty <- DiscardMortality()
dm_empty
```

```
## ---- Constant discard mortality (scalar) ----
# A scalar is applied to all age classes across all simulations and years.
# Here 30% of discarded fish die regardless of age.
dm_scalar <- DiscardMortality(MeanAtAge = 0.3)

## ---- Age-varying discard mortality (vector) ----
# A vector of length nAge specifies mortality per age class.
# Younger fish (smaller, more vulnerable) may have higher discard mortality.
ages    <- 0:20
dm_vals <- pmax(1 - ages / 10, 0)   # declines from 1 at age 0 to 0 at age 10

dm_vec <- DiscardMortality(MeanAtAge = dm_vals)

## ---- Full array (Sim x Age x Year) ----
# A full array allows discard mortality to vary across simulations and years.
dm_array <- array(
  0.2,
  dim      = c(1, length(ages), 1),
  dimnames = list(Sim = 1, Age = ages, Year = 2000)
)

dm_full <- DiscardMortality(MeanAtAge = dm_array)
MeanAtAge(dm_full)

## ---- Specifying discard mortality at length ----
# MeanAtLength takes precedence over MeanAtAge when both are supplied.
# MeanAtAge is derived from MeanAtLength via the age-length key during
# PopulateDiscardMortality().
lengths  <- seq(10, 100, by = 5)
dm_at_l  <- pmax(1 - lengths / 60, 0)   # mortality declines with length

dm_length_array <- array(
  dm_at_l,
  dim      = c(1, length(lengths), 1),
  dimnames = list(Sim = 1, Length = lengths, Year = 2000)
)

dm_len <- DiscardMortality(
  MeanAtLength = dm_length_array,
  Classes      = lengths
)
MeanAtLength(dm_len)

## ---- Attaching to a Fleet object ----
f <- Fleet(Name = "Trawl")
DiscardMortality(f) <- DiscardMortality(MeanAtAge = 0.2)
DiscardMortality(f)
MeanAtAge(DiscardMortality(f))

## ---- Extracting DiscardMortality from a Fleet ----
# Passing a fleet object as the first argument returns its DiscardMortality slot.
dm <- DiscardMortality(f)
```

```
dm
```

---

`discardmortality-class` *The* `discardmortality` *S4 Class*

---

## Description

Defines the proportion of discarded catch that dies, at age or at length, for use in a fleet object. Discard mortality is optional; if not specified, all discarded fish are assumed to survive. Objects are typically created via the `DiscardMortality()` constructor, which documents all parameters in detail.

## Slots

`MeanAtAge` `numeric` array or `NULL`. Mean discard mortality-at-age (`Sim x Age x Year x Area`). Values between 0 (all discards survive) and 1 (all discards die). See `DiscardMortality()`.

`MeanAtLength` `numeric` array or `NULL`. Mean discard mortality-at-length (`Sim x Length x Year x Area`). Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). See `DiscardMortality()`.

`Classes` `numeric` or `NULL`. Lower bounds of length bins corresponding to the second dimension of `MeanAtLength`. Bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength` are evaluated at bin midpoints, not at these lower bounds.

`Misc` `list`. Miscellaneous additional inputs. Used internally.

## See Also

- `DiscardMortality()` for the constructor and full parameter documentation.
- fleet for the enclosing fleet object.
- `Selectivity()`, `Retention()` for related fleet components.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

---

`DLMDataDir` *Directory of the installed package on your computer*

---

### Description

Directory of the installed package on your computer

### Usage

```
DLMDataDir(stock = NA)
```

### Arguments

| | |
|---|---|
| `stock` | Character string representing the name of a .csv file e.g. 'Snapper', 'Rockfish' |

### Value

The file path to the object

---

`doHCR` *Hockey Stick Harvest control rule that modifies TAC.*

---

### Description

A hockey stick (2 inflection points) HCR that accepts estimated level relative to reference level and modifies a proposed TAC based on control points for the x axis (est/ref) and y axis (fraction of TAC)

### Usage

```
doHCR(trial_TAC, est, ref, CP = c(0, 1), CPy = c(0, 1))
```

### Arguments

| | |
|---|---|
| `trial_TAC` | Postitive real number, the proposed total allowable catch before HCR modification. |
| `est` | Positive real number on same scale as ref, the estimated stock level (e.g. mean current index level) |
| `ref` | Positive real number on same scale as est, a reference level of stock level (e.g. index level at BMSY) |
| `CP` | Vector of real numbers, 2 positions long (c(Lx, Ux)), the lower and upper control points of a hockey stick HCR on the xaxis (est/ref). Below Lx (est/ref $<$ Lx) the TAC is trial_TAC x Ly. Above Ux (est/ref $>$ Ux) the TAC is trial_TAC x Uy. Between the TAC is linearly ramped between these levels. |
| `CPy` | Vector of real numbers, 2 positions long (c(Ly, Uy)), the lower and upper control points of a hockey stick HCR on the yaxis (fraction of trial_TAC). |

### Value

A real number (TAC advice but theoretically could be used for effort, size limits etc).

### Author(s)

T. Carruthers

---

`doIfreq` *Create indices that are sampled at various frequencies*

---

### Description

Given an index (historical period and projected period) this function creates sparsity in the projected index to simulate varying frequency (intensity) of data collection.

### Usage

```
doIfreq(I_hist, I_freq, LHYr, CurYr, Year)
```

### Arguments

| | |
|---|---|
| `I_hist` | Vector of real numbers, concatinated observed (historical) and simulated (projected) indices. |
| `I_freq` | Positive integer. The frequency of index sampling (e.g. 1 is every year, 2 is every 2 years - a gap every 2 years in the projected, simulated data). |
| `LHYr` | Positive integer, a year (e.g. 2023), the last historical year, demarks the historical period where observations have been collected from the projected period where sparsity is to be simulated. |
| `CurYr` | Positive integer, a year (e.g. 2043), the most recent year of the simulation. |
| `Year` | Vector of positive integers (as long as I_hist), the years corresponding with I_hist. |

### Value

A thinned ector I_hist long of index observations.

### Author(s)

T. Carruthers

Dom *Determine dominate MPs*

## Description

MPs that perform worse than comparable MPs across all performance metrics are considered 'dominated' as other options are always preferable.

## Usage

```
Dom(MSEobj, ..., PMlist = NULL, Refs = NULL, Yrs = NULL)
```

## Arguments

| | |
|---|---|
| MSEobj | An object of class MSE |
| ... | Names of Performance Metrics (PMs), or other arguments to TradePlot. First PM is recycled if number of PMs is not even |
| PMlist | Optional list of PM names. Overrides any supplied in ... above |
| Refs | An optional named list (matching the PM names) with numeric values to override the default Ref values. |
| Yrs | An optional named list (matching the PM names) with numeric values to override the default Yrs values. |

## Details

The Dom function compares the probabilities calculated in the performance metric (PM) functions and determines the MPs that have a lower probability across all PMs compared to other MPs of the same management type (e.g., size limit, TAC, etc). Consequently, it is important that all PM functions are constructed so that higher probabilities = better performance (e.g, PNOF is the probability of NOT overfishing)

## Value

A named list of length 2 with a character vector of non-dominated MPs in MPs and a data.frame of dominated MPs and the names of the relevant dominated MPs in DomMPs

## Author(s)

A. Hordyk

## Examples

```
## Not run:
MSE <- runMSE(MPs=NA) # run all MPs
Nondom <- Dom(MSE, "P10", "LTY", "PNOF")
# Non-dominated MPs
Nondom$MPs
```

```
# Dominated MPs
Nondom$DomMPs


## End(Not run)
```

---

`DomeExFleet` *Dome-Shaped Selectivity Example Fleet*

---

## Description

An example fleet object demonstrating a fleet with increasing historical effort and dome-shaped length-based selectivity. Intended for use with AlbacoreExStock or ButterfishExStock to illustrate fleet parameterization.

## Usage

```
DomeExFleet
```

## Format

A fleet object with the following slots populated:

- `Name`: `"DomeExFleet"`. Unique fleet identifier.
- `Effort`: An `Effort()` object with a piecewise-linear historical effort trajectory defined at four relative time points (0, 0.3, 0.6, 1.0), with lower and upper bounds of (0, 0.4, 0.4, 1) and (0, 0.6, 0.6, 1) respectively, and `CV = 0.1` controlling stochastic variation around the trend.
- `Selectivity`: A `Selectivity()` object with dome-shaped length-based selectivity parameters — `L5` (0.2–0.4), `LFS` (0.75–1.1), and `Vmaxlen` (0.5–1.0) — with `isRel = TRUE` indicating that `L5` and `LFS` are expressed relative to maturity `L50`.
- `Catchability`: An empty `Catchability()` object; efficiency defaults to 1 across all years during population (see Details).
- `Retention`: An empty `Retention()` object; full retention is assumed during population (see Details).
- `DiscardMortality`: An empty `DiscardMortality()` object. Discard mortality is assumed to be 0.
- `Closure`: `NULL`. No spatial or temporal closures defined.
- `WeightFleetRetained`, `WeightFleetSelected`: `NA`. Computed from selectivity, retention, and the stock's weight-at-length during population (see Details).
- `Bioeconomic`: An empty `Bioeconomic()` object.
- `nYear`, `pYear`, `nSim`, `CurrentYear`, `Years`, `Seasons`: Inherited from the paired `Stock()` object by `PopulateFleet()`.
- `Misc`: Empty list for user-defined information.

## Details

PopulateFleet() expands the input parameters into [nSim x nAge x nYear] arrays using the age, length, and spatial structure of the paired populated stock object. The sections below describe what each component-level Populate*() function produces.

### Effort:

PopulateEffort() interpolates the piecewise-linear effort trajectory across all historical years for each simulation, then applies log-normal stochastic deviates scaled by CV = 0.1. The trajectory rises from 0 to a plateau at 40–60% of maximum effort around 30–60% of the historical period, then increases to full effort by the final historical year.

### Catchability:

Because the Catchability slot is empty, PopulateCatchability() sets catchability efficiency to 1 across all simulations and years. No trend (qInc) or inter-annual variability (qCV) is applied.

### Selectivity:

PopulateSelectivity() samples L5, LFS, and Vmaxlen for each simulation from uniform distributions defined by the prior bounds. Because isRel = TRUE, L5 and LFS are scaled by the maturity L50 of the paired stock before computing the selectivity curve. The resulting dome-shaped selectivity-at-length is converted to selectivity-at-age using the stock age–length key, and an area dimension is added. The Vmaxlen range (0.5–1.0) allows selectivity at maximum length to vary from strongly dome-shaped to asymptotic across simulations.

### Retention:

Because the Retention slot is empty, PopulateRetention() sets full retention (1.0) across all ages, lengths, simulations, and years.

### Discard Mortality:

Because the DiscardMortality slot is empty, PopulateDiscardMortality() applies default discard mortality values. All discarded fish are assumed to survive (discard mortality = 0) unless overridden.

### Spatial Closures:

Because Closure is NULL, PopulateClosure() applies no spatial or temporal closures; all areas are open to fishing in all years.

### Fleet Weight-at-Age:

Because WeightFleetRetained/WeightFleetSelected are NA, PopulateFleet() computes them from selectivity, retention, and the stock's weight-at-length, so fleet-level landings/discards biomass calculations reflect the fleet's own selectivity and retention pattern rather than the stock's unselected mean weight-at-age.

## See Also

AlbacoreExStock, ButterfishExStock, Fleet(), PopulateFleet(), Populate(), fleet

### Examples

```
DomeExFleet

## Not run:
PopulatedStock <- PopulateStock(AlbacoreExStock, nYear = 50, pYear = 30, nSim = 48)
PopulatedFleet <- PopulateFleet(DomeExFleet, Stock = PopulatedStock, seed = 42)

## End(Not run)
```

---

## doRec *Calculate a management recommendation given constraints*

---

### Description

Creates a TAC management recommendation given constraints on how much that can change from previous TAC and contraints on minimum and maximum TAC

### Usage

```
doRec(MPrec, mod, delta_down, delta_up, TACrng)
```

### Arguments

| | |
|---|---|
| `MPrec` | Positive real number, the previous management recommendation (e.g. 100 tonnes). |
| `mod` | Imperfect fraction, the proposed modification (change to MPrec) (e.g. 1.2 is a 20% increase) |
| `delta_down` | A vector 2 positions long, the minimum and maximum levels of downward change (e.g. when mod $< 1$) in the recommendation. |
| `delta_up` | A vector 2 positions long, the minimum and maximum levels of upward change (e.g. when mod $> 1$) in the recommendation. |
| `TACrng` | A vector 2 positions long, the minimum and maximum TAC (same units as MPrec). |

### Value

n object of class `Rec`.

### Author(s)

T. Carruthers

`Effort` *Effort Constructor and Accessors*

## Description

Construct and manipulate an effort object defining historical fishing effort and spatial structure for a `Fleet()` object.

## Usage

```
Effort(
  Effort = NULL,
  Units = NULL,
  Distribution = NULL,
  Targeting = NULL,
  Maximum = NULL,
  Mode = NULL,
  TripsScalar = NULL,
  AnglerPerTrip = NULL,
  Theta = NULL,
  StockTargetingLambda = NULL,
  Misc = list(),
  df = TRUE
)

Effort(x) <- value

Distribution(x)

Distribution(x) <- value

Targeting(x)

Targeting(x) <- value

Maximum(x)

Maximum(x) <- value

Mode(x)

Mode(x) <- value

TripsScalar(x)

TripsScalar(x) <- value
```

```
AnglerPerTrip(x)

AnglerPerTrip(x) <- value

Theta(x)

Theta(x) <- value

StockTargetingLambda(x)

StockTargetingLambda(x) <- value
```

## Arguments

| | |
|---|---|
| `Effort` | Numeric array, data frame, or `NULL`. Historical fishing effort. Accepted forms:<br>• `NULL` (default): creates an empty effort object; no effort is populated by `PopulateEffort()`.<br>• Numeric array with dimensions `Sim x Year` and named dimnames. The `Year` dimension must match the historical years of the OM.<br>• Data frame in the format required by `GenHistEffort()`; a stochastic `Sim x Year` array is generated automatically during population.<br>If `Effort` is a fleet, effort, hist, obs, or mse object, the `Effort` slot of that object is extracted and returned (see Details). |
| `Units` | Character or `NULL`. Units of fishing effort (e.g., `"hours"`, `"trips"`). Default `NULL`. Note that effort is converted to fishing mortality via gear efficiency (`q`) defined in the associated catchability object. When `Units = "trips"` the effort array is already on the absolute trips scale and `TripsScalar` is ignored by bag-limit management procedures. |
| `Distribution` | Numeric array or `NULL`. Fraction of total effort allocated to each spatial area. Used only for spatial models (`nArea > 1`). Must have dimensions `Sim x Year x Area` with named dimnames, and values must sum to 1 over the `Area` dimension within each simulation and year. The `Sim` and `Year` dimensions may each be length 1 (replicated internally) or match `nSim` and the historical years respectively. Default `NULL`; if not specified, `PopulateEffort()` initialises all cells to `NA` and the internal spatial allocation algorithm fills them (see Details). Any cells set to a non-`NA` numeric value are treated as fixed overrides and are not modified by the algorithm. |
| `Targeting` | Numeric array or `NULL`. Spatial targeting concentration parameter (`lambda >= 0`) with dimensions `Sim x Year`. The `Sim` dimension may be length 1 or match `nSim`. The `Year` dimension must be length 1, match the historical years, or have named `Year` dimnames. Default `NULL`; a value of `0.8` is applied to all simulations and years during `PopulateEffort()`. See Details for how this parameter controls the concentration of effort across areas. |

`Maximum` Numeric array or `NULL`. Maximum allowable effort. Default `NULL`. Not currently used.

`Mode` Character or `NULL`. Controls whether spatial utility is calculated per unit area or as raw biomass. `"Density"` (default) divides exploitable biomass by the relative area size (`RelSize`) before computing utility, so fleets are attracted to areas with high biomass concentration regardless of area size. `"Biomass"` uses raw exploitable biomass, so larger areas are intrinsically more attractive. See the Technical Manual for equations.

`TripsScalar` Numeric array or `NULL`. Time-varying scalar $\alpha_f(t)$ converting the effort index to absolute angler trip counts via $T_f(t) = \alpha_f(t) \cdot E_f(t)$. Must have dimensions `Sim x Year` with named dimnames. The `Sim` dimension may be length 1 (replicated internally). The `Year` dimension may be length 1 (replicated internally) or span the full model time series; values for projection years beyond the historical period are typically held constant at the last historical value or extended by the analyst prior to use. Default `NULL`. Required by bag-limit management procedures when `Units != "trips"`. Ignored when `Units = "trips"`. See Details.

`AnglerPerTrip` Numeric array or `NULL`. Mean number of anglers per trip. Must have dimensions `Sim x Year` with named dimnames. The `Sim` dimension may be length 1 (replicated internally). The `Year` dimension may be length 1 (replicated internally) if the value does not vary over time, or span the full model time series if it does. Used by bag-limit management procedures to scale a per-angler bag limit to a fleet-level retention cap. Default `NULL`. See Details.

`Theta` Numeric, numeric array, or `NULL`. Overdispersion parameter $\theta$ of the negative binomial within-trip catch distribution. Accepted forms:

- `NULL` (default): `Theta` is not populated; bag-limit management procedures will error if `Theta` is required and not supplied.
- Scalar numeric (e.g. `1.2`): constant overdispersion applied across all simulations and years.
- Numeric vector length 2. Treated as lower and upper bounds of a uniform distribution, sampled once per simulation and held constant across years.
- Numeric vector length `nSim`. One value per simulation, held constant across years.
- Numeric array with dimensions `Sim x Year` and named dimnames, for `Theta` that varies over time. The `Sim` dimension may be length 1 (replicated internally). The `Year` dimension may be length 1 (replicated internally) or span the full model time series.

In every form other than a full `Sim x Year` array, the resolved value is stored with a `Year` dimension of length 1, named with the first historical year, and is replicated across all historical and projection years during `Simulate()`.

`StockTargetingLambda` Numeric, numeric array, or `NULL`. Multiplier on this fleet's resistance to changing its stock-targeting mix when solving for effort against multiple

| | |
|---|---|
| | stock complexes. Accepts the same forms as `Theta` (scalar, `Sim x 1`, or `Sim x Year`).<br>`NULL` (default) is equivalent to `1` and leaves the internally derived resistance unchanged. `0` removes the resistance entirely, letting the fleet reallocate its targeting freely. Values above `1` make the fleet hold its existing mix more strongly, approaching a fixed mix by around `100`. Only consulted for multi-complex operating models; ignored otherwise. See the Stock Targeting Resistance section below. |
| `Misc` | List. Miscellaneous additional inputs. Default `list()`. |
| `df` | Logical. Only used when `Effort` is a hist or mse object. If `TRUE` (default), a tidy `data.frame` is returned. If `FALSE`, the raw `Effort` array is returned. |
| `x` | An effort object for accessor and replacement functions. |
| `value` | The replacement value for the corresponding slot. |

## Details

An effort object defines total fleet fishing activity prior to spatial allocation. Effort is converted to fishing mortality via gear efficiency (`q`) in the associated catchability object.

### Effort Array Format:

When supplied as a numeric array, `Effort` must have dimensions `Sim x Year` with named dimnames. The `Sim` dimension may be length 1 (replicated internally by `PopulateEffort()`). The `Year` dimension must match the historical years of the OM exactly.

When supplied as a data frame, the format must meet the requirements of `GenHistEffort()`, which generates a stochastic `Sim x Year` array during population. The `Units` slot is set to `"unitless"` automatically in this case.

### Spatial Distribution:

`Distribution` is only used for spatial models (`nArea > 1`). It must sum to 1 over the `Area` dimension within each simulation and year. The `Sim` and `Year` dimensions may be length 1 (replicated internally). When not specified (default), `PopulateEffort()` initialises all cells to `NA`.

In `Simulate()` and `Project()`, the spatial allocation algorithm only fills cells that remain `NA`; any non-`NA` values supplied in `Distribution` are left unchanged and act as fixed overrides for those simulation-year-area combinations.

### Spatial Utility and Effort Allocation:

When `Distribution` cells are `NA`, effort allocation across areas is determined internally at each timestep by a two-stage algorithm:

**Stage 1 — Exploitable biomass.** For each fleet and area, raw exploitable biomass is computed as the sum over ages of abundance × weight × selectivity × retention, scaled by catchability (`q`). When `Mode = "Density"` (default), this biomass is divided by the relative area size (`RelSize`) to give biomass density; when `Mode = "Biomass"`, raw biomass is used.

**Stage 2 — Depletion-adjusted utility.** A depletion discount `h(phi) = (1 - exp(-phi)) / phi` is applied, where `phi` is the local fishing pressure (own-fleet effort × `q` × `D0` in

the area, plus a one-timestep-lagged contribution from competing fleets). This discounts utility in heavily fished areas, approximating an ideal free distribution under depletion.

**Stage 3 — Softmax targeting.** The depletion-adjusted utilities are normalised and passed through a softmax function with concentration parameter `lambda` (`Targeting`). The resulting probabilities are written to `Distribution`.

**Targeting:**

`Targeting` (lambda) is a non-negative concentration parameter controlling how strongly effort is concentrated in high-utility areas. At `lambda = 0`, effort is distributed uniformly across all accessible areas regardless of their relative utility. As `lambda` increases, effort becomes progressively more concentrated in the highest-utility area. The default value of `0.8` produces moderate concentration, broadly consistent with opportunistic targeting behaviour where fleets favour productive areas but do not exclusively fish the single best one.

Values around `0.5` approach near-uniform allocation; values above `2`–`3` produce strongly directed behaviour where most effort concentrates in the top one or two areas. The parameter is open-ended with no fixed upper bound, but very large values (e.g. `> 5`) effectively collapse all effort to the single highest-utility area in most configurations.

**Effort-to-Trips Conversion and Angler Scaling:**

Bag-limit management procedures require the effort index to be expressed as absolute angler trip counts. When `Units = "trips"` the effort array is already on the trips scale and no conversion is needed.

When the bag limit is defined per angler (the default), `AnglerPerTrip` provides the mean number of anglers per trip $A_f(t)$, such that the fleet-level retention cap is:

$$C_f^{\text{bag}}(t) = B_f \cdot A_f(t) \cdot T_f(t)$$

where $B_f$ is the bag limit in fish per angler per trip. The `Year` dimension of `AnglerPerTrip` may be length 1 if the value is assumed constant over time; it is replicated internally to match the full model time series.

When the bag limit is defined per vessel (boat limit), `AnglerPerTrip` is not used.

**Within-Trip Overdispersion: `Theta`:**

`Theta` ($\theta$) parameterises the negative binomial distribution used by bag-limit management procedures to model within-trip catch counts:

$$n_f(t) \sim \text{NegBin}(\mu_f(t),\ \theta_f(t))$$

where $\mu_f(t)$ is the mean per-trip catch at time $t$ that clears all other retention rules (size limits, discard mortality, etc.)

- i.e. what would be kept absent any bag limit. The bag limit itself then truncates $n_f(t)$ at the cap; $\mu_f(t)$ is the input to that truncation, not its output. The variance of within-trip catch (before truncation) is:

$$\text{Var}(n_f) = \mu_f + \frac{\mu_f^2}{\theta_f(t)}$$

It determines the fraction of trips that catch at or above the bag limit at any given mean catch rate, and therefore how strongly the regulation constrains total retention as stock abundance changes.

Smaller values (e.g. 0.5-2) produce high trip-to-trip variability with many zero-catch trips and occasional large catches, typical of recreational marine fisheries. Larger values approach the Poisson distribution. Only required when a bag-limit management procedure is active for this fleet.

The `Year` dimension of `Theta` may be length 1 if the value is assumed constant over time (the typical case, absent creel data suggesting otherwise); it is replicated internally to match the full model time series. A time-varying `Theta` can be supplied directly as a `Sim x Year` array, e.g. to represent a hypothesis that within-trip catch variability changes with stock depletion.

When trip-level creel data are available, `Theta` is estimated by maximum likelihood fitting of the negative binomial to observed per-trip counts. In the absence of creel data, `Theta` must be assumed

- e.g. a single plausible value, or a range sampled across simulations to test management performance against this uncertainty.

**Pass-Through Extraction:**

When `Effort` is a fleet, effort, hist, obs, or mse object, `Effort()` returns the `Effort` slot of that object rather than constructing a new one. For hist and mse objects, the `df` argument controls whether a tidy `data.frame` (`TRUE`) or the raw array (`FALSE`) is returned.

**Stock Targeting Resistance:** `StockTargetingLambda`**:**

When a fleet operates on several stock complexes, a single effort level usually cannot meet every complex's TAC at once. Effort solving therefore also adjusts each fleet's stock-targeting weights, subject to a penalty on how far the mix moves from the previous year. The weight on that penalty is derived per fleet from the historical variability of its own targeting: a fleet whose mix has shifted a lot in the past is allowed to shift it again cheaply, while a fleet with a stable record resists.

The derived weight is normalised so that, at a multiplier of `1`, shifting the targeting mix by a given proportion costs about as much as missing the TAC by the same proportion.

`StockTargetingLambda` scales that derived weight, and is specified per stock and fleet since it lives on the fleet object:

- `1` (or `NULL`) keeps the derived value: targeting shifts and TAC misses are weighted comparably.
- `0` removes the penalty, so the fleet reallocates targeting freely to hit its TACs.
- Values below `1` (say `0.1`) let targeting move more readily.
- Values above `1` hold the existing mix more strongly; by around `100` the mix is effectively fixed at the previous year's.

It has no effect on single-complex operating models, where there is no competing TAC to reconcile against.

**Attaching to a Fleet:**

An effort object can be attached to a `Fleet()` with `Effort(Fleet) <- MyEffort` and retrieved with `Effort(Fleet)`.

### Value

- `Effort()` returns an effort object. If `Effort` is a fleet, effort, hist, obs, or mse object, the `Effort` slot of that object is returned (as a `data.frame` or array for hist and mse depending on `df`).
- `Effort<-` returns `x` with the `Effort` slot replaced by `value`.
- `Distribution()`, `Targeting()`, `Maximum()`, `Mode()`, `TripsScalar()`, `AnglerPerTrip()`, `Theta()`, `StockTargetingLambda()` return the corresponding slot from the effort object `x`.
- Their replacement forms return `x` with the corresponding slot updated.

### See Also

- effort for the class definition and slot-level documentation.
- `Fleet()` for the enclosing fleet constructor.
- `GenHistEffort()` for the data frame format used to generate stochastic historical effort.
- `Catchability()` for the gear efficiency that converts effort to fishing mortality.
- `PopulateEffort()` for how effort, distribution, and targeting arrays are expanded across simulations and years.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

### Examples

```
## ---- Empty Effort object ----
# An empty Effort object is created by default when constructing a Fleet.
e <- Effort()
e

## ---- Effort from a pre-specified array ----
# A Sim x Year numeric array can be supplied directly.
# Year dimnames must match the historical years of the OM.
hist_years <- 2000:2024
nSim <- 48

effort_array <- array(
  seq(0.2, 1, length.out = length(hist_years)),
  dim = c(1, length(hist_years)),
  dimnames = list(Sim = 1, Year = hist_years)
)

e_array <- Effort(Effort = effort_array, Units = "hours")
e_array_pop <- Populate(e_array, HistYears = hist_years, nSim = nSim, nArea = 3, seed = 42)
Effort(e_array_pop)

## ---- Effort from a data frame (stochastic) ----
# A data frame of control points can be supplied to generate a stochastic
```

```
# Sim x Year effort array via GenHistEffort(). Effort is sampled uniformly
# between Lower and Upper at each control year, linearly interpolated to
# annual resolution, perturbed by lognormal process error (CV), and
# normalised to 1 in the terminal year.
effort_df <- data.frame(
  Year  = c(2000, 2008, 2016, 2024),
  Lower = c(0,    0.3,  0.7,  1),
  Upper = c(0,    0.5,  0.9,  1),
  CV    = c(0.1,  0.1,  0.1,  0.1)
)

e_df <- Effort(Effort = effort_df, Units = "hours")
e_df_pop <- Populate(e_df, HistYears = hist_years, nSim = nSim, seed = 42)
Effort(e_df_pop) |> head()

## ---- Spatial distribution across 3 areas ----
# Distribution must have dimensions Sim x Year x Area and sum to 1 over the
# Area dimension. Here effort is fixed: 50% in area 1, 30% in area 2, and
# 20% in area 3 across all simulations and years. Supplying non-NA values
# fixes those cells; the internal allocation algorithm only fills NA cells.
dist_array <- array(
  c(0.5, 0.3, 0.2),
  dim      = c(1, 1, 3),
  dimnames = list(Sim = 1, Year = hist_years[1], Area = 1:3)
)

e_dist <- Effort(Effort = effort_df, Distribution = dist_array, Units = "hours")
e_dist_pop <- Populate(e_dist, HistYears = hist_years, nSim = nSim, nArea = 3, seed = 42)
Distribution(e_dist_pop)

## ---- Targeting: concentration of effort across areas ----
# Targeting (lambda) controls how strongly effort concentrates in
# high-utility areas via a softmax function.
# lambda = 0  : uniform distribution across accessible areas
# lambda = 0.8: moderate concentration (default)
# lambda = 3  : strongly directed effort toward highest-utility area
yr1 <- array(0, dim = c(1, 1), dimnames = list(Sim = 1, Year = hist_years[1]))

e_low  <- Effort(Effort = effort_df, Targeting = yr1)          # near-uniform
e_high <- Effort(Effort = effort_df, Targeting = yr1 + 3)      # concentrated

## ---- Mode: Density vs Biomass ----
# Mode = "Density" (default) divides exploitable biomass by relative area
# size before computing utility, attracting fleets to high-concentration
# areas regardless of area size. Mode = "Biomass" uses raw exploitable
# biomass, making larger areas intrinsically more attractive.
e_density <- Effort(Effort = effort_df, Mode = "Density")
e_biomass <- Effort(Effort = effort_df, Mode = "Biomass")

## ---- Accessor and replacement functions ----
e <- Effort(Effort = effort_df, Units = "hours")

Mode(e)
```

```
Units(e)
Targeting(e)
Distribution(e)
Maximum(e)

Mode(e) <- "Biomass"

## ---- Extracting Effort from a Fleet object ----
f <- Fleet(Name = "Trawl", Effort = e)
Effort(f)
```

---

`effort-class` *The* `effort` *S4 Class*

---

## Description

Defines historical fishing effort and associated spatial structure for use in a fleet object. Objects are typically created via the `Effort()` constructor, which documents all parameters in detail.

## Slots

`Effort numeric` array, data frame, or `NULL`. Historical fishing effort. See `Effort()` for accepted formats.

`Units character` or `NULL`. Units of fishing effort (e.g., `"hours"`, `"trips"`). When `Units = "trips"` the effort array is already on the absolute trips scale and `TripsScalar()` is ignored by bag-limit management procedures. See `Effort()`.

`Distribution numeric` array or `NULL`. Fraction of total effort allocated to each spatial area (`Sim x Year x Area`). See `Effort()`.

`Targeting numeric` array or `NULL`. Fleet targeting concentration parameter (`Sim x Year`). See `Effort()`.

`Maximum numeric` array or `NULL`. Maximum allowable effort. See `Effort()`.

`Mode character` or `NULL`. Spatial utility calculation mode: `"Density"` (default) or `"Biomass"`. See `Effort()`.

`TripsScalar numeric` array or `NULL`. Time-varying scalar $\alpha_f(t)$ converting the effort index to absolute angler trip counts via $T_f(t) = \alpha_f(t) \cdot E_f(t)$ (`Sim x Year`). The `Year` dimension may be length 1 (replicated internally) or span the full model time series. Required by bag-limit management procedures when `Units != "trips"`. Ignored when `Units = "trips"`. See `Effort()`.

`AnglerPerTrip numeric` array or `NULL`. Mean number of anglers per trip (`Sim x Year`). Used by bag-limit management procedures to convert a per-angler bag limit to a fleet-level retention cap via $C_f^{\text{bag}}(t) = B_f \cdot A_f(t) \cdot T_f(t)$, where $B_f$ is the bag limit (fish per angler per trip) and $T_f(t)$ is the number of trips. See `Effort()`.

`Theta numeric` array or `NULL`. Overdispersion parameter $\theta$ of the negative binomial within-trip catch distribution (`Sim x Year`). The `Year` dimension may be length 1 (replicated internally) or span the full model time series. Used by bag-limit management procedures to model the distribution of retained catch across trips at a given mean catch rate. See `Effort()`.

`StockTargetingLambda numeric` array or `NULL`. Multiplier on this fleet's resistance to changing its stock-targeting mix (`Sim x Year`). Multi-stock TAC solving only. `1` (default) leaves the internally derived value unchanged; `0` removes the resistance; larger values make the fleet hold its existing mix more strongly. See `Effort()`.

`Misc list`. Miscellaneous additional inputs.

### See Also

`Effort()` for the constructor and full parameter documentation. fleet for the enclosing fleet object.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `fleet-class`, `retention-class`, `selectivity-class`

---

## `EffortAllocation` *Fleet Allocation of Absolute Effort for OM Objects*

---

### Description

Access or replace the `EffortAllocation` slot of an om object: the fleet-level split applied when an MP returns a single, scalar, absolute Effort recommendation (`EffType = "Abs"`) for a complex with more than one fleet. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
EffortAllocation(x)

EffortAllocation(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a named list of length 0 or the number of stock complexes, each element an `nSim` (or 1, recycled) by `nFleet` matrix with rows summing to 1. `NULL` clears it. |

## Details

A scalar `Advice@Effort` value is interpreted differently depending on `EffType`:

- `EffType = "Rel"` (the default): the scalar is a fraction of the last historical year's effort, applied per fleet to that fleet's own historical level. Each fleet's share of total effort is preserved automatically, with no allocation step needed.
- `EffType = "Abs"`: the scalar is a single total effort value to be split across fleets. `EffortAllocation` supplies that split: for a complex with `nFleet > 1`, the total is multiplied by `EffortAllocation[[complex]][sim, ]` to get each fleet's absolute effort.

`EffortAllocation` is only consulted for this scalar-plus-`"Abs"` case. It is ignored for scalar `"Rel"` effort, a length-`nFleet` effort vector (already fleet-specific), a fleet x area effort matrix (already fleet-specific), and any complex with only one fleet.

If unspecified for a complex, `EffortAllocation` falls back to the mean relative effort by fleet over the last five historical years, i.e. absolute effort is split in the same proportions the fleets have historically fished in.

## Value

- `EffortAllocation()` returns the value of the `EffortAllocation` slot.
- `EffortAllocation<-()` returns `x` with the `EffortAllocation` slot updated.

## See Also

- `OM()` for the constructor, om for the class definition.
- `Advice()` for the interpretation of scalar Effort advice.
- `FleetAllocation()` for the equivalent split applied to TAC.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
om <- OM(nSim = 10)
EffortAllocation(om) <- list(Stock1 = matrix(c(0.3, 0.7), 10, 2, byrow = TRUE))
EffortAllocation(om)
```

`EffortData` *EffortData Constructor and Accessors*

## Description

Construct an effortdata object storing observed fishing effort and uncertainty, or access and replace individual slots.

## Usage

```
EffortData(Name = NULL, Value = NULL, CV = NULL, Units = NULL)
```

## Arguments

| | |
|---|---|
| `Name` | `character` or `NULL`. Name of the dataset. Default `NULL`. |
| `Value` | `array` or `NULL`. Observed effort values with dimensions `[nYear x nFleet]`. Default `NULL`. |
| `CV` | `array` or `NULL`. Coefficients of variation for the effort observations, matching the dimensions of `Value`. Default `NULL`. |
| `Units` | `character` or `NULL`. Units of effort measurement, one element per fleet (e.g., `"days"`, `"trips"`, `"unitless"`). Default `NULL`. |

## Details

### Usage:

`EffortData()` is the standard constructor. All arguments are optional; an empty object is valid and slots are populated later (e.g., when `Simulate()` generates historical data).

### Attaching to a Data Object:

```
Effort(data) <- EffortData(Value = myArray, CV = myCVArray)
```

## Value

- `EffortData()` returns an effortdata object.
- `Effort()` returns the `Effort` slot from a data object `x`.
- `Effort<-` returns `x` with the `Effort` slot replaced.

## See Also

- effortdata for the class definition.
- data and `Data()` for the enclosing data object.
- `EffortObs()` for the observation error structure that generates effortdata objects during simulation.

Other data: `CatchData()`, `CompData()`, `IndicesData()`, `catchdata-class`, `compdata-class`, `effortdata-class`, `indicesdata-class`

### Examples

```
# Empty object
ed <- EffortData()

# With values
yrs <- 2000:2020
nFleet <- 2
val <- array(runif(length(yrs) * nFleet),
             dim = c(length(yrs), nFleet),
             dimnames = list(Year = yrs, Fleet = c("Trawl", "Longline")))
ed <- EffortData(Value = val,
                 CV    = array(0.2, dim = dim(val), dimnames = dimnames(val)),
                 Units = c("hours", "hooks"))
```

---

## effortdata-class *The* effortdata *S4 Class*

---

### Description

Stores time series observations of fishing effort and associated uncertainty. Used in the `Effort` slot of a data object. Objects are typically created via `EffortData()`, which documents all parameters in detail.

### Slots

`Name character` or `NULL`. Names of the fleets

`Value array` or `NULL`. Observed effort values with dimensions `[nYear x nFleet]`. See `EffortData()`.

`CV array` or `NULL`. Coefficients of variation matching the dimensions of `Value`. See `EffortData()`.

`Units character` or `NULL`. Units of effort measurement per fleet (e.g., `"days"`, `"trips"`, `"unitless"`). See `EffortData()`.

### See Also

`EffortData()` for the constructor and full parameter documentation. data for the enclosing data object.

Other data: `CatchData()`, `CompData()`, `EffortData()`, `IndicesData()`, `catchdata-class`, `compdata-class`, `indicesdata-class`

`EffortObs` *EffortObs Constructor and Accessors*

## Description

Construct an effortobs object defining observation error for fishing effort data, or access and replace individual slots.

## Usage

```
EffortObs(
  CV = NULL,
  Bias = NULL,
  Error = NULL,
  Years = NULL,
  Units = NULL,
  Ref = NULL,
  Misc = list()
)
```

## Arguments

| | |
|---|---|
| `CV` | `numeric` or `NULL`. Coefficient of variation of the lognormal effort observation error. Accepted forms:<br>• `NULL` (default): no observation error is generated unless `Error` is supplied directly.<br>• Length-1 scalar: a single CV applied uniformly across all simulations and years when generating the `Error` array internally.<br>• Length-2 vector `c(mean, sd)`: `CV` is itself drawn stochastically — a unique CV per simulation is generated before being used to generate the `Error` array.<br>• Length-`nSim` named array: used directly, one CV per simulation. |
| `Bias` | `numeric` or `NULL`. Multiplicative observation bias. Accepted forms:<br>• `NULL` (default): bias defaults to `1` (unbiased) internally.<br>• Positive scalar: interpreted as the CV of a lognormal distribution; a unique bias is drawn per simulation as `rlnorm(nSim, mconv(1, Bias), sdconv(1, Bias))`.<br>• Length-2 vector `c(lower, upper)` (when `nSim != 2`): treated as uniform bounds; a unique bias per simulation is drawn via `runif(nSim, lower, upper)`.<br>• Length-`nSim` vector or named array: used directly, one bias per simulation. |
| `Error` | `numeric` array or `NULL`. Pre-specified lognormal observation error multipliers with dimensions `nSim x nYear` and named dimnames (`Sim`, `Year`). When supplied, `Error` is used directly and `CV` is ignored for error generation. Default `NULL`, in which case `Error` is generated from `CV` internally. |

| | |
|---|---|
| `Years` | `numeric` vector or `NULL`. Calendar years over which the observation error applies. Default `NULL` (all historical years). |
| `Units` | `character(1)` or `NULL`. Units of effort (e.g., `"hours"`, `"trips"`, `"unitless"`). Default `NULL`. |
| `Ref` | `numeric` or `NULL`. Reference effort value. Accepted forms:<br>• `NULL` (default): no reference value.<br>• Positive scalar: interpreted as the CV of a lognormal distribution; a unique reference value is drawn per simulation as `rlnorm(nSim, mconv(1, Ref), sdconv(1, Ref))` internally.<br>• Length-`nSim` vector or named array: used directly. |
| `Misc` | `list`. Miscellaneous additional objects. Default `list()`. |

## Details

### Observation Error Generation:

The error applied to simulated effort is `Bias[sim] * Error[sim, year]`. The `Error` array (`nSim x nYear`) is obtained in one of two ways, in order of precedence:

1. **User-supplied** `Error`: stored directly; `CV` is not used.
2. **Stochastic from** `CV`: if `Error` is `NULL` and `CV` is specified, Internally the model generates a lognormal array where each cell is `rlnorm(1, mconv(1, CV[sim]), sdconv(1, CV[sim]))`. `CV` is first expanded to length `nSim` internally — a length-2 input triggers a stochastic draw of a unique CV per simulation.

**Note**: Effort observation error is not currently conditioned from observed data.

`Bias` is expanded internally: a scalar input is treated as the CV of a lognormal from which one bias per simulation is drawn; a length-`nSim` input is used directly.

`Ref` is expanded internally using the same convention as `Bias`.

Unlike catch and index observation error, effort observation error is not conditioned on real data; the user-supplied `CV`, `Bias`, and `Error` are always used as specified.

### Attaching to an Obs Object:

```
Effort(obs) <- EffortObs(CV = 0.15)
```

## Value

- `EffortObs()` returns an effortobs object.
- `Effort()` (accessor) returns the `Effort` slot from object `x`.
- `Effort<-` returns `x` with the `Effort` slot replaced.
- `CV()`, `Bias()`, `Units()`, `Years()`, `Ref()` return the corresponding slot from an effortobs object `x`.
- Their replacement forms return `x` with the corresponding slot updated.

## See Also

- effortobs for the class definition.
- `Obs()` for the enclosing observation model constructor.

Other obs: `CatchObs()`, `CompObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

## Examples

```
# Uniform 15% CV
eo <- EffortObs(CV = 0.15)

# Stochastic CV - unique CV per simulation drawn from lognormal(mean=0.15, sd=0.03)
eo <- EffortObs(CV = c(0.15, 0.03))

# Bias as a CV - unique bias per simulation
eo <- EffortObs(CV = 0.15, Bias = 0.05)

# Attach to obs
obs <- Obs(Effort = EffortObs(CV = 0.15))
```

---

# `effortobs-class` *The* `effortobs` *S4 Class*

---

## Description

Defines the observation error structure for fishing effort data. Used in the `Effort` slot of an obs object. Objects are typically created via `EffortObs()`, which documents all parameters in detail.

## Slots

`CV numeric` or `NULL`. Coefficient of variation of effort observation error. See `EffortObs()`.

`Error numeric` array or `NULL`. Realised lognormal observation error multipliers (`nSim x nYear`). Populated internally during simulation; see `EffortObs()`.

`Bias numeric` or `NULL`. Multiplicative observation bias. See `EffortObs()`.

`Years numeric` or `NULL`. Calendar years over which the observation error applies. See `EffortObs()`.

`Units character` or `NULL`. Units of fishing effort (e.g., `"hours"`, `"trips"`). See `EffortObs()`.

`Ref numeric` array or `NULL`. Reference effort values, one per simulation. See `EffortObs()`.

`Misc list`. Miscellaneous additional objects.

### See Also

`EffortObs()` for the constructor and full parameter documentation. obs for the enclosing observation model object.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

---

`Emp` *A flexible empirical management procedure.*

---

### Description

An all-purpose empirical MP that runs of Indices of relative abundance

### Usage

```
Emp(
  x,
  Data,
  reps = 1,
  Inds = NA,
  I_freq = NA,
  I_wt = NA,
  calib_yrs = 2,
  enp_mult = 0.3,
  Ind_fac = NA,
  TACrng = NA,
  delta_down = c(0.01, 0.5),
  delta_up = c(0.01, 0.5),
  resp = 1,
  curI_2_target = NA,
  HCR_CP_B = c(0, 0),
  HCR_CP_TAC = c(0, 1),
  Mode = 1
)
```

### Arguments

| | |
|---|---|
| `x` | Positive integer, the simulation number (a position in data object Data) |
| `Data` | An object of class 'Data' containing all fishery data (simulated or real - real has only one 'simulation') |
| `reps` | Positive integer, the number of stochastic samples of management advice (not applicable here) |
| `Inds` | Vector of positive integers. The indices (dimension 2) of the Additional Indices Data@AddInd to be used in calculation. When this is NA, the single index Data@Ind is used |

| | |
|---|---|
| `I_freq` | Vector of positive integers. Same length as Inds - how frequently will each index be available. 1 is every year, 2 is every 2 years, etc. |
| `I_wt` | Vector of positive real numbers. Same lengtt as Inds - the weighting of each index in the calculation of mean index level. |
| `calib_yrs` | Positive integer. The number of recent historical years used to calculate the 'current' Catch per Index value (more or less a nuisance parameter) |
| `enp_mult` | Fraction. The degree of smoothing for the polynomial function of indices. Larger numbers mean more smoothing. This is effective number of parameters. 0.3 means that the number of parameters in the polynomial smoother is 30% the length of the index. |
| `Ind_fac` | Positive real number. The factor (multiplier) of current catch(calib_yrs) / index(calib_yrs) to fish at in the future. A value of 2 means that per index the catches will be twice as high as today. If NA, the fraction of defaults to perfectly known mean((0.75 * FMSY)/last_historical_F) - mean over simulations. |
| `TACrng` | Vector 2 positions long, the minimum and maximum allowable catches. If NA this defaults to c(0, max_historical_catch*100) - essentially no TAC limit. |
| `delta_down` | Vector 2 positions long, the minimum and maximum allowable fractional downward change in TAC among management cycles. |
| `delta_up` | Vector 2 positions long, the minimum and maximum allowable fractional upward change in TAC among management cycles. |
| `resp` | Positive real number, the responsiveness of the TAC change algorithm. TAC_change = exp(log(new_TAC/old_TAC)*resp). Lower values linearly reduce the logspace TAC response and make smaller adjustements as proposed TAC changes are larger). |
| `curI_2_target` | Positive real number, the current (most recent historical year) index relative that at the target biomass level. If NA this defaults to perfectly known mean(last_historical_biomass / (1.25 * BMSY)), mean over all simulations. |
| `HCR_CP_B` | Vector of positive real numbers. Biomass control points of an HCR. These are the x-axis locations of the hockey stick inflection points. c(0,1) means a linear ramp from I/I_target. c(0.5,1) means no fishing til half I_target then a linear ramp in fishing to I_target. c(0,0) means no HCR. |
| `HCR_CP_TAC` | Vector of positive real numbers. Response control points of an HCR. These are the y-levels corresponding with the hockey stick. These are the minimum and maximum modifiers applied to the TAC recommendation. |
| `Mode` | Integer. What type of index-based MP is used? 1 = Index rate, aims to fish at a rate of index (ie TAC = f(I, current_C / current_I, Ind_fac, HCR_CP_B, HCR_CP_TAC)), 2 = Index target, makes incremental TAC adjustments based on I/I_target (i.e. TAC = f(I, curI_2_target, )) |

## Value

An object of class `MP`.

### Author(s)

T. Carruthers

---

`EmptyObject` *Test Whether an Object is Empty*

---

### Description

Returns `TRUE` if an object contains no user-supplied information — i.e. it is either a freshly constructed default object or all of its slots are `NULL`, zero-length, or entirely `NA`.

### Usage

```
EmptyObject(object)
```

### Arguments

| | |
|---|---|
| `object` | An S4 object, or any R object.<br>For S4 objects, `EmptyObject()` first checks whether the object is identical to a freshly constructed default of the same class via `isNewObject()`. If so, it returns `TRUE` immediately.<br>Otherwise it inspects every slot recursively:<br>• Slots containing functions are skipped.<br>• Slots containing nested S4 objects are tested recursively via `EmptyObject()`.<br>• All other slots are considered empty if they are `NULL`, zero-length, or entirely `NA`.<br>`EmptyObject()` returns `TRUE` only when every slot passes the emptiness check. A single non-empty slot causes the object to be considered non-empty.<br>For non-S4 objects, `EmptyObject()` returns `TRUE` if the object has length zero or consists entirely of `NA` values.<br>`EmptyObject()` is used throughout the `Populate*()` functions as an early exit guard: if an object is empty, population is skipped and the object is returned unchanged. |

### Value

A length-1 logical.

### See Also

`isNewObject()`

### Examples

```
# A freshly constructed object is empty
EmptyObject(CompObs())

# An object with any slot filled is not empty
EmptyObject(CompObs(SampleSize = 200))

# Non-S4 objects
EmptyObject(NULL)
EmptyObject(numeric(0))
EmptyObject(c(NA, NA))
```

---

## equilibrium-class *The* equilibrium *S4 Class*

---

### Description

An S4 class storing absolute-scale equilibrium quantities evaluated across a vector of apical fishing mortality values, accounting for the stock-recruitment relationship. Produced by CalcEquilibrium().

### Details

**Scaling from per-recruit to absolute:**

All F-dependent slots are obtained by multiplying the corresponding per-recruit quantity from a perrecruit object by equilibrium recruitment:

$$\texttt{Recruits}_{s,F} = R_{0,s} \cdot \phi(\texttt{SPR}_{s,F})$$

where $\phi$ is the relative recruitment function from the stock's SRR() and $R_{0,s}$ is unfished recruitment. Values of RelRecruits below zero are floored at zero before scaling.

**Complexes:**

When stocks belong to the same complex they share a single apical F, and SPR for each stock is computed relative to that complex-level F. The RelRecruits scaling is still applied per stock.

### Slots

apicalF Numeric vector of apical fishing mortality values at which equilibrium quantities were evaluated. Length nF.

SPR0 Spawning-per-recruit at unfished conditions. Array (Sim × Stock × Year).

SPR Spawning-per-recruit relative to unfished, i.e. SPRF / SPR0. Array (Sim × Stock × Year × F).

RelRecruits Relative recruitment from the stock-recruitment relationship, evaluated at each SPR value. Array (Sim × Stock × Year × F).

Recruits Absolute equilibrium recruitment (R0 × RelRecruits). Array (Sim × Stock × Year × F).

`Number` Absolute equilibrium numbers-at-age (`NPRF × Recruits`), summed over the age dimension. Array (`Sim × Stock × Year × F`).

`Biomass` Absolute equilibrium total biomass. Array (`Sim × Stock × Year × F`).

`SBiomass` Absolute equilibrium spawning biomass. Array (`Sim × Stock × Year × F`).

`SProduction` Absolute equilibrium spawning production, computed from fecundity-at-age rather than weight × maturity. Array (`Sim × Stock × Year × F`).

`Removals` Absolute equilibrium total removals (landings + dead discards), summed over fleets. Array (`Sim × Stock × Year × F`).

`Landings` Absolute equilibrium landings, summed over fleets. Array (`Sim × Stock × Year × F`).

`Misc` List reserved for internal use.

### See Also

`CalcEquilibrium()`, perrecruit

---

## `EstimateBeta` *Estimate (or Fix) the Hyperstability Parameter Beta*

---

### Description

Fits `Beta` per simulation by OLS regression of `log(Observed)` on `log(Simulated)`, or returns a supplied fixed value.

### Usage

```
EstimateBeta(
  logObs,
  logSim,
  beta = NULL,
  MinPoints = 8,
  MinLogSD = 0.1,
  alpha = 0.05,
  Bounds = c(0.1, 3)
)
```

### Arguments

| | |
|---|---|
| `logObs` | Numeric vector, length n. `log(Observed)` for the years being fit (already subset to valid/fitted years). |
| `logSim` | Numeric matrix, nSim x n. `log(Simulated)` for the same years, one row per simulation. |
| `beta` | `NULL`, numeric scalar, or length-nSim vector. When `NULL` (default), `Beta` is estimated as described above. Otherwise the supplied value(s) are used directly (recycled to nSim) and no fitting or testing is performed (`Status = "fixed_user"`). |

| | |
|---|---|
| `MinPoints` | Integer. Minimum usable years required before attempting a free fit. Default `8`. |
| `MinLogSD` | Numeric. Minimum standard deviation of `log(Simulated)` (per simulation) required before attempting a free fit – guards against an unstable slope estimate from a near-flat nominal index. Default `0.1`. |
| `alpha` | Numeric. Two-sided significance level for testing whether the fitted slope differs from `1`. Default `0.05`. |
| `Bounds` | Numeric length 2. Lower/upper clamp applied to a significant, freely estimated `Beta`. Default `c(0.1, 3)`. |

### Details

When estimating, `Beta` is fixed at `1` when there isn't enough evidence to distinguish it from `1`, or clamped to `[0.1, 3]` when the fit is significant but extreme. The `Status` element records which of these happened for each simulation, so callers can surface it rather than have `Beta` silently fall back to `1`.

### Value

A list with elements, each numeric length nSim unless noted:

- `Beta`: the fitted or supplied `Beta`.
- `SE_Beta`: standard error of the OLS slope. `NA` unless `Status == "estimated"`.
- `CI_Lower`, `CI_Upper`: two-sided `1 - alpha` confidence interval on the OLS slope (pre-clamping). `NA` unless `Status == "estimated"` or `"fixed_bounds"`.
- `R2`: fit R-squared. `NA` unless `Status == "estimated"` or `"fixed_bounds"`.
- `PValue`: two-sided p-value for `Beta == 1`. `NA` unless `Status == "estimated"` or `"fixed_bounds"`.
- `nPoints`: scalar integer, the number of years used in the fit (`n`).
- `Status`: character. One of `"estimated"`, `"fixed_user"` (`Beta` was supplied), `"fixed_insufficient_data"` (`n < MinPoints`), `"fixed_low_variance"` (nominal index too flat to fit reliably), `"fixed_not_significant"` (slope not distinguishable from `1` at `alpha`), or `"fixed_bounds"` (significant slope clamped to `Bounds`).

---

## `ExampleMPs` *Example Management Procedures*

---

### Description

Simple management procedures (MPs) designed for testing and demonstration. Each MP function accepts a data object and returns an advice object. MPs are assigned `class "mp"` so they are recognised by the MSE framework.

### Usage

```
CurrentEffort(Data)

CurrentCatch(Data)

CurrentLandings(Data)

AverageCatch(Data)

ExampleMPs()
```

### Arguments

| | |
|---|---|
| `Data` | A data object containing historical observations. |

### Details

- `CurrentEffort`: Fixes fishing effort at 1 (last historical effort) for all fleets, maintaining the spatial distribution of effort from the last historical timestep, and (in a seasonal model) each season's own share of that pattern - see *Seasonal models* below. The `Data` argument is accepted for framework compatibility but is not used.
- `CurrentCatch`: Sets the TAC for each fleet to the removals observed in the last historical year, in the same units as `Landings`. Returns an empty advice object if no landings data are available.
- `CurrentLandings`: Sets the TAC for each fleet to the landings observed in the last historical year, in the same units as `Landings`. Returns an empty advice object if no landings data are available.
- `AverageCatch`: Sets the TAC to the mean total removals (summed over fleets) across all historical years, in the same units as `Data@Landings`. Returns an empty advice object if no landings data are available.
- `ExampleMPs`: Returns a character vector naming all example MPs defined in this group, for use in batch testing or MP selection.

**Seasonal models:**

`CurrentCatch`, `CurrentLandings`, and `AverageCatch` reference a fixed historical baseline that does not depend on when the MP is called, so each carries an `EverySeason` attribute of `TRUE`, forcing the framework to call them at every timestep of the operating model regardless of `Interval`. This is required so that, in a seasonal model (`Seasons > 1`), each call can return the value for the specific season it is advising, rather than the same figure being reapplied to every season of the year.

`CurrentEffort` always returns an empty `Advice()`, which the framework's default fallback (used whenever neither `TAC` nor `Effort` is set) resolves to last year's effort, relative to the last complete historical year - separately season-matched by the framework itself so, e.g., this year's Q1 reuses last year's Q1 pattern rather than the historical period's final season. `CurrentEffort` carries the same `EverySeason` attribute as the other three for consistency, though - unlike them - its output does not actually depend on how often it is called.

**Data lag:**

`CurrentCatch`, `CurrentLandings`, and `AverageCatch` carry a `DataLag` attribute of `0`, so they always see the complete historical period regardless of the OM's `DataLag()`, and the same baseline catch applies from the first projection year onward. `CurrentEffort` uses the last historical effort directly and is likewise unaffected by `DataLag`.

## Value

- `CurrentEffort`, `CurrentCatch`, `CurrentLandings`, `AverageCatch`: an advice object; see `Advice()` for constructor details.
- `ExampleMPs`: a character vector of the example MP names.

## See Also

`Advice()`, `CheckCatch()`, data, advice

## Examples

```
## Not run:
MSE <- runMSE(SingleStockOM, MPs=ExampleMPs())

## End(Not run)
```

---

# exploitationdata-class

`exploitationdata` *Class*

---

## Description

Groups the fleet-specific exploitation process components required to describe how fishing mortality is distributed across the population. Used in the `Exploitation` slot of a data object.

## Usage

```
ExploitationData()
```

## Value

`ExploitationData()` returns a `exploitationdata` object.

### Slots

`Selectivity` An object of class selectivity defining the age- or length-based selectivity pattern for each fleet.

`Retention` An object of class retention defining the probability of retaining a fish of a given age or length once caught.

`DiscardMortality` An object of class discardmortality defining the mortality rate of discarded fish by age or length.

`Misc` A named list for any additional exploitation-level metadata.

`ExploitationData()` creates a new `exploitationdata` object.

### See Also

data, `Data()`

---

## `ExploitationObs` *ExploitationObs Constructor*

---

### Description

Construct an exploitationobs object defining observation error for exploitation process parameters.

### Usage

```
ExploitationObs(
  Selectivity = list(),
  Retention = list(),
  DiscardMortality = list(),
  Misc = list()
)
```

### Arguments

| | |
|---|---|
| `Selectivity` | `list`. Observation error specification for selectivity. Default `list()`. |
| `Retention` | `list`. Observation error specification for retention. Default `list()`. |
| `DiscardMortality` | `list`. Observation error specification for discard mortality. Default `list()`. |
| `Misc` | `list`. Miscellaneous additional objects. Default `list()`. |

### Details

**Placeholder.** This class and constructor are reserved for future development and are not currently used during model runs. Each slot will eventually mirror the structure of the corresponding slot in exploitationdata, providing bias and CV specifications for selectivity, retention, and discard mortality estimation. All slot values are silently ignored during simulation.

When converted from a legacy Obs-legacy object via `ConvertObs()`, the selectivity bias CVs (`LFCbiascv`, `LFSbiascv`) are not yet mapped to this class and are silently dropped.

### Value

An exploitationobs object.

### See Also

- exploitationobs for the class definition.
- `Obs()` for the enclosing observation model constructor.
- exploitationdata for the complementary observed-values class.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

### Examples

```
eo <- ExploitationObs()
```

---

## exploitationobs-class

*The* `exploitationobs` *S4 Class*

---

### Description

Defines the observation error structure for exploitation process parameters (selectivity, retention, discard mortality). Used in the `Exploitation` slot of an obs object. Objects are created via `ExploitationObs()`.

### Details

This class is a placeholder. Each slot mirrors the corresponding sub-object of exploitationdata and will eventually hold an error structure for the associated exploitation parameter. Slots are currently untyped lists and are not populated during model runs. See `ExploitationObs()` for further details.

### Slots

`Selectivity list`. Observation error specification for selectivity. See `ExploitationObs()`.

`Retention list`. Observation error specification for retention. See `ExploitationObs()`.

`DiscardMortality list`. Observation error specification for discard mortality. See `ExploitationObs()`.

`Misc list`. Miscellaneous additional objects.

### See Also

- `ExploitationObs()` for the constructor.
- obs for the enclosing object.
- exploitationdata for the complementary observed-values class.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

---

## `Extend` *Extend an Array Along Named Dimensions*

---

### Description

Extends an array with named dimensions to include all simulations, ages, size classes, areas, and any missing years. Can also recurse into S4 objects and lists, or broadcast a tidy `data.frame`'s collapsed `Sim` rows.

### Usage

```
Extend(
  array,
  nSim = NULL,
  AgeClasses = NULL,
  Classes = NULL,
  Years = NULL,
  Areas = NULL,
  default = NULL,
  backfill = FALSE,
  maintain_seasonal_pattern = TRUE,
  skip_data = TRUE,
  debug = FALSE
)

ExtendSims(array, nSim = NULL)

ExtendAges(array, AgeClasses = NULL)

ExtendClasses(array, Classes = NULL)
```

```
ExtendYears(
  array,
  Years = NULL,
  default = NULL,
  backfill = FALSE,
  maintain_seasonal_pattern = TRUE
)

ExtendAreas(array, Areas = NULL)
```

## Arguments

| | |
|---|---|
| `array` | An `array()` with named dimensions including at least one of `Sim`, `Year`, `Age`, or `Area`. Alternatively, an S4 or `list()` object, in which case the function recurses over all slots or elements and extends any named arrays; or a tidy `data.frame` with a `Sim` column, in which case it is passed to `ExtendSims()`. Any other object is returned unchanged. |
| `nSim` | Integer. Total number of simulations, or `NULL` to skip. |
| `AgeClasses` | Numeric vector of age classes, or `NULL` to skip. |
| `Classes` | Numeric vector of size classes (length or weight bins), or `NULL` to skip. |
| `Years` | Numeric vector of `Year` values to extend to, or `NULL` to skip. |
| `Areas` | Numeric vector of area indices (`1:nArea`), or `NULL` to skip. |
| `default` | Default fill value for forward-filled years. If `NULL` (default), the value from the most recent existing year is used. |
| `backfill` | Logical. If `TRUE`, also back-fill years earlier than the earliest existing year. Default `FALSE`. |
| `maintain_seasonal_pattern` | Logical. Seasonal models only. If `TRUE` fills in missing year values by matching those from the closest corresponding season (e.g., maintains seasonal recruitment pattern) |
| `skip_data` | `logical(1)` Skip any data objects? Default `TRUE` |
| `debug` | Logical. If `TRUE`, prints the class of each object as it is processed. Default `FALSE`. |

## Details

### ExtendSims:

The `Sim` dimension must have length `1` or `nSim`. If length `1`, it is replicated to length `nSim` with simulation 1 values copied to all simulations. If already length `nSim`, the array is returned unchanged.

### ExtendAges:

The `Age` dimension must have length `1` or `nAge`. If length `1`, it is replicated to length `nAge` with the first age class copied to all ages. If already length `nAge`, the array is returned unchanged.

**ExtendClasses:**

The `Class` dimension must have length `1` or `nClass`. If length `1`, it is replicated to length `nClass` with class 1 values copied to all classes. If already length `nClass`, the array is returned unchanged.

**ExtendYears:**

The `Year` dimension can be any length. If all values in `Years` are already present, the array is returned unchanged. Otherwise missing years are added by forward-filling from the most recent existing year, back-filling from the earliest (if `backfill = TRUE`), or step-filling for years within the existing range.

**ExtendAreas:**

The `Area` dimension must have length `1` or `nArea`. If length `1`, it is replicated to length `nArea` with area 1 values copied to all areas. If already length `nArea`, the array is returned unchanged.

`ExtendSims()` also accepts a tidy `data.frame` (e.g. from `SBiomass(df = TRUE)` or similar accessors) with a `Sim` column.

## Value

The input object with dimensions extended as specified. The return type matches the input type (array, S4, or list).

## See Also

`ExtendSims()`, `ExtendAges()`, `ExtendClasses()`, `ExtendYears()`, `ExtendAreas()`

## Examples

```
# ExtendSims
array <- array(c(0.5, 0.1),
               dim = c(1,2),
               dimnames = list(Sim = 1, Year=2025:2026)
)
Extend(array, nSim=5)
ExtendSims(array, 5)

# ExtendAges
array <- array(c(0.5, 0.1),
               dim = c(2,1,2),
               dimnames = list(Sim = 1:2, Age=0, Year=2025:2026)
)
Extend(array, AgeClasses = 0:10)
ExtendAges(array, 0:10)


# ExtendYears

## Annual
array <- array(1:8,
```

```
  dim = c(2, 4),
  dimnames = list(
    Sim = 1:2,
    Year = c(1950, 1960, 1980, 2020)
  )
)
Years <- c(1940, 1955, 1965, 1985, 2025)
Extend(array, Years=Years)
ExtendYears(array, Years)

## Seasonal
array <- array(1:8,
  dim = c(2, 4),
  dimnames = list(
    Sim = 1:2,
    Year = c(1950, 1950.25, 1950.5, 1950.75)
  )
)
Years <- c(1945.25, 1960, 1975.5)
ExtendYears(array, Years)
```

---

`extract-unfished` *Unfished Reference Point Quantities*

---

## Description

Extract unfished reference point quantities from a hist object. `B0()` returns total biomass, `SB0()` spawning biomass, `SP0()` spawning production, and `N0()` numbers-at-age. All functions support both equilibrium and dynamic unfished calculations (see `type` argument).

## Usage

```
B0(Hist, type = c("Equilibrium", "Dynamic"), Reduce = TRUE)

SB0(Hist, type = c("Equilibrium", "Dynamic"), Reduce = TRUE)

SP0(Hist, type = c("Equilibrium", "Dynamic"), Reduce = TRUE)

N0(Hist, type = c("Equilibrium", "Dynamic"), Reduce = TRUE)
```

## Arguments

| | |
|---|---|
| `Hist` | A hist object; the output of `Simulate()`, containing the simulated historical fishery dynamics. |
| `type` | Character. One of `'Equilibrium'` or `'Dynamic'`; partial matching is supported. |

- `'Equilibrium'`: unfished values assuming equilibrium conditions — no recruitment deviations and all life-history parameters constant over time.
- `'Dynamic'`: unfished values calculated by running the full population dynamics model, including recruitment deviations and any time-varying parameters.

`Reduce` Logical. If `TRUE` (default), calls `ReduceDims()` to collapse any `Sim`, `Year`, or `Age` dimensions where values are identical across that dimension, reducing array size. If `FALSE`, the slot contents are returned as-is with all dimensions intact.

### Value

- `N0()`: a named list with one element per Stock. Each element is an array with dimensions `[Sim, Age, Year]`. The `Sim` and `Year` dimensions are collapsed by `ReduceDims()` when `Reduce = TRUE` and values do not vary across those dimensions.
- `B0()`, `SB0()`, `SP0()`: an array with dimensions `[Sim, Stock, Year]`. The `Sim` and `Year` dimensions are collapsed by `ReduceDims()` when `Reduce = TRUE` and values do not vary across those dimensions.

### See Also

`ReduceDims()` for details on dimension collapsing.

---

## `Fease` *MP feasibility diagnostic*

---

### Description

What MPs may be run (best case scenario) for various data-availability scenarios and management constraints?

### Usage

```
Fease(
  Data = NULL,
  TAC = TRUE,
  TAE = TRUE,
  SL = TRUE,
  Spatial = TRUE,
  names.only = TRUE,
  msg = TRUE,
  include.ref = FALSE
)
```

### Arguments

| | |
|---|---|
| `Data` | An object of class 'Data'. Optional. If Data object is included, the returned MPs are both feasible (in terms of management) and possible (sufficient data to run MP) |
| `TAC` | Logical. Are catch limits feasible for this fishery? |
| `TAE` | Logical. Are effort controls feasible for this fishery? |
| `SL` | Logical. Are size-selectivity regulations (either gear changes or size-retention regulations) feasible for this fishery? |
| `Spatial` | Logical. Are spatial closures feasible for this fishery? |
| `names.only` | Logical. Should only the names of the feasible MPs be returned (default)? If FALSE, a data frame with MP name, and two columns of logical values: Can (possible given data) and Fease (feasible given management constraints) is returned |
| `msg` | Logical. Should messages be printed to the console? |
| `include.ref` | Logical. Should reference MPs (e.g. FMSYref) be included as feasible methods? Default is FALSE |

### Value

Either a vector of MP names that are feasible for the fishery (default) or a 3 column data frame (`names.only=FALSE`).

### Author(s)

T. Carruthers & A. Hordyk

### Examples

```
## Not run:
Fease(TAC=FALSE)
Fease(SL=FALSE, Spatial=FALSE)
Fease(Atlantic_mackerel, TAE=FALSE, names.only=FALSE)

## End(Not run)
```

---

## `Fecundity` *Fecundity Constructor and Accessors*

---

### Description

Construct a fecundity object defining the fecundity schedule for a stock, or access and replace the `Fecundity` slot of a stock and its individual slots. `Fecundity` is optional — see *Default Behaviour* below.

### Usage

```
Fecundity(
  Pars = list(),
  Model = NULL,
  Units = "eggs",
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  Classes = NULL,
  Timing = NULL,
  Misc = list()
)

Fecundity(x) <- value
```

### Arguments

| | |
|---|---|
| `Pars` | `list`. Named list of fecundity parameters. Element names must match the arguments of a built-in fecundity model (see `FecundityModels()`). When `Pars` is non-empty and `Model` is `NULL`, the model is inferred automatically from the parameter names. When `Pars` is a stock object, `Fecundity()` acts as a pass-through accessor and returns `x@Fecundity`. See also Specifying Biological and Fleet Schedules for the full set of accepted input formats. Default `list()`. |
| `Model` | `character(1)` or `function`. Fecundity model identifier. When `NULL` (default), the model is inferred from `Pars`. May be set to a character string naming a built-in model or to a custom R function — see Specifying Biological and Fleet Schedules. |
| `Units` | `character(1)`. Unit of reproductive output (e.g., `"eggs"`). Sets the unit of the `SProduction` slot in hist output objects. Default `"eggs"`. |
| `MeanAtAge` | `array` or `NULL`. Mean fecundity-at-age with named dimensions `Sim × Age × Year`. Supply directly when bypassing the model-based approach (`Pars = list()`). Only the years at which values *change* need to be included; `Extend()` fills the remainder. When `Pars` contains a matched model, any values here are **overwritten** during `Populate()`. A numeric vector of length `nAge` is also accepted and converted to a `1 × nAge × 1` array automatically. Default `NULL`. |
| `MeanAtLength` | `array` or `NULL`. Mean fecundity-at-length with named dimensions `Sim × Class × Year`. Populated automatically during `Populate()` when an at-length fecundity model is used; may also be supplied directly. When `MeanAtLength` is populated and `MeanAtAge` is not, `Populate()` converts it to `MeanAtAge` via the age-length key (`ALK`) from a length object. A numeric vector is also accepted and converted to a `1 × nClass × 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`. Default `NULL`. |
| `Classes` | `numeric` or `NULL`. Age or length class midpoints corresponding to the `MeanAt*` array in use. Default `NULL`. |

| | |
|---|---|
| `Timing` | `array` or `NULL`. Reserved for future use. Intended to represent the timing of spawning within the time step. Currently stored but not applied; spawning timing is controlled by `SRR()`. Default `NULL`. |
| `Misc` | `list`. Used internally. Default `list()`. |
| `x` | A fecundity object for slot accessors, or a stock object for `Fecundity<-`. |
| `value` | For `Fecundity<-`: a fecundity object. |

## Details

### Default Behaviour When Omitted:

`Fecundity` is optional. When this object is empty (the default), `Populate()` computes spawning production (`SProduction` in hist output) as mature weight-at-age — the product of `Weight@MeanAtAge` and `Maturity@MeanAtAge`. In this case `SProduction` equals spawning biomass (`SBiomass`) and both slots carry the same values.

When this object is populated, `SProduction` is computed in the units of `Units` (e.g. eggs per individual multiplied through maturity), allowing reproductive output to differ from biomass. This is relevant for species where egg production is not proportional to body weight — for example, batch-spawning fish or species with strongly size-dependent clutch size.

### Specifying Fecundity:

There are two ways to define the fecundity schedule; see Specifying Biological and Fleet Schedules for full details on input formats and the rules that govern how `Pars`, `Model`, and `MeanAt*` arrays interact.

**Model-based** (recommended): supply `Pars` with named parameters matching a built-in model (see `FecundityModels()`). If `Model = NULL` and the parameter names uniquely match a model, the model is resolved automatically. `MeanAtAge` is then populated by `Populate()` when the stock is added to an `OM()`:

```
# Egg production proportional to weight - model inferred from Pars names
fec <- Fecundity(Pars = list(a = 1000, b = 3.0))

# Stochastic a across simulations
fec <- Fecundity(Pars = list(a = c(800, 1200), b = 3.0))

# Time-varying a (increases in 2010; Extend() fills the rest)
a_arr <- array(c(1000, 1200), dim = c(1, 2),
               dimnames = list(Sim = 1, Year = c(1990, 2010)))
fec <- Fecundity(Pars = list(a = a_arr, b = 3.0))

# Inter-annual random walk on a (log-normal, SD = 0.1)
fec <- Fecundity(Pars = list(a = c(800, 1200), aSD = 0.1, b = 3.0))
```

**Direct array**: supply `MeanAtAge` (or `MeanAtLength`) with `Pars = list()`. Values are preserved during `Populate()`:

```
ages  <- 0:20
fec_aa <- Fecundity(
  MeanAtAge = array(
```

```
    1000 * seq(0, 1, length.out = length(ages))^2,
    dim      = c(1, length(ages), 1),
    dimnames = list(Sim = 1, Age = ages, Year = 1990)
  )
)
```

**At-Length Models:**

Models whose class contains `"at-Length"` produce `MeanAtLength`, which is converted to `MeanAtAge` via the `ALK` — a populated `Length()` object must be passed to `Populate()`.

**Pass-Through Access from a Stock:**

When `Pars` is a stock object, `Fecundity()` returns the `Fecundity` slot directly:

```
Fecundity(my_stock)             # returns my_stock@Fecundity
Fecundity(my_stock) <- my_fec   # replaces my_stock@Fecundity
```

**Slot Accessors:**

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Pars(fec)          <- list(a = 1000, b = 3.0)
Model(fec)         <- "EggProduction"
Units(fec)         <- "eggs"
MeanAtAge(fec)     <- my_array
MeanAtLength(fec)  <- my_length_array
Classes(fec)       <- 0:20
```

## Value

- `Fecundity()` returns a fecundity object. If `Pars` is a stock, returns `x@Fecundity`.
- `Fecundity<-` returns the stock `x` with the `Fecundity` slot replaced and the object re-validated.

## See Also

- fecundity for the class definition and slot-level documentation.
- `FecundityModels()` for available models and required parameter sets.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.
- `Length()` for the companion length schedule, required when using at-length fecundity models.
- `Weight()` and `Maturity()` for the schedules used in the default spawning biomass calculation.
- `SRR()` for spawning timing within the time step.

Other fecundity: `fecundity-class`

### Examples

```
# See FecundityModels() for all available built-in fecundity models and the
# parameter names required for each. All models use a logistic 50/95
# parameterisation with an asymptote of MaxFec.
FecundityModels()

# ---- Default behaviour (Fecundity omitted) ----

## When Fecundity is left empty, Populate() computes SProduction as mature
## weight-at-age (Weight@MeanAtAge * Maturity@MeanAtAge), so SProduction
## equals SBiomass. This is appropriate for most species.
stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
Fecundity(stk)    # empty fecundity-class object

# ---- Model-based specification - age-based ----

## FecundityAtAge: logistic fecundity-at-age
## A50     = age at 50% maximum fecundity
## A50_95  = age interval from 50% to 95% maximum fecundity
## MaxFec  = asymptotic (maximum) fecundity

## Constant parameters - same value for every simulation and year
fec <- Fecundity(Pars = list(A50 = 4, A50_95 = 2, MaxFec = 1e6))

## Stochastic A50 across simulations - drawn from Uniform(lower, upper)
fec <- Fecundity(Pars = list(A50    = c(3, 5),
                             A50_95 = 2,
                             MaxFec = 1e6))


## Time-varying MaxFec with named change-point arrays
## Only the years where values change need to be supplied; Extend() fills
## all intermediate and future years by forward-filling automatically.
nSim <- 48

mf_arr <- array(
  c(rep(1e6, nSim), rep(1.2e6, nSim)),   # MaxFec increases in 2010
  dim      = c(nSim, 2),
  dimnames = list(Sim = seq_len(nSim), Year = c(1990, 2010))
)
fec <- Fecundity(Pars = list(A50    = c(3, 5),
                             A50_95 = 2,
                             MaxFec = mf_arr))


# ---- Model-based specification - length-based ----

## FecundityAtLength: logistic fecundity-at-length
## L50     = length at 50% maximum fecundity
## L50_95  = length interval from 50% to 95% maximum fecundity
fec_l <- Fecundity(Pars = list(L50    = 40,
                               L50_95 = 8,
```

```
                                MaxFec = 1e6))

# ---- Model-based specification - weight-based ----

## FecundityAtWeight: logistic fecundity-at-weight
## W50     = weight at 50% maximum fecundity
## W50_95  = weight interval from 50% to 95% maximum fecundity
fec_w <- Fecundity(Pars = list(W50    = 2.0,
                                W50_95 = 0.5,
                                MaxFec = 1e6))

# ---- Direct array specification ----

## Supply MeanAtAge directly (Pars left empty).
## A plain numeric vector of length nAge is accepted; treated as a single
## simulation, single year and replicated by Extend() as needed.
ages   <- 0:20
fec_aa <- Fecundity(
  MeanAtAge = logistic_50_95(ages, x50 = 4, x50_95 = 2, asymp = 1e6)
)


## Named Sim x Age x Year array - two change-point years, single simulation.
## Extend() replicates Sim = 1 to all nSim simulations automatically.
maa <- array(
  c(logistic_50_95(ages, x50 = 4,   x50_95 = 2, asymp = 1e6),  # 1990
    logistic_50_95(ages, x50 = 4.5, x50_95 = 2, asymp = 1.2e6)),# 2010
  dim      = c(1, length(ages), 2),
  dimnames = list(Sim = 1, Age = ages, Year = c(1990, 2010))
)
fec_arr <- Fecundity(MeanAtAge = maa)

## Supply MeanAtLength directly - converted to MeanAtAge via the ALK
## during Populate() when a Length object is provided.
lens   <- seq(5, 120, by = 5)
fec_al <- Fecundity(
  MeanAtLength = array(
    logistic_50_95(lens, x50 = 40, x50_95 = 8, asymp = 1e6),
    dim      = c(1, length(lens), 1),
    dimnames = list(Sim = 1, Length = lens, Year = 1990)
  )
)

# ---- Slot accessors ----

fec <- Fecundity(Pars = list(L50 = 40, L50_95 = 8, MaxFec = 1e6))

## Read slots
MeanAtAge(fec)     # NULL until Populate() is called
MeanAtLength(fec)  # NULL unless an at-length model or direct array is used
Units(fec)

## Replace slots
```

```
Units(fec)         <- "eggs"
Classes(fec)       <- seq(5, 120, by = 5)

# ---- Attaching to a Stock ----

Fecundity(stk) <- Fecundity(Pars = list(L50 = 40, L50_95 = 8, MaxFec = 1e6))
Fecundity(stk)

# ---- Populate ----

## The final populated object (typically done internally).
## When Fecundity is populated, SProduction differs from SBiomass.
## At-length models require a populated Length object passed to Populate().
pop_fec <- Populate(Fecundity(stk),
                    Ages   = Ages(MaxAge = 20),
                    Length = Length(Pars = list(Linf = 100, K = 0.2, t0 = -0.1))
)
pop_fec

MeanAtLength(pop_fec)
MeanAtAge(pop_fec)
```

---

`fecundity-class` *The* `fecundity` *S4 Class*

---

## Description

Defines the fecundity schedule for a stock object, expressed as reproductive output per individual as a function of age or length. This object is optional — see *Default Behaviour* in `Fecundity()` for what happens when it is omitted. Objects are typically created via `Fecundity()`, which documents all parameters, validates inputs, and infers the fecundity model automatically when possible.

## Details

Direct construction via `methods::new()` is not recommended; use `Fecundity()` instead, which handles model inference and object validation.

This object is optional. When empty, `Populate()` computes spawning production (`SProduction`) as mature weight-at-age (i.e. identical to spawning biomass, `SBiomass`). When populated, `SProduction` is in the units of this object (e.g. eggs per individual), allowing reproductive output to differ from biomass — relevant for species where egg production is not proportional to weight.

## Slots

`Pars list`. Named list of fecundity model parameters whose names correspond to those expected by `Model`. See `FecundityModels()` for parameter sets and Specifying Biological and Fleet Schedules for accepted input formats.

`Model function` or `character(1)`. Fecundity model identifier, matched to one of `FecundityModels()`. Set automatically from the parameter names when `Pars` is supplied without an explicit model.

`Units character(1)`. Unit of reproductive output (e.g., `"eggs"`). Determines the unit of the `SProduction` slot in hist objects. When this object is empty, `SProduction` defaults to spawning biomass and `Units` is irrelevant.

`MeanAtAge array`. Mean fecundity-at-age with named dimensions `Sim`, `Age`, and `Year`. Populated automatically by `Populate()` when `Pars` and `Model` are set; may also be supplied directly when `Pars` is empty. When `Pars` contains a matched model, any existing values are overwritten. See Specifying Biological and Fleet Schedules.

`MeanAtLength array`. Mean fecundity-at-length with named dimensions `Sim`, `Length`, and `Year`. Populated automatically when an at-length fecundity model is used, or may be supplied directly. Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). Converted to `MeanAtAge` via the `ALK` during `Populate()` when `MeanAtAge` is not already populated. See Specifying Biological and Fleet Schedules.

`Classes numeric`. Age classes (in years) or lower bounds of length bins, corresponding to the `MeanAt*` array in use. For length bins, bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength` are evaluated at bin midpoints, not at these lower bounds.

`Timing array` or `numeric`. Reserved for future use. Intended to represent the timing of spawning within the time step. Currently stored but not applied; spawning timing is controlled by `SRR()`.

`Misc list`. Used internally.

## See Also

- `Fecundity()` for the constructor and accessor functions.
- `FecundityModels()` for available models and their required parameters.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `Length()` and `Weight()` for the companion schedules used in the default spawning biomass calculation and at-length fecundity models.
- `Maturity()` for the maturity schedule multiplied through fecundity during population.
- `SRR()` for spawning timing within the time step.
- Specifying Biological and Fleet Schedules for the full description of how `Pars`, `Model`, and `MeanAt*` arrays interact.

Other fecundity: `Fecundity()`

`Fecundity-Models` *Fecundity Models*

## Description

Logistic fecundity-at-age, fecundity-at-length, and fecundity-at-weight models for use in a `Fecundity()` object.

## Usage

```
FecundityAtAge(Ages, A50, A50_95, MaxFec)

FecundityAtLength(Length, L50, L50_95, MaxFec)

FecundityAtWeight(Weight, W50, W50_95, MaxFec)

FecundityModels(full = TRUE, print = TRUE)

FecundityModelsLength(full = TRUE, print = TRUE)

FecundityModelsAge(full = TRUE, print = TRUE)

FecundityModelsWeight(full = TRUE, print = TRUE)
```

## Arguments

| | |
|---|---|
| `Ages` | numeric vector of age classes (for age-based models) |
| `A50` | age at 50% maximum fecundity (Logistic-at-Age) |
| `A50_95` | interval between `A50` and age at 95% maximum fecundity (Logistic-at-Age) |
| `MaxFec` | Numeric. Maximum (asymptotic) fecundity. Used as the upper asymptote of the logistic curve. |
| `Length` | numeric vector of length classes (for length-based models) |
| `L50` | length at 50% maximum fecundity (Logistic-at-Length) |
| `L50_95` | interval between `L50` length at 95% maximum fecundity (Logistic-at-Length) |
| `Weight` | numeric vector of weight classes (for weight-based models) |
| `W50` | weight at 50% maximum fecundity (Logistic-at-Weight) |
| `W50_95` | weight at 95% maximum fecundity (Logistic-at-Weight) |
| `full` | Logical. Provide a complete table (TRUE) or just the model names (FALSE)? |
| `print` | Logical. Print out the results (TRUE) or just return the data.frame (FALSE)? |

### Details

All fecundity models use a logistic 50/95 parameterisation:

$$F(x) = \frac{\texttt{MaxFec}}{1 + \exp\left(-\ln(19) \cdot \frac{x - x_{50}}{x_{95} - x_{50}}\right)}$$

where $x_{95} = x_{50} + x_{50\_95}$.

The available model functions are:

- `FecundityAtAge()`: logistic fecundity as a function of age.
- `FecundityAtLength()`: logistic fecundity as a function of length.
- `FecundityAtWeight()`: logistic fecundity as a function of weight.

The `FecundityModels*` functions list available models:

- `FecundityModels()`: all fecundity models.
- `FecundityModelsAge()`: fecundity-at-age models only.
- `FecundityModelsLength()`: fecundity-at-length models only.
- `FecundityModelsWeight()`: fecundity-at-weight models only.

### Value

- `FecundityAtAge()`, `FecundityAtLength()`, `FecundityAtWeight()`: a numeric vector of fecundity values at each age, length, or weight class respectively.
- `FecundityModels()`, `FecundityModelsAge()`, `FecundityModelsLength()`, `FecundityModelsWeight()`: invisibly returns a data frame (if `full = TRUE`) or character vector (if `full = FALSE`) of available models. Prints to console if `print = TRUE`.

Prints to console and invisible data.frame or model names

### See Also

`Fecundity()`, `Maturity()`, `Ages()`, `Length()`, `Weight()`, `Stock()`

### Examples

```
Ages <- 0:15
Length <- seq(10, 100, 10)
Weight <- seq(1, 50, 5)

F_age <- FecundityAtAge(Ages, A50 = 5, A50_95 = 3, MaxFec=1e6)

F_length <- FecundityAtLength(Length, L50 = 50, L50_95 = 20, MaxFec=1e6)

F_weight <- FecundityAtWeight(Weight, W50 = 20, W50_95 = 15, MaxFec=1e6)


plot(Ages, F_age, type = "l", lwd = 2,
     xlab = "Age (years)", ylab = "Fecundity",
```

```
     main = "Fecundity-at-Age")

plot(Length, F_length, type = "l", lwd = 2,
     xlab = "Length", ylab = "Fecundity",
     main = "Fecundity-at-Length")

plot(Weight, F_weight, type = "l", lwd = 2,
     xlab = "Weight", ylab = "Fecundity",
     main = "Fecundity-at-Weight")
```

---

`FilterTAC` *Filter a TAC Recommendation for* `data`*/*`mp`*-Style Management Procedures*

---

## Description

Replaces negative or non-finite TAC values with `NA`. Intended as a final robustness check on TAC values proposed inside a management procedure (see `Advice()`) before they are returned. Each element of `TAC` (e.g. one per fleet) is checked independently.

## Usage

```
FilterTAC(TAC)
```

## Arguments

| | |
|---|---|
| `TAC` | A numeric vector of TAC recommendations (e.g. one value, or one per fleet). |

## Value

A numeric vector the same length as `TAC`, with negative or non-finite values replaced by `NA`.

## See Also

`IndexRate()`, `Advice()`

## Examples

```
FilterTAC(c(120, -5, NA, Inf, 340))
```

FInteract *Extract Fishing Mortality Time Series*

## Description

FInteract(), FDead(), and FRetain() extract fishing mortality for interactions (encounters), killed, and retained fish respectively, from a hist or mse object.

## Usage

```
FInteract(
  object,
  df = TRUE,
  byAge = TRUE,
  byArea = FALSE,
  byFleet = FALSE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

FDead(
  object,
  df = TRUE,
  byAge = TRUE,
  byArea = FALSE,
  byFleet = FALSE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

FRetain(
  object,
  df = TRUE,
  byAge = TRUE,
  byArea = FALSE,
  byFleet = FALSE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)
```

### Arguments

| | |
|---|---|
| `object` | A hist or mse object. |
| `df` | Logical. If `FALSE` the raw array slot is returned. If `TRUE` (default) a tidy `data.frame` is returned. |
| `byAge` | Logical. If `TRUE` the data frame retains the `Age` dimension (sourced from the `*Area` slots, which always include `Area`). If `FALSE` the apical fishing mortality — the maximum over ages — is returned. Ignored when `df = FALSE`. |
| `byArea` | Logical. If `TRUE` the `Area` dimension is retained. When `byAge = TRUE` area is always included regardless of this argument. Ignored when `df = FALSE`. |
| `byFleet` | Logical. If `TRUE` the `Fleet` dimension is retained. If `FALSE` values are summed over fleets before the apical or age-structured extraction. Ignored when `df = FALSE`. |
| `Reduce` | Logical. If `TRUE` (default) simulation dimensions are reduced using `ReduceDims()` before conversion to a data frame. |
| `IncYear` | Logical. Passed to `ReduceDims()`; controls whether the year dimension is retained during reduction. Default `FALSE`. |
| `Extend` | Logical. If `TRUE`, rows sharing one value across every simulation (e.g. the historical period, or any deterministic quantity – see `Reduce`/`ReduceDims()`) are broadcast to all `nSim` simulations via `ExtendSims()`, rather than appearing once with `Sim = 1`. Default `FALSE`. Ignored when `df = FALSE`. |
| `silent` | Logical. If `FALSE` and `Extend = FALSE`, emits a message when the returned data frame has rows sharing one value across every simulation, pointing at `Extend = TRUE`. Default `TRUE`. Ignored when `df = FALSE`. |

### Details

When `byAge = FALSE` the apical (maximum over ages) fishing mortality rate is returned. When `byAge = TRUE` the full age-structured array is retained; `byArea` is always included when `byAge = TRUE`.

When applied to an mse object the historical and projection periods are row-bound and labelled via the `Period` column; historical rows carry `MP = "Historical"`.

### Value

- `df = FALSE` — the raw array slot (`FInteract`, `FDead`, or `FRetain`).
- `df = TRUE` — a tidy `data.frame` with columns `Sim`, `Stock`, `Year`, `Period`, `MP` (MSE only), and optionally `Age`, `Area`, and/or `Fleet`, plus `Value` and `Variable`.

### See Also

`ExtendSims()`

### Examples

```
Hist <- Simulate(SingleStockOM)
MSE <- Project(Hist, 'CurrentEffort')

# Raw array slots
FInteract(Hist)
FDead(MSE)
FRetain(MSE)

# Age-structured F, as a tidy data frame (area always included)
FInteract(Hist, df = TRUE)
FDead(MSE, df = TRUE)

# Apical F (max over ages)
FDead(MSE, df = TRUE, byAge = FALSE)

# Retain fleet and area dimensions
FDead(MSE, df = TRUE, byFleet = TRUE)
FDead(MSE, df = TRUE, byAge = FALSE, byArea = TRUE, byFleet = TRUE)
```

---

## FitMovement *Fit a Movement Matrix to Target Spatial Parameters*

---

### Description

Fits a row-stochastic movement matrix for a single simulation, age class, and year by numerical optimisation, so that the asymptotic (equilibrium) unfished distribution and the diagonal (staying probability) simultaneously reproduce the targets in `Spatial@UnfishedDist` and `Spatial@ProbStaying`. Separate optimisation routines are used for two-area models and for models with three or more areas.

### Usage

```
FitMovement(Spatial, sim = 1, age = 1, year = 1)
```

### Arguments

| | |
|---|---|
| `Spatial` | A populated spatial object whose `UnfishedDist`, `ProbStaying`, and (for multi-area models) `FracOther` slots have been expanded to named arrays. |
| `sim` | `integer(1)`. Simulation index. Clamped to the `Sim` dimension of each slot if that dimension has length 1. Default `1`. |
| `age` | `integer(1)`. Age index (position in the `Age` dimension). Clamped to the `Age` dimension if length 1. Default `1`. |
| `year` | `integer(1)`. Year index (position in the `Year` dimension). Clamped to the `Year` dimension if length 1. Default `1`. |

### Details

This function is called internally by `Populate()` for every combination of simulation, age, and year. It can also be called directly to inspect or debug the fitted movement matrix for a specific slice of a spatial object.

### Value

A numeric matrix of dimensions `nArea × nArea`. Each row gives the movement probabilities from that area to all areas (including itself), and rows sum to exactly 1.

### See Also

- `Spatial()` for the constructor and full spatial specification.
- `Populate()` which calls this function internally.

---

## FitSP *Fit a Surplus Production Model to Catch and Index Data*

---

### Description

Fits a Pella-Tomlinson surplus production model to the catch and one or more relative abundance indices in a data object, and returns estimates of stock status and MSY-based reference points. The fitted model is used by the `SurplusProduction()` management procedure, and `FitSP()` can also be used directly on real or simulated `Data` objects.

### Usage

```
FitSP(
  Data,
  Indices = NULL,
  IndexSource = "auto",
  IndexFreq = NULL,
  IndexWeight = NULL,
  IndexSeasons = NULL,
  IndexUnits = c("Biomass", "Number"),
  CatchType = c("Removals", "Landings"),
  FitYears = NULL,
  Model = c("internal", "spict"),
  Shape = 2,
  EstShape = FALSE,
  ShapeCV = 0.5,
  Depletion = 1,
  EstDepletion = FALSE,
  IndexSD = c("estimate", "data"),
  Priors = list(FMSY = c(0.2, 2)),
  Control = NULL,
  Start = NULL,
```

```
  Uncertainty = FALSE
)
```

## Arguments

| | |
|---|---|
| `Data` | A data object. |
| `Indices`, `IndexSource`, `IndexFreq`, `IndexWeight`, `IndexSeasons` | Index selection, as in `IndexRate()`. `IndexWeight` is normalised to a mean of `1`. |
| `IndexUnits` | Character vector of index `Units` to use: one or both of `'Biomass'` and `'Number'`. Indices with other units (including `'Recruitment'`) are excluded. Indices with no units are assumed to be in biomass. |
| `CatchType` | Character. The catch the model is fitted to: `'Removals'` (landings plus discards, default) or `'Landings'`. |
| `FitYears` | `NULL` (default) fits from the first year with catch data to the last data year. Otherwise a single number (fit to that many most recent years) or a vector of consecutive years. Missing catch after the first fitted year is set to `0`. |
| `Model` | Character. The surplus production model: `'internal'` (default) or `'spict'`; see Models. |
| `Shape` | Positive number. Shape parameter `n` (fixed value, or the median of its prior when `EstShape = TRUE`). Default `2` (Schaefer). |
| `EstShape` | Logical. Estimate `Shape`? Default `FALSE`. |
| `ShapeCV` | Positive number. CV of the lognormal prior on `Shape` when `EstShape = TRUE` and `Priors$Shape` is not given. Default `0.5`. |
| `Depletion` | Positive number. Biomass in the first fitted year relative to `K` (fixed value, or starting value when `EstDepletion = TRUE`). Default `1`. |
| `EstDepletion` | Logical. Estimate `Depletion`? Default `FALSE`. |
| `IndexSD` | Character. `'estimate'` (default) or `'data'`; see Model. |
| `Priors` | Named list of lognormal priors, each `c(median, CV)`, on any of `FMSY`, `MSY` (in catch units), `Depletion`, and `Shape`. A prior on a fixed parameter is ignored. Default `list(FMSY = c(0.2, 2))`, a vague prior that keeps the `FMSY` estimate away from zero when the data contain little information on productivity. `list()` uses no priors. |
| `Control` | An `SPControl()` (`Model = 'internal'`) or `SpictControl()` (`Model = 'spict'`) list. `NULL` (default) uses the defaults of the selected model. |
| `Start` | `NULL` (default), or a named list or vector with starting values for the estimated parameters (`FMSY`, `MSY`, `Depletion`, `Shape`), e.g. the `par` of a previous fit. Used only if it includes both `FMSY` and `MSY`; otherwise the fit starts from a grid search (see `SPControl()`). |
| `Uncertainty` | Logical. Calculate the standard errors in `SE`? Default `FALSE`. Overridden by `Hessian` in `SPControl()` or `SDReport` in `SpictControl()` when these are not `'auto'`. |

## Details

### Model:

Biomass follows

$$\frac{dB}{dt} = \gamma\, MSY \left[\frac{B}{K} - \left(\frac{B}{K}\right)^n\right] - FB, \qquad \gamma = \frac{n^{n/(n-1)}}{n-1},$$

where `n` is the shape parameter (`Shape`; `2` is the Schaefer model, and `n = 1` the Fox model, used when `Shape` is within `FoxTol` of `1`; see `SPControl()`). The estimated parameters are `FMSY` and `MSY`, and optionally the initial depletion (`B/K` at the start of the first fitted year, `Depletion`) and `Shape`. Carrying capacity is `K = MSY / (FMSY * BMSY/K)` with `BMSY/K = n^(1/(1-n))`.

The model is deterministic (no process error) and the catch is assumed known. The fishing mortality in each year is solved so that the predicted catch equals the observed catch. Each year is integrated in `nSubStep` sub-steps (see `SPControl()`).

Each index is assumed proportional to biomass at the index's `Timing` within the year, with lognormal observation error. The catchability of each index is calculated analytically at its maximum likelihood value. The observation standard deviation is either estimated for each index (`IndexSD = 'estimate'`, with lower bound `MinSD`, see `SPControl()`) or calculated from the index `CV` in `Data` (`IndexSD = 'data'`, `sd = sqrt(log(1 + CV^2))`). The negative log-likelihood of each index is multiplied by its `IndexWeight`.

For seasonal data (`Data@Seasons > 1`), `Data` is first aggregated to calendar years with `AnnualData()`. The timing of an aggregated index is the mean timing, as a fraction of the calendar year, of the seasons in `IndexSeasons`.

### Models:

- `Model = 'internal'` (default): the model above, fitted by maximum likelihood with `stats::nlminb()`, using the exact gradient and Hessian of the negative log-likelihood. Numerical settings are set with `SPControl()`.
- `Model = 'spict'`: the stochastic surplus production model in continuous time (SPiCT; Pedersen and Berg 2017), fitted with the `spict` package, which must be installed (see `SpictControl()`). SPiCT estimates process error and observation error in the catch as well as the indices. The `FitSP()` arguments are converted to `spict` inputs as described in `SpictControl()`.

## Value

An object of class `spfit`, a list with elements:

- `Model`, `Converged` (logical), `Message`
- `par`: named vector of `FMSY`, `MSY`, `Depletion`, `Shape`
- `Est`: names of the estimated parameters
- `Years`: fitted years
- `B` and `B_BMSY` (length `length(Years) + 1`: at the start of each fitted year and after the last), and `F`, `F_FMSY`, and `CatchPred` (one per fitted year; `CatchPred` is `NA` for `Model = 'spict'`)
- `BMSY`, `FMSY`, `MSY`, `K`, `Shape`, `Depletion`

- `q`, `Sigma`: catchability and observation standard deviation of each index
- `Indices`: a `data.frame` with one row per index used in the fit: `Index` (name), `Source` (`'Survey'` or `'CPUE'`), `Units`, `Timing`, `Weight` (the `IndexWeight` multiplier of its negative log-likelihood, normalised to a mean of `1`), `nObs` (number of positive observations in the fitted years), `q`, and `Sigma`
- `Terminal`: named vector of `B_BMSY` (after the last fitted year) and `F_FMSY` (in the last fitted year)
- `SE`: named vector of standard errors of `logB_BMSY` (as in `Terminal`), `logFMSY`, `logF_FMSY` (as in `Terminal`), and `logMSY` (`NA` unless calculated)
- `NLL` (objective function value), `Time` (seconds), `Log` (character vector of assumptions and warnings)
- `Control`, `Prep` (the prepared data), and, for `Model = 'spict'`, `Spict` (the `spict` fit)

If the model cannot be fitted, `Converged` is `FALSE`, `Message` gives the reason, and the estimates are `NA` or absent.

### References

Pedersen, M. W. and Berg, C. W. 2017. A stochastic surplus production model in continuous time. Fish and Fisheries 18: 226-243.

### See Also

`SurplusProduction()`, `SPControl()`, `SpictControl()`, `AnnualData()`

### Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
Fit  <- FitSP(Hist@Data[[1]][[1]])
Fit$Terminal

## End(Not run)
```

---

## FitStockTargeting *Fit statistical properties of historical stock targeting*

---

### Description

Estimates the covariance of log-targeting deviations from historical targeting weights stored in `OM@StockTargeting@Targeting`, as produced by `StandardizeEffort()`.

### Usage

```
FitStockTargeting(OM, tol = 1e-06, active_thresh = 0.1)
```

### Arguments

| | |
|---|---|
| `OM` | An operating model object (om) with a populated `@StockTargeting@Targeting` array of dimension `[nSim, nStock, nFleet, nYearHist]`, as produced by `StandardizeEffort()`. |
| `tol` | `numeric(1)`. Tolerance below which targeting values are treated as zero for the purpose of covariance estimation. Default `1e-6`. |
| `active_thresh` | `numeric(1)`. Minimum proportion of historical years with positive targeting weight for a stock-fleet combination to be included in covariance estimation (strictly greater than this value). Default `0.1`. |

### Details

For each simulation, fleet, and year, the fleet-level log-effort is the geometric mean of log-targeting weights over **active stocks at that time step** (those with positive targeting weight):

$$\log \hat{E}_{f,t} = \frac{1}{|\mathcal{A}_{f,t}|} \sum_{s \in \mathcal{A}_{f,t}} \log T_{s,f,t}, \quad \mathcal{A}_{f,t} = \{s : T_{s,f,t} > 0\}$$

Stock targeting deviations are the multiplicative departures from this fleet mean:

$$\tilde{w}_{s,f,t} = \log T_{s,f,t} - \log \hat{E}_{f,t}$$

The stationary covariance $\Sigma_f$ is estimated from $\tilde{w}_{s,f,t}$ across historical years for stocks that are consistently active (positive targeting in strictly more than `active_thresh` proportion of years). Element $(\Sigma_f)_{s,s'}$ captures the co-variation in log-targeting deviations between stocks $s$ and $s'$ for fleet $f$.

The function returns `OM` unchanged if `@Mean` and `@Covariance` are already fully populated, or if `nStock == 1`.

### Value

The input `OM` object with the following slots populated:

- `@StockTargeting@Mean`: temporal mean of log-targeting deviations, exponentiated to the natural scale. Array of dimension `[nSim, nStock, nFleet]`. Values are approximately 1 for active stocks (close to zero on the log scale by the sum-to-zero construction of `StandardizeEffort()`, but not guaranteed to be exactly 1) and exactly 1 for inactive stocks (neutral, log-deviation fixed at zero).
- `@StockTargeting@Covariance`: stationary covariance matrix of log-targeting deviations, array of dimension `[nSim, nStock, nStock, nFleet]`. Rows and columns for inactive stocks are zero.

### See Also

`StandardizeEffort()`, `GenerateStockTargeting()`

`Fleet` *Fleet*

## Description

Construct and manipulate a fleet object defining the exploitation characteristics of a fleet for a stock in an operating model.

## Usage

```
Fleet(
  Name = NULL,
  Effort = new("effort"),
  Catchability = new("catchability"),
  Selectivity = new("selectivity"),
  Retention = new("retention"),
  DiscardMortality = new("discardmortality"),
  Closure = array(),
  WeightFleetRetained = array(),
  WeightFleetSelected = array(),
  Bioeconomic = new("bioeconomic"),
  Dynamics = list(),
  Misc = list()
)

Closure(x)

Closure(x) <- value

WeightFleetRetained(x)

WeightFleetRetained(x) <- value

WeightFleetSelected(x)

WeightFleetSelected(x) <- value

Fleet(x) <- value
```

## Arguments

| | |
|---|---|
| `Name` | Character or `NULL`. Fleet name, or an om object for pass-through access (see Details). |
| `Effort` | An effort object defining historical fishing effort, spatial distribution, and targeting. See `Effort()` for full documentation. Default is an empty effort. When `Name` is an om object, a numeric value here indexes the stock within the OM (see Details). |

`Catchability` A catchability object defining gear efficiency and optional projected-year stochasticity or trends. See `Catchability()` for full documentation. Default is an empty catchability. When `Name` is an om object and `Effort` is numeric, a numeric value here further indexes the fleet within the stock (see Details).

`Selectivity` A selectivity object defining selectivity-at-age, -at-length, or -at-weight. Required for all fleets. See `Selectivity()` for full documentation. Default is an empty selectivity.

`Retention` A retention object defining retention-at-age, -at-length, or -at-weight. Optional. If not specified, all age and length classes are assumed fully retained during `PopulateFleet()`. See `Retention()` for full documentation. Default is an empty retention.

`DiscardMortality`
A discardmortality object defining the proportion of discarded catch that dies. Optional. If not specified, discard mortality is set to 0 (all discards survive) during `PopulateFleet()`. See `DiscardMortality()` for full documentation. Default is an empty discardmortality.

`Closure` Numeric array or `NULL`. Spatio-temporal closure schedule with dimensions `Sim x Year x Area`. Values of 1 indicate an open area; 0 indicates a closed area. The `Sim` and `Year` dimensions may be length 1 (replicated internally). Default is an empty array; if not specified, `PopulateClosure()` sets all areas to open (1) across all simulations and years.

`WeightFleetRetained`
Numeric array or `NA`. Mean weight-at-age of fish retained (landed) by this fleet (`Sim x Age x Year`), selectivity- and retention-weighted. Used to convert landed numbers-at-age to landed biomass. If `NA` (default), `PopulateFleet()` computes it from selectivity, retention, and the stock's weight-at-length. Accessed via `WeightFleetRetained()`.

`WeightFleetSelected`
Numeric array or `NA`. Mean weight-at-age of fish selected (encountered, before retention) by this fleet (`Sim x Age x Year`), selectivity-weighted only. Combined with `WeightFleetRetained` to compute discards biomass. If `NA` (default), `PopulateFleet()` computes it from selectivity and the stock's weight-at-length.

`Bioeconomic` A bioeconomic object. Not currently used. Default is an empty bioeconomic.

`Dynamics` List. Reserved for future use for fleet dynamics model parameters. Default `list()`.

`Misc` List. Miscellaneous additional inputs. Default `list()`.

`x` A fleet object or an om object for `Fleet<-`.

`value` A fleet object or named list of fleet objects for `Fleet<-`.

## Details

A fleet object describes the fishing characteristics of a fleet for a particular stock in an operating model, including effort dynamics, catchability, selectivity, retention, discard mortality, and spatial closures.

**Required and Optional Components:**

`Selectivity()` and `Effort()` are required for all fleets. All other components are optional and will use default values if not specified:

- `Catchability()`: defaults to efficiency = 1 across all years.
- `Retention()`: defaults to full retention for all age and length classes.
- `DiscardMortality()`: defaults to 0 (all discards survive).
- `Closure`: defaults to 1 (all areas open) across all simulations and years.
- `WeightFleetRetained`/`WeightFleetSelected`: computed from selectivity, retention, and the stock's weight-at-length.

**Pass-Through Access from an OM:**

When `Name` is an om object, `Fleet()` acts as an accessor rather than a constructor:

- `Fleet(om)` returns the full fleet list (`om@Fleet`).
- `Fleet(om, st)` returns all fleets for stock index `st` (`om@Fleet[[st]]`), where `st` is passed via the `Effort` argument.
- `Fleet(om, st, fl)` returns fleet `fl` for stock `st` (`om@Fleet[[st]][[fl]]`), where `fl` is passed via the `Catchability` argument.

**Assigning Fleets to an OM:**

`Stock()` objects must be added to the OM before fleets. A single fleet object assigned with `Fleet(om) <- MyFleet` is applied to all stocks using the fleet name. A list must be structured as a stock-indexed list of fleet lists.

**Bookkeeping Slots:**

The slots `nYear`, `pYear`, `nSim`, `CurrentYear`, `Years`, and `Seasons` are inherited from the paired stock object automatically when `PopulateFleet()` is called and do not need to be set manually.

**Closure Array:**

`Closure` has dimensions `Sim x Year x Area`. The `Sim` and `Year` dimensions may be length 1 (replicated internally by `PopulateClosure()`). If the array is empty or `NULL`, all areas are set to open (1) across all simulations and years. Dimension names for `Sim`, `Year`, and `Area` are added automatically if missing.

Individual slots may be accessed or modified using `Effort()`, `Catchability()`, `Selectivity()`, `Retention()`, `DiscardMortality()`, `Closure()`, `WeightFleetRetained()`, `WeightFleetSelected()`, and `Bioeconomic()`.

## Value

- `Fleet()` returns a fleet object. If `Name` is an om object, returns the fleet list or a specific fleet from the OM.
- `Fleet<-` returns the om object `x` with the `Fleet` slot updated.
- `Closure()`, `WeightFleetRetained()`, and `WeightFleetSelected()` return the corresponding slot from the fleet object `x`.
- Their replacement forms return `x` with the corresponding slot updated.

## See Also

- fleet for the class definition and slot-level documentation.
- `OM()` for the operating model constructor.
- `Effort()`, `Catchability()`, `Selectivity()`, `Retention()`, `DiscardMortality()` for sub-object constructors.
- `PopulateFleet()` for how fleet components are expanded across simulations and years.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## Examples

```
## ---- Minimal Fleet object ----
# Effort and Selectivity are required. All other components use defaults:
# Catchability = 1, Retention = full, DiscardMortality = 0,
# Closure = all open, WeightFleet = stock weight-at-age.
eff <- Effort(Effort = data.frame(
  Year  = c(2000, 2010, 2024),
  Lower = c(0,    0.5,  1),
  Upper = c(0,    0.7,  1),
  CV    = c(0.1,  0.1,  0.1)
))
sel <- Selectivity(Pars = list(L5 = 20, LFS = 35, Vmaxlen = 1))

f <- Fleet(Name = "Trawl", Effort = eff, Selectivity = sel)
f

## ---- Fleet with all components specified ----
f_full <- Fleet(
  Name             = "Trawl",
  Effort           = eff,
  Catchability     = Catchability(Efficiency = 0.01),
  Selectivity      = sel,
  Retention        = Retention(Pars = list(LR5 = 15, LFR = 28, Rmaxlen = 1)),
  DiscardMortality = DiscardMortality(MeanAtAge = 0.2)
)
f_full

## ---- Closure: partial spatio-temporal closure ----
# Closure has dimensions Sim x Year x Area (Sim/Year may be length 1 and are
# replicated internally). 1 = open, 0 = closed.
# Here area 3 is closed from 2010 onwards; areas 1 and 2 are always open.
years <- 2000:2024
closure_array <- array(1, dim = c(1, length(years), 3),
                       dimnames = list(Sim = 1, Year = years, Area = 1:3))
closure_array[1, as.character(2010:2024), 3] <- 0

f_closure <- Fleet(Name = "Trawl", Effort = eff, Selectivity = sel,
                   Closure = closure_array)
Closure(f_closure)
```

```
## ---- Slot accessors and replacement ----
Effort(f_full)
Catchability(f_full)
Selectivity(f_full)
Retention(f_full)
DiscardMortality(f_full)
WeightFleet(f_full)

Selectivity(f_full)   <- Selectivity(Pars = list(L5 = 25, LFS = 40, Vmaxlen = 0.8))
Catchability(f_full)  <- Catchability(Efficiency = 0.02)

## ---- Assigning fleets to an OM ----
# Stock objects must be added to the OM before fleets.
stock1 <- Stock(Name = "Snapper")
stock2 <- Stock(Name = "Grouper")
om <- OM()
Stock(om) <- list(Snapper = stock1, Grouper = stock2)

# A single Fleet applied to all stocks by name
Fleet(om) <- f
Fleet(om)

# A stock-indexed list of fleets: stock -> fleet -> Fleet object
longline <- Fleet(Name = "Longline", Effort = eff,
                  Selectivity = Selectivity(Pars = list(SA50 = 4, SA50_95 = 2)))
Fleet(om) <- list(Snapper = list(Trawl = f), Grouper = list(Longline = longline))
Fleet(om)

## ---- Pass-through access from an OM ----
Fleet(om)        # full fleet list
Fleet(om, 1)     # all fleets for stock index 1 (Snapper)
Fleet(om, 2, 1)  # fleet index 1 for stock index 2 (Grouper -> Longline)
```

---

`fleet-class` *The* `fleet` *S4 Class*

---

### Description

Defines the fishing characteristics of a fleet for use in an operating model. Objects are typically created via the `Fleet()` constructor, which documents all parameters in detail.

### Slots

`Name` `character` or `NULL`. Unique fleet identifier.

`Effort` An effort object defining historical fishing effort, spatial distribution, and targeting behaviour. See `Effort()`.

`Catchability` A catchability object defining gear efficiency and optional projected-year trends or stochasticity. See `Catchability()`.

`Selectivity` A selectivity object defining selectivity-at-age, -at-length, or -at-weight. Required for all fleets. See `Selectivity()`.

`Retention` A retention object defining retention-at-age, -at-length, or -at-weight. Optional; defaults to full retention if not supplied. See `Retention()`.

`DiscardMortality` A discardmortality object defining the proportion of discarded catch that dies. Optional; defaults to 0 (all discards survive) if not supplied. See `DiscardMortality()`.

`Closure numeric` array or `NULL`. Spatio-temporal closure schedule with dimensions `Sim x Year x Area`. Values of 1 indicate an open area; 0 indicates a closed area. Defaults to 1 (all areas open) if not supplied. See `Fleet()`.

`WeightFleetRetained numeric` array or `NULL`. Mean weight-at-age of fish *retained* (landed) by this fleet (`Sim x Age x Year`) – selectivity- and retention-weighted. Used to convert landed numbers-at-age to landed biomass. If `NA`, computed from selectivity, retention, and the stock's weight-at-length during `PopulateFleet()`. Exposed via the `WeightFleet()`/`` `WeightFleet<-`() `` accessors. See `Fleet()`.

`WeightFleetSelected numeric` array or `NULL`. Mean weight-at-age of fish *selected* (encountered/caught, before retention) by this fleet (`Sim x Age x Year`) – selectivity-weighted only, not retention-weighted. Combined with `WeightFleetRetained` to compute discards biomass as the residual between selected and retained biomass. If `NA`, computed from selectivity and the stock's weight-at-length during `PopulateFleet()`. See `Fleet()`.

`Bioeconomic` A bioeconomic object. Not currently used.

`Dynamics` List. Reserved for future use for fleet dynamics model parameters. Default `list()`. Not currently used.#'

`nYear numeric` or `NULL`. Number of years. Inherited from the paired stock during `PopulateFleet()`.

`pYear numeric` or `NULL`. Number of projection years. Inherited from the paired stock during `PopulateFleet()`.

`nSim numeric` or `NULL`. Number of simulation replicates. Inherited from the paired stock during `PopulateFleet()`.

`CurrentYear numeric` or `NULL`. Current calendar year. Inherited from the paired stock during `PopulateFleet()`.

`Years numeric` or `NULL`. Full year vector (historical and projected). Inherited from the paired stock during `PopulateFleet()`.

`Seasons numeric` or `NULL`. Number of seasons. Inherited from the paired stock during `PopulateFleet()`.

`Misc list`. Miscellaneous additional inputs.

## See Also

`Fleet()` for the constructor and full parameter documentation. `Effort()`, `Catchability()`, `Selectivity()`, `Retention()`, `DiscardMortality()` for sub-object constructors.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `retention-class`, `selectivity-class`

---

`Fleet-legacy-class` *Class* `'Fleet'`

---

## Description

The component of the operating model that controls fishing dynamics

## Slots

`Name` Identifying name for the fleet. Usually includes location and gear type.

`nyears` The number of years for the historical simulation. Single value. For example, if the simulated population is assumed to be unfished in 1975 and this is the year you want to start your historical simulations, and the most recent year for which there is data available is 2019, then nyears equals 45.

`CurrentYr` The last historical year simulated before projections begin. Single value. Note that this should match the last historical year specified in the `Data` object, which is usually the last historical year for which data is available.

`EffYears` Vector indicating the historical years where there is information available to infer the relative fishing effort expended.This vector is specified in terms of the position of the year in the vector rather than the calendar year. For example, say our simulation starts with an unfished stock in 1975,and the current year (the last year for which there is data available) is 2019. Then there are 45 historical years simulated, and EffYears should include numbers between 1 and 45. Note that there may not be information available for every historical year, especially for data poor fisheries. In these situations, the EffYears vector should include only the positions of the years for which there is information, and the vector may be shorter than the total number of simulated historical years (nyears).

`EffLower` Lower bound on relative fishing effort corresponding to EffYears. EffLower must be a vector that is the same length as EffYears describing how fishing effort has changed over time. Information on relative fishing effort can be entered in any units provided they are consistent across the entire vector because the data provided will be scaled to 1 (divided by the maximum number provided).

`EffUpper` Upper bound on relative fishing effort corresponding to EffYears. EffUpper must be a vector that is the same length as EffYears describing how fishing effort has changed over time. Information on relative fishing effort can be entered in any units provided they are consistent across the entire vector because the data provided will be scaled to 1 (divided by the maximum number provided).

`Esd` Additional inter-annual variability in fishing mortality rate. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. If this parameter has a positive (non-zero) value, the yearly fishing mortality rate is drawn from a log-normal distribution with a standard deviation (in log space) specified by the value of `Esd` drawn for that simulation. This parameter applies only to historical projections.

`qinc` Mean temporal trend in catchability (also though of as the efficiency of fishing gear) parameter, expressed as a percentage change in catchability (q) per year. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Positive numbers indicate an increase and negative numbers indicate a decrease. q then changes by this amount for in each year of the simulation This parameter applies only to forward projections.

`qcv` Inter-annual variability in catchability expressed as a coefficient of variation. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This parameter applies only to forward projections.

`L5` Shortest length at which 5% of the population is vulnerable to selection by the gear used in this fleet. Values can either be specified as lengths (in the same units used for the maturity and growth parameters in the stock object) or as a percentage of the size of maturity (see the parameter isRel for more information). For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years unless cpars is used to provide time-varying selection.

`LFS` Shortest length at which 100% of the population is vulnerable to selection by the gear used by this fleet. Values can either be specified as lengths (in the same units used for the maturity and growth parameters in the stock object) or as a percentage of the size of maturity (see the parameter isRel for more information). For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years unless cpars is used to provide time-varying selection.

`Vmaxlen` Proportion of fish selected by the gear at the asymptotic length (`Stock@Linf`). Upper and Lower bounds between 0 and 1. A value of 1 indicates that 100% of fish are selected at the asymptotic length, and the selection curve is logistic. If `Vmaxlen` is less than 1 the selection curve is dome shaped. For example, if `Vmaxlen` is 0.4, then only 40% of fish are vulnerable to the fishing gear at the asymptotic length.

`isRel` Specify whether selection and retention parameters use absolute lengths or relative to the size of maturity. Single logical value (TRUE or FALSE).

`LR5` Shortest length at which 5% of the population is vulnerable to retention by the fleet. Values can either be specified as lengths (in the same units used for the maturity and growth parameters in the stock object) or as a percentage of the size of maturity (see the parameter isRel for more information). For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years unless cpars is used to provide time-varying selection.

`LFR` Shortest length where 100% of the population is vulnerable to retention by the fleet. Values can either be specified as lengths (in the same units used for the maturity and growth parameters in the stock object) or as a percentage of the size of maturity (see the parameter `isRel` for more information). For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years unless cpars is used to provide time-varying selection.

`Rmaxlen` Proportion of fish retained at the asymptotic length (`Stock@Linf`). Upper and Lower bounds between 0 and 1. A value of 1 indicates that 100% of fish are retained at the asymptotic length, and the selection curve is logistic. If `Rmaxlen` is less than 1 the retention curve is dome shaped. For example, if `Rmaxlen` is 0.4, then only 40% of fish at the asymptotic length are retained.

`DR` Discard rate, defined as the proportion of fully selected fish that are discarded by the fleet. Upper and Lower bounds between 0 and 1, with a value of 1 indicates that 100% of selected fish are discarded. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided.

`Spat_targ` Distribution of fishing in relation to vulnerable biomass (VB) across areas. The distribution of fishing effort is proportional to VB^Spat_targ. Upper and lower bounds of a uniform distribution. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This parameter allows the user to model either avoidance or spatial targeting behavior by the fleet. If the parameter value is 1, fishing effort is allocated across areas in proportion to the population density of that area. Values below 1 simulate an avoidance behavior and values above 1 simulate a targeting behavior.

`MPA` Logical argument (TRUE or FALSE). Creates an MPA in Area 1 for all years if true is selected. Defaults to FALSE.

`Misc` Miscellaneous list for bio-economic parameters

### Creating Object

Objects can be created by calls of the form `new('Fleet')`

### Author(s)

T. Carruthers and A. Hordyk

### Examples

```
showClass('Fleet')
```

---

## FMSYref *Reference management procedures*

---

### Description

Several reference MPs for your operating model to use in the management strategy evaluation. FMSYref (and related) assume perfect information about FMSY (FMSY is taken from the operating model stored at `Data@Misc$ReferencePoints$ByYear$FMSY`), and set an effort limit (TAE) so that F=FMSY (or some fraction of FMSY) in each year the MP is applied. NFref sets annual catch to zero and is used for looking at variability in stock with no fishing.

### Usage

```
FMSYref(x, Data, reps = 100, plot = FALSE)

FMSYref50(x, Data, reps = 100, plot = FALSE)
```

```
FMSYref75(x, Data, reps = 100, plot = FALSE)

NFref(x, Data, reps = 100, plot = FALSE)

curEref(x, Data, reps = 100, plot = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | A position in the data object |
| `Data` | A data object |
| `reps` | The number of stochastic samples of the MP recommendation(s) |
| `plot` | Logical. Show the plot? |

### Details

Note that you can out-perform `FMSYref` easily. The requirement for fixed F is actually quite strict and is by no means the upper limit in terms of yield. Don't panic if your method beats this one for yield, especially for short-lived species of high temporal variability in productivity!

### Value

An object of class `Rec` with the `TAC` slot populated with a numeric vector of length `reps`

### Functions

- `FMSYref()`: A reference FMSY method that fishes at FMSY
- `FMSYref50()`: A reference FMSY method that fishes at 50% of FMSY
- `FMSYref75()`: A reference FMSY method that fishes at 75% of FMSY
- `NFref()`: A reference MP that sets annual catch to almost zero (1e-15)
- `curEref()`: A reference MP that keeps fishing effort at the level of the last historical year

### Required Data

See Data-legacy for information on the `Data` object

### Author(s)

T. Carruthers, A. Hordyk

### Examples

```
FMSYref(1, MSEtool::SimulatedData, plot=TRUE)
FMSYref50(1, MSEtool::SimulatedData, plot=TRUE)
FMSYref75(1, MSEtool::SimulatedData, plot=TRUE)
NFref(1, MSEtool::SimulatedData, plot=TRUE)
curEref(1, MSEtool::SimulatedData)
```

`FullComplianceImp` *Full-Compliance Implementation Error*

## Description

An imp object representing perfect implementation of management advice: TAC and effort recommendations are realised exactly, and any size-based regulation (a change to `Advice@Retention` or `Advice@Selectivity`) is adopted immediately by the entire fleet. This is the imp object assigned to every built-in example om object (e.g. SingleStockOM, TwoFleetOM).

## Usage

```
FullComplianceImp
```

## Format

An imp object. Populated slots:

- `TAC` (impslot): `Mean = 1`, `SD = 0` – the realised TAC always equals the advised TAC exactly, no bias or stochastic error.
- `Effort` (impslot): `Mean = 1`, `SD = 0` – same, for effort-based advice.
- `Size` (impslot): `Compliance = 1` – 100% of the fleet adopts a newly-advised size regulation in the year it changes; there is no partial/lagged uptake.

## Details

Because this is behaviourally identical to leaving `Imp` unspecified (`NULL`) on an om object – a missing `Imp` entry is already treated as full compliance throughout the simulation – attaching `FullComplianceImp` makes that default explicit rather than implicit. See OverageImp and UnderageImp for TAC/effort implementation error examples, and PartialSizeComplianceImp for a partial size-regulation adoption example.

## See Also

imp, `Imp()`, `ImpSlot()`, OverageImp, UnderageImp, PartialSizeComplianceImp

Other imp: `Imp()`, `ImpSlot()`, `OverageImp`, `PartialSizeComplianceImp`, `UnderageImp`, `imp-class`, `impslot-class`

## Examples

```
FullComplianceImp
slot(FullComplianceImp, "TAC")
slot(FullComplianceImp, "Size")
```

`GenerateStockTargeting`
*Generate Stochastic Stock Targeting Trajectories*

## Description

Simulates stochastic stock-specific targeting multipliers for each fleet using independent multivariate lognormal draws at each time step.

## Usage

```
GenerateStockTargeting(
  OM,
  Period = c("Historical", "Projection"),
  TruncSD = 2,
  n_recent = NULL,
  seed = 101
)
```

## Arguments

| | |
|---|---|
| `OM` | An operating model object (om) containing a populated `@StockTargeting` slot with elements:<br>• `@Mean`: geometric mean targeting weight on the natural scale (from `FitStockTargeting()` or user defined)<br>• `@Covariance`: stationary covariance matrix of log-targeting deviations |
| `Period` | Character string specifying which period to simulate:<br>• `"Historical"`: assigns or replaces historical targeting values<br>• `"Projection"`: appends simulated projection years |
| `TruncSD` | `numeric(1)`. Number of standard deviations beyond which log-deviations are truncated. Prevents extreme targeting values when variance is high relative to the historical record length. Default `2`. Set to `Inf` to disable truncation. |
| `n_recent` | `integer(1)`. For `Period = "Projection"` only: number of recent historical years used to determine whether a stock has been continuously inactive at the end of history (converted to time steps as `n_recent * OM@Seasons`, and capped to however much history is available). A stock with zero targeting weight throughout that window is treated as inactive and receives $T_{s,t} = 0$ throughout the projection. Default `NULL`, which falls back to `OM@Control$StockTargeting$n_recent`, then to `5`. |
| `seed` | `numeric(1)`. Offset added to `OM@Seed` for the random number generator, ensuring independence from other stochastic processes. Default `101`. |

## Value

The input `OM` object with updated `@StockTargeting@Targeting`.

Targeting is modelled in log-space as multivariate normal deviations at each time step, centred at the bias-corrected historical log-mean:

$$\boldsymbol{\delta}_t \sim \text{MVN}(\mathbf{0},\, \Sigma_w)$$

$$T_{s,t} = \exp(\mu_s^* + \delta_{s,t})$$

where:

- $T_{s,t}$ is the targeting multiplier for stock $s$ at time $t$
- $\delta_{s,t}$ is the log-space deviation for stock $s$ at time $t$, drawn from the multivariate normal with mean zero and covariance $\Sigma_w$
- $\Sigma_w$ is the stationary covariance matrix of log-targeting deviations estimated by `FitStockTargeting()`
- $\mu_s^* = \bar{w}_s - \frac{1}{2}\sigma_{w,s}^2$ is the lognormal bias correction for stock $s$, where $\bar{w}_s = \log(\texttt{Mean}_s)$ is the fitted log-scale mean from `FitStockTargeting()` and $\sigma_{w,s}^2 = (\Sigma_w)_{ss}$ is the marginal variance. This ensures $\text{E}[T_{s,t}] = \exp(\bar{w}_s)$, preserving the historical mean targeting level for each stock on the natural scale.

Deviations are truncated to $|\delta_{s,t}| \leq \texttt{TruncSD} \times \sigma_{w,s}$ to guard against extreme values when variance is poorly estimated from short historical series. When variance is low, truncation has little effect and simulated values stay close to the historical mean, as intended.

Inactive stocks (zero marginal variance or non-finite mean in `@StockTargeting@Mean`, or continuously inactive over the last `n_recent` historical years when `Period = "Projection"`) receive $T_{s,t} = 0$ throughout.

## See Also

`FitStockTargeting()`, `StandardizeEffort()`

## Examples

```
## Not run:
OM <- GenerateStockTargeting(OM, Period = "Projection")
OM <- GenerateStockTargeting(OM, Period = "Projection", TruncSD = 2, n_recent = 5)

## End(Not run)
```

---

`Generic_Obs` *Obs class objects*

---

## Description

Example objects of class Obs

## Usage

```
Generic_Obs

Imprecise_Biased

Imprecise_Unbiased

Perfect_Info

Precise_Biased

Precise_Unbiased
```

## Format

An object of class `Obs` of length 1.

An object of class `Obs` of length 1.

An object of class `Obs` of length 1.

An object of class `Obs` of length 1.

An object of class `Obs` of length 1.

An object of class `Obs` of length 1.

## Examples

```
avail("Obs")
```

---

`GenHistEffort` *Generate Stochastic Historical Effort Time Series*

---

## Description

Generates stochastic historical fishing effort trajectories from an `data.frame` containing effort control points.

## Usage

```
GenHistEffort(Effort, nSim = 5, Years = NULL)
```

### Arguments

| | |
|---|---|
| `Effort` | A `data.frame` with columns:<br>**Year** Calendar years or indices defining effort control points.<br>**Lower** Lower bound on relative effort at each control year.<br>**Upper** Upper bound on relative effort at each control year.<br>**CV** Coefficient of variation for lognormal annual effort noise (typically constant across years; only the first value is used). |
| `nSim` | Integer. Number of stochastic effort simulations to generate. |
| `Years` | Optional integer vector of calendar years defining the full effort time series. Required if `Effort$Year` is specified using indices rather than calendar years. |

### Details

For each simulation, effort is sampled uniformly between specified lower and upper bounds at control years, linearly interpolated to a full annual time series, optionally perturbed by lognormal process error, and finally normalised to equal 1 in the terminal year.

For each simulation:

1. Effort at each control year is drawn independently from a uniform distribution between `Lower` and `Upper`.
2. Effort is linearly interpolated to annual resolution across `Years`.
3. If `CV` is non-`NULL`, multiplicative lognormal process error is applied with mean 1.
4. The resulting effort trajectory is normalised so that effort in the final year equals 1.

### Value

An `nSim` x `nYear` numeric array used in an `Effort()` object

### See Also

`Effort()`

---

## `GenMeanAtLength` *Generate Mean-at-Age, Length, or Weight Arrays*

---

### Description

These functions generate simulated ”mean at” arrays using a model and parameters.

### Usage

```
GenMeanAtAge(Model, Pars, Ages)

GenMeanAtLength(Model, Pars, Length)

GenMeanAtWeight(Model, Pars, Weight)
```

### Arguments

| | |
|---|---|
| `Model` | A character string naming a built-in model or a valid R function. |
| `Pars` | A named `list` of parameters for the model. |
| `Ages` | Numeric vector of age classes (for `GenMeanAtAge`). |
| `Length` | Numeric vector of length classes (for `GenMeanAtLength`). |
| `Weight` | Numeric vector of weight classes (for `GenMeanAtWeight`). |

### Value

A numeric array with dimensions: `Sim`, `Age/Class`, `Year`, and sometimes `Area`.

### See Also

`LengthModels()`, `WeightModels()`, `NaturalMortalityModels()`, MaturityModels, `FecundityModels()`, `SelectivityModels()`, `RetentionModels()`

### Examples

```
# List `Maturity` Models
MaturityModels()


# Mean-at-Age
GenMeanAtAge(Model="MaturityAtAge",
             Pars=list(A50=5, A50_95=1),
             Ages=1:10)

# Mean-at-Length
GenMeanAtLength(Model="MaturityAtLength",
                Pars=list(L50=50, L50_95=10),
                Length=seq(10,100,10))

# Mean-at-Weight
GenMeanAtWeight(Model="MaturityAtWeight",
                Pars=list(W50=3, W50_95=1),
                Weight=1:10)
```

---

## `GenMultiStockRecDevs` *Generate Correlated Multi-Stock Recruitment Deviations*

---

### Description

Generates stochastic recruitment deviations for multiple stocks using a multivariate lognormal AR(1) process, preserving both cross-stock covariance and temporal autocorrelation.

### Usage

```
GenMultiStockRecDevs(OM, TruncSD = 3, silent = FALSE, overwrite = NULL)
```

### Arguments

| | |
|---|---|
| `OM` | An operating model object of class om. |
| `TruncSD` | Numeric scalar. Number of standard deviations at which the log deviations are bounded, as in `GenRecDevs()`. Defaults to `3`. |
| `silent` | `logical(1)` Display messages? |
| `overwrite` | `logical`, length `nStock(OM)`, or `NULL` (default). Controls which stocks' `RecDevProj` are written. `NULL` overwrites every stock. Stocks with `overwrite = FALSE` keep their existing `RecDevProj` (e.g. user-supplied values) untouched, but their historical deviations still contribute to the estimated cross-stock covariance. |

### Details

Within each simulation, stocks whose historical deviations (`SRR@RecDevHist[sim, ]`) are byte-identical are treated as a single group: one latent series is simulated for the group and copied to every member, so linked stocks (e.g. a female/male pair sharing recruitment) stay identical in the projection period rather than drifting apart under independent stochastic draws. Remaining stocks/groups are correlated, not forced identical, via the process below.

The AR(1) process runs on a unit-variance latent scale that carries the cross-stock correlation:

$$Z_t = \Phi Z_{t-1} + \epsilon_t, \quad \epsilon_t \sim MVN(0, \Sigma_\epsilon)$$

where $\Phi$ is a diagonal matrix of lag-1 autocorrelation coefficients and $\Sigma_\epsilon$ is derived by `CalcInnovationCov()` from the cross-stock correlation matrix.

### Value

The input `OM` object with `RecDevProj` populated for all stocks and simulations (subject to `overwrite`).

### See Also

`GenRecDevs()`, `CalcInnovationCov()`

---

## GenRecDevs *Generate Stock-Recruitment Deviations*

---

### Description

Generate recruitment deviations for initial ages, historical years, and projected years, with optional auto-correlation and truncation.

### Usage

```
GenRecDevs(
  SD = 0.2,
  AC = 0,
  TruncSD = 3,
  Ages = NULL,
  HistYears = NULL,
  ProjYears = NULL,
  nSim = 48,
  RecDevInit = NULL,
  RecDevHist = NULL,
  RecDevProj = NULL
)
```

### Arguments

| | |
|---|---|
| `SD` | Numeric vector of standard deviations for the log recruitment deviations. This is the marginal standard deviation of the generated series, after autocorrelation and truncation. |
| `AC` | Numeric vector of autocorrelation coefficients (0 = no autocorr). Must be within -1 and 1, exclusive. |
| `TruncSD` | Numeric. Number of standard deviations at which the log deviations are bounded. Because the latent spread is widened to preserve the marginal standard deviation, the realised bound is somewhat wider than `TruncSD * SD`. Default `3`. |
| `Ages` | An `Ages()` object defining age classes. |
| `HistYears` | Numeric vector of historical years. |
| `ProjYears` | Numeric vector of projected years. |
| `nSim` | Integer. Number of simulation replicates. |
| `RecDevInit` | Optional array of initial age recruitment deviations. |
| `RecDevHist` | Optional array of historical recruitment deviations. |
| `RecDevProj` | Optional array of projected recruitment deviations. |
| | `GenRecDevs()` generates recruitment deviations across initial ages, historical years, and projected years. |
| | If `RecDevInit`, `RecDevHist`, or `RecDevProj` are provided as non-NULL arrays, their non-`NA` entries will not be overwritten and are returned as-is. Individual `NA` entries (e.g. recent historical years not yet supported by data in an imported assessment) are generated in their place, chained via the same AR(1)/truncation process off the nearest known deviation. |

### Value

A list with three elements:

- `RecDevInit`: Array of recruitment deviations for initial ages.
- `RecDevHist`: Array of recruitment deviations for historical years.
- `RecDevProj`: Array of recruitment deviations for projection years.

### Examples

```
recdevs <- GenRecDevs(
  SD = 0.2,
  AC = 0.3,
  TruncSD = 3,
  Ages = Ages(10),
  HistYears = 2000:2020,
  ProjYears = 2021:2030,
  nSim = 5
)
```

---

`GenResiduals` *Generate log-normally distributed multiplicative residuals with optional AR(1)-autocorrelation*

---

### Description

Generate random, AR(1)-autocorrelated multiplicative (normal-space) residuals for multiple simulations and years, lognormally distributed with marginal standard deviation `SD` (in log-space) and autocorrelation `AC`.

### Usage

```
GenResiduals(
  SD,
  AC,
  Years,
  nSim = NULL,
  TruncSD = 2,
  nSeasons = 1,
  NA_Season = list(),
  LastError = NULL
)
```

### Arguments

| | |
|---|---|
| `SD` | Numeric vector of length nSim giving the marginal standard deviation, in log-space, of the returned (AC-propagated) residuals. |
| `AC` | Numeric vector of length nSim giving autocorrelation (must be within -1 and 1, exclusive). |
| `Years` | Numeric vector of years for which residuals are generated. |
| `nSim` | Optional integer number of simulations (defaults to length(SD)). |
| `TruncSD` | Numeric scalar specifying how many standard deviations at which to truncate the innovations. |
| `nSeasons` | Number of seasons in the array (default 1). |

| | |
|---|---|
| `NA_Season` | Optional. List length `nSim` with each element containing integer vector indicating which (if any) seasons have NA values for all years |
| `LastError` | Optional numeric vector of length nSim giving the starting value for the AR(1) recursion, in normal space; i.e. the last observed multiplicative residual. `NA` (default) means no prior state |

### Value

Numeric array of dimensions nSim x nYear containing multiplicative (normal-space) log-normal residuals with AR(1) autocorrelation `AC` applied.

### See Also

`ApplyAC()`, `CalcResidualStats()`

---

## `getclass` *get object class*

---

### Description

Internal function for determining if object is of classy

### Usage

```
getclass(x, classy)
```

### Arguments

| | |
|---|---|
| `x` | Character string object name |
| `classy` | A class of object (character string, e.g. 'Fleet') |

### Value

TRUE or FALSE

### Author(s)

T. Carruthers with nasty hacks from A. Hordyk

`getDataList` *Get part of an MP specific data-list*

## Description

Get part of an MP specific data-list

## Usage

```
getDataList(MSElist, mm)
```

## Arguments

| | |
|---|---|
| `MSElist` | A hierarchical list [Stock][Fleet][MP] |
| `mm` | integer the MP number |

## Value

a sublist of MSElist for a specific MP

`getfirstlev` *Extract the first dimension of a hierarchical list of recommendation objects*

## Description

Extract the first dimension of a hierarchical list of recommendation objects

## Usage

```
getfirstlev(x, name, pp, ff)
```

## Arguments

| | |
|---|---|
| `x` | Simulation number |
| `name` | Character. The slot name to extract. |
| `pp` | Integer. The stock number (second level list) |
| `ff` | Integer. The fleet number (third level list) |

## Author(s)

T. Carruthers

`GetMetaData` *Retrieve Operating Model Metadata*

## Description

Extracts key metadata from an operating model (OM) object, including number of simulations, stock-specific age classes, number of areas, and stock names and fleet names

## Usage

```
GetMetaData(OM, Period = c("Historical", "Projection", "All"), Years = NULL)
```

## Arguments

| | |
|---|---|
| `OM` | An operating model object. |
| `Period` | Character string specifying the model period. Must be one of `"Historical"`, `"Projection"`, or `"All"`. Default is `"All"`. |
| `Years` | Optional vector of years to subset the OM. If `NULL`, all years for the specified period are returned. |

## Value

A named list with the following elements:

- `nSim` — number of simulations in the OM
- `AgeClasses` — list of age classes for each stock
- `nAges` — list of the number of age classes for each stock
- `nAreas` — number of spatial areas (must be consistent across stocks)
- `StockNames` — character vector of stock names
- `Years` — vector of years for the specified period
- `Period` — resolved period argument
- `FleetNames` — character vector of fleet names (from the first stock)

## Examples

```
## Not run:
meta <- GetMetaData(MyOM, Period = "Historical")
meta$nSim
meta$AgeClasses

## End(Not run)
```

`getmov2` *Optimization function to find a movement model that matches user specified movement characteristics modified for Rcpp.*

## Description

The user specifies the probability of staying in the same area and spatial heterogeneity (both in the unfished state).

## Usage

```
getmov2(x, Prob_staying, Frac_area_1)
```

## Arguments

| | |
|---|---|
| `x` | A position in vectors Prob_staying and Frac_area_1 |
| `Prob_staying` | User specified probability that individuals in area 1 remain in that area (unfished conditions) |
| `Frac_area_1` | User specified fraction of individuals found in area 1 (unfished conditions) |

## Details

This is paired with movfit to find the correct movement model.

## Value

A markov movement matrix

## Author(s)

T. Carruthers

## Examples

```
Prob_staying<-0.8 # probability  that individuals remain in area 1 between time-steps
Frac_area_1<-0.35 # the fraction of the stock found in area 1 under equilibrium conditions
markovmat<-getmov2(1,Prob_staying, Frac_area_1)
vec<-c(0.5,0.5) # initial guess at equilibrium distribution (2 areas)
for(i in 1:300)vec<-apply(vec*markovmat,2,sum) # numerical approximation to stable distribution
c(markovmat[1,1],vec[1]) # pretty close right?
```

---

`getMP` *Search R session for MP*

---

## Description

Calls `dynGet()`, then `get()` in order to find the MP definition in the R session.

## Usage

```
getMP(MP)
```

## Arguments

| | |
|---|---|
| `MP` | Character of MP name |

## Value

The function definition or an error message from `try()` if unsuccessful

## Author(s)

Q. Huynh

---

`getnIVs` *Count independent variables for a MICE relationship at position x in a Rel list*

---

## Description

Count independent variables for a MICE relationship at position x in a Rel list

## Usage

```
getnIVs(x, Rel)
```

## Arguments

| | |
|---|---|
| `x` | Position of a MICE relationship in the list Rel (MOM@Rel) |
| `Rel` | The list of MICE relationships (MOM@Rel) |

## Author(s)

T.Carruthers

## getsel *Calculate selectivity curve*

### Description

Calculate selectivity curve

### Usage

```
getsel(x, lens, lfs, sls, srs)
```

### Arguments

| | |
|---|---|
| x | Simulation number |
| lens | Matrix of lengths (nsim by nlengths) |
| lfs | Vector of length at full selection (nsim long) |
| sls | Vector of sigmas of ascending limb (nsim long) |
| srs | Vector of sigmas of descending limb (nsim long) |

## GetSSRefPoints *Get Reference Points from Stock Synthesis Output*

### Description

Reads biological reference points from one or more Stock Synthesis (SS3) model outputs and returns the requested quantities from the `derived_quants` table.

### Usage

```
GetSSRefPoints(
  SSDir,
  Labels = c("annF_F01", "Dead_Catch_F01", "SSB_SPR", "annF_SPR", "Dead_Catch_SPR",
    "SSB_MSY", "SPR_MSY", "annF_MSY", "Dead_Catch_MSY", "Ret_Catch_MSY")
)
```

### Arguments

| | |
|---|---|
| SSDir | One of:<br>• A character string giving the path to an SS3 output directory, passed to `ImportSSReport()`.<br>• A `RepList` object returned by `ImportSSReport()`.<br>• A plain list in `RepList` format, i.e. `RepList[[1]]`. |
| Labels | Character vector of `derived_quants` labels to extract. Defaults to a standard set of F, catch, and biomass reference points: |

- `"annF_F01"`, `"Dead_Catch_F01"` — F0.1 fishing mortality and dead catch.
- `"SSB_SPR"`, `"annF_SPR"`, `"Dead_Catch_SPR"` — SSB, F, and dead catch at the SPR target.
- `"SSB_MSY"`, `"SPR_MSY"`, `"annF_MSY"`, `"Dead_Catch_MSY"`, `"Ret_Catch_MSY"` — SSB, SPR, F, dead catch, and retained catch at MSY.

### Value

If `SSDir` contains a single model, a data frame with columns:

- `Label` — the `derived_quants` label.
- `Value` — the point estimate, rounded to 2 decimal places.
- `StdDev` — the standard deviation, rounded to 2 decimal places.

If `SSDir` contains multiple models, a list of such data frames, one per model.

### See Also

`ImportSSReport()`

---

## hconv *Stock recruit parameterization*

---

### Description

Convert stock recruit parameters from steepness parameterization to alpha/beta (and vice versa)

### Usage

```
hconv(alpha, phi0, SR = 1, type = 1)

R0conv(alpha, beta, phi0, SR = 1, type = 1)

SRalphaconv(h, phi0, SR = 1, type = 1)

SRbetaconv(h, R0, phi0, SR = 1, type = 1)
```

### Arguments

| | |
|---|---|
| `alpha` | Alpha parameter |
| `phi0` | Unfished spawners per recruit |
| `SR` | Stock-recruit function: (1) Beverton-Holt, or (2) Ricker |
| `type` | The parameterization of the Beverton-Holt function with respect to `alpha` and `beta`. See details. |
| `beta` | Beta parameter |
| `h` | Steepness parameter |
| `R0` | Unfished recruitment parameter |

### Details

The Type 1 Beverton-Holt equation is

$$R = \alpha S/(1 + \beta S)$$

The Type 2 Beverton-Holt equation is

$$R = S/(\alpha + \beta S)$$

The Ricker equation is

$$R = \alpha S \exp(-\beta S)$$

### Value

A numeric.

### Functions

- `hconv()`: Returns steepness (h) from `alpha` and `phi0`
- `R0conv()`: Returns unfished recruitment (R0) from `alpha`, `beta`, and `phi0`
- `SRalphaconv()`: Returns `alpha` from `h` and `phi0`
- `SRbetaconv()`: Returns `beta` from `h`, `R0`, and `phi0`

### Author(s)

Q. Huynh

---

## `Herm` *Stock Transition Constructor*

---

### Description

Construct a stocktransition object defining an age-dependent reclassification of individuals from one stock's numbers-at-age into another's.

### Usage

```
Herm(From, To, Frac = NULL, Misc = list())
```

### Arguments

| | |
|---|---|
| `From` | `character(1)` or `numeric(1)`. Source stock, specified by stock name or 1-based integer index. |
| `To` | `character(1)` or `numeric(1)`. Destination stock, specified by stock name or 1-based integer index. Must differ from `From`. |
| `Frac` | `array` or `NULL`. Cumulative "fraction remaining in `From` by age" curve - `[sim, age]` or `[sim, age, year]`, values in `[0, 1]`. Starts near `1` at the youngest ages (almost no one has transitioned yet) and decreases toward `0` at older ages (nearly everyone has transitioned out of `From`). Default `NULL`. |
| `Misc` | `list`. Used internally. Default `list()`. |

### Details

`Herm()` is the general mechanism, not exclusive to biological hermaphroditism, but it is named and documented around its primary motivating use case, sequential hermaphroditism (e.g. protogynous wrasse or grouper changing sex from female to male at a given age). The same mechanism applies to any scenario shaped as "individuals move from one `stock` object's bookkeeping into another's as they age" (ontogenetic stage transitions modelled as separate stocks).

`om@Herm` takes a list of `stocktransition` objects (one per `From`/`To` pair), supporting multiple simultaneous transition relationships.

### Value

A stocktransition object. When `From` is an om, hist, or mse object (and `To` is not supplied), `Herm()` instead acts as a pass-through accessor and returns that object's `Herm` slot (a list of `stocktransition` objects, or `NULL`).

### See Also

stocktransition

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
# Protogynous transition: female stock transitions into male stock,
# onset around age 5 with a step-function fraction-remaining curve
Frac <- array(c(1, 1, 1, 1, 1, 0, 0, 0, 0, 0), dim = c(1, 10))
herm <- Herm(From = "Female", To = "Male", Frac = Frac)

# equivalent, specified by stock index
herm <- Herm(From = 1, To = 2, Frac = Frac)
```

`Herm-int` *Internal Herm functions*

## Description

- `expandHerm` expands the Herm list in SexPars to a matrix of fractions at age
- `checkHerm` checks that each array in the list has dimension nsim x maxage+1 x nyears + proyears. For backwards compatibility, also converts matrices to arrays by adding the year dimension.
- `subsetHerm` returns year-specific Herm values.

## Usage

```
expandHerm(Herm, maxage, np, nsim)

checkHerm(Herm, maxage, nsim, nyears, proyears)

subsetHerm(Herm, y)
```

## Arguments

| | |
|---|---|
| `Herm` | A list of Hermaphroditic fractions at age |
| `maxage` | The maximum age of stocks being simulated |
| `np` | The total number of stocks being simulated |
| `nsim` | The number of simulations |
| `nyears` | The number of historical years |
| `proyears` | The number of projection years |
| `y` | The year to subset |

## Author(s)

T. Carruthers

Q. Huynh

`HermOM` *Hermaphroditic Wrasse Example Operating Model*

## Description

An example om object representing a protogynous hermaphrodite wrasse (*Thalassoma* spp.): a female stock and a male stock, each exploited by two fleets.

## Usage

```
HermOM
```

## Format

An om object with the following slots populated:

- `Name`: `"Hermaphroditic Wrasse"`.
- `nSim`: `8`. Number of stochastic simulations.
- `nYear`: `20`. Number of historical years.
- `pYear`: `30`. Number of projection years.
- `Stock`: a length-2 list of `[[WrasseFemaleExStock, WrasseMaleExStock]]`.
- `Fleet`: a length-2 list, each containing a length-2 list of `[[AsympExFleet, DomeExFleet]]`. Structure is `[[stock]][[fleet]]`. Both stocks are fished by the same two fleet types.
- `Obs`: a length-2 list, each containing a length-2 list of `[[AgeStructuredObs, CommercialFleetObs]]`. Structure is `[[stock]][[fleet]]`.
- `Imp`: a length-1 FullComplianceImp, broadcast to both stocks and fleets. Perfect TAC/effort implementation and full immediate size-regulation compliance.
- `Herm`: a length-1 list containing one stocktransition object (built via `Herm()`) transitioning individuals from the female stock into the male stock as they age. See `## Herm Transition` below.

## Details

Linked by two mechanisms that operate together but are otherwise independent: `SRR@SPFrom` (the male sources its spawning production from the female) and a `Herm()` transition (a fraction of females change sex into males as they age).

Intended to illustrate both mechanisms together on a realistic case study. Pairings of stocks, fleets, and observation models are for illustration only and should not be taken as ecologically meaningful. See also MultiStockOM for a non-hermaphroditic two-stock example.

**Stocks:**

WrasseFemaleExStock and WrasseMaleExStock share an identical age structure (`Ages@Classes = 0:15`) - required for the `Herm()` transition, which reclassifies individuals age-for-age between the two stocks - but differ in growth, maturity, natural mortality, and `SRR`. See WrasseFemaleExStock/WrasseMaleExStock for full parameter details.

**Fleets:**

Both stocks are fished by the same two fleet types, as in MultiStockOM:

- AsympExFleet: asymptotic length-based selectivity; effort stabilises early in the historical period.
- DomeExFleet: dome-shaped selectivity; effort increases gradually across the historical period.

**Observation Model:**

Both stocks use the same pair of observation models by fleet: AgeStructuredObs (landings, a spawning-biomass survey, and age-composition samples) on the AsympExFleet fleet, and CommercialFleetObs (landings, discards, effort, and a commercial CPUE index) on the DomeExFleet fleet.

**Implementation Error:**

FullComplianceImp models perfect implementation - realised TAC/effort always equal advice exactly, and size-regulation advice is adopted immediately. See `Imp()` for details.

**SPFrom:**

`SRR@SPFrom` lets one stock's recruitment be driven by another stock's spawning production, rather than its own.

In this example only the female stock spawns: WrasseMaleExStock's `SRR` sets `SPFrom = "Example Wrasse Female Stock"`, so the male stock's spawning biomass/production (`SP/SP0`) is read from the female stock.

**Herm Transition:**

`Herm()` physically moves individuals from one stock's numbers-at-age into another's as they age. `HermOM@Herm` is a stocktransition object, `Herm(From = "Example Wrasse Female Stock", To = "Example Wrasse Male Stock", Frac = ...)`, where `Frac` is the cumulative fraction of individuals remaining female by age: close to 1 at young ages, decreasing smoothly (a logistic curve centred on age 8 here) toward 0 by the oldest ages. Once transitioned, an individual is an age-matched male from that point on: weight, maturity, natural mortality, and selectivity are all determined by the male stock's own biology.

## See Also

MultiStockOM for a non-hermaphroditic two-stock structural variant. WrasseFemaleExStock, WrasseMaleExStock, AsympExFleet, DomeExFleet, AgeStructuredObs, CommercialFleetObs, FullComplianceImp for the component objects. `SRR()` for `SPFrom`, `Herm()` and stocktransition for the transition mechanism. `OM()`, om, `runMSE()`, `PopulateOM()`, `ExampleMPs()`

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
HermOM
nStock(HermOM)
```

```
HermOM@Herm

## Not run:
Hist <- runMSE(HermOM, MPs = ExampleMPs())

## End(Not run)
```

---

`Hist` `hist` *Class and Accessor*

---

## Description

The hist object stores the complete historical time-series generated during a MSE spool-up, including population dynamics, exploitation history, reference points, and observational data. See hist for full documentation of all slots.

## Usage

```
Hist(MSE = NULL)
```

## Arguments

| | |
|---|---|
| `MSE` | An mse object. If `NULL` (default), an empty hist object is returned. |

## Details

`hist` objects are created by `Simulate()`. `Hist()` recovers the `hist` object from an mse object, combining the historical time-series in `MSE@Hist` with the operating model, reference points, and historical data stored in the `mse` object.

## Value

A hist object.

## See Also

hist, mse, `Simulate()`, `Project()`

## Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
MSE  <- Project(Hist, MPs = c('CurrentCatch', 'CurrentEffort'))
MSE2 <- Project(Hist(MSE), MPs = 'NoFishing')

## End(Not run)
```

`hist-class` `hist` *Class*

## Description

The `hist` class stores the complete historical time-series generated from an om object. It extends the timeseries with the originating operating model, unfished reference states, biological reference points, and associated observational data. Objects of this class are typically created by running `Simulate()`, and serve as the starting point for forward projections in the MSE.

## Slots

`OM` The om object used to generate the historical time-series.

`Unfished` An unfished object containing two popdynamics sub-objects:

- `Equilibrium`: Unfished equilibrium population dynamics, used as the baseline for computing biological reference points such as B0 and SSB0.
- `Dynamic`: Unfished dynamic population trajectories, representing the counterfactual population in the absence of fishing over the historical period.

`Reference` A reference object containing biological and management reference points.

`Data` A nested named list of data objects containing historical observations generated during the spool-up phase. The list has two levels:

- **Level 1** (`nSim` elements, or length 1 if all simulations are identical): one entry per simulation.
- **Level 2** (one entry per stock or stock complex defined in the om): a data object holding the observations for that stock/complex.

`Log` `list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

`Misc` Named list for carrying arbitrary additional objects alongside the historical results. Useful for attaching model-specific diagnostics or intermediate outputs.

`Number` List of numbers-at-age arrays, one element per stock. Each array has dimensions `Sim × Age × Year × Area`.

`Biomass` Total biomass array with dimensions `Sim × Stock × Year`.

`SBiomass` Spawning biomass array with dimensions `Sim × Stock × Year`.

`SProduction` Spawning production array with dimensions `Sim × Stock × Year`.

`Interactions` Total biomass interacting with fishing gear, by stock and fleet. Array with dimensions `Sim × Stock × Year × Fleet`.

`Landings` Total landed biomass by stock and fleet. Array with dimensions `Sim × Stock × Year × Fleet`.

`Discards` Total discarded biomass by stock and fleet. Array with dimensions `Sim × Stock × Year × Fleet`.

`InteractAtAge` List of interactions-at-age arrays (numbers), one element per stock. Each array has dimensions `Sim × Age × Year × Fleet × Area`.

`LandingsAtAge` List of landings-at-age arrays (numbers), one element per stock. Each array has dimensions `Sim × Age × Year × Fleet × Area`.

`DiscardsAtAge` List of discards-at-age arrays (numbers), one element per stock. Each array has dimensions `Sim × Age × Year × Fleet × Area`.

`LandingsAtSize` Nested list of landings-at-size arrays (numbers), indexed by stock then fleet. Each array has dimensions `Sim × Class × Year × Area`.

`DiscardsAtSize` Nested list of discards-at-size arrays (numbers), indexed by stock then fleet. Each array has dimensions `Sim × Class × Year × Area`.

`Effort` Fishing effort array with dimensions `Sim × Year × Fleet`.

`Distribution` Fleet effort distribution across areas. Array with dimensions `Sim × Year × Fleet × Area`.

`FInteract` Apical fishing mortality for fish that interact with the fishing gear. Array with dimensions `Sim × Stock × Year × Fleet` (`array.null`).

`FDead` Apical fishing mortality for fish killed by the fishing gear. Array with dimensions `Sim × Stock × Year × Fleet` (`array.null`).

`FRetain` Apical fishing mortality for fish retained by fishers. Array with dimensions `Sim × Stock × Year × Fleet` (`array.null`).

`FInteractArea` Area- and age-specific fishing mortality for fish interacting with the gear, as a list by stock. Each element has dimensions `Sim × Age × Year × Fleet × Area` (`array.list.null`).

`FDeadArea` Area- and age-specific fishing mortality for fish killed by the gear, as a list by stock. Each element has dimensions `Sim × Age × Year × Fleet × Area` (`array.list.null`).

`FRetainArea` Area- and age-specific fishing mortality for retained fish, as a list by stock. Each element has dimensions `Sim × Age × Year × Fleet × Area` (`array.list.null`).

## See Also

om, unfished, reference, timeseries, data, popdynamics

---

# `Hist-legacy-class` *Class* `'Hist'`

---

## Description

An object for storing information generated by the end of the historical simulations

## Slots

`Data` The Data object at the end of the historical period

`OMPars` A numeric data.frame with nsim rows with sampled Stock, Fleet, Obs, and Imp parameters.

`AtAge` A named list with arrays of dimensions: `c(nsim, maxage+1, nyears+proyears)` or `c(nsim, maxage+1, nyears, nareas)`

- Length: Length-at-age for each simulation, age, and year

- Weight: Weight-at-age for each simulation, age, and year
- Select: Selectivity-at-age for each simulation, age, and year
- Retention: Retention-at-age for each simulation, age, and year
- Maturity: Maturity-at-age for each simulation, age, and year
- N.Mortality: Natural mortality-at-age for each simulation, age, and year
- Z.Mortality: Total mortality-at-age for each simulation, age, year and area
- F.Mortality: Fishing mortality-at-age for each simulation, age, year and area
- Fret.Mortality: Fishing mortality-at-age for retained fish for each simulation, age, year and area
- Number: Total numbers by simulation, age, year and area
- Biomass: Total biomass by simulation, age, year and area
- VBiomass: Vulnerable biomass by simulation, age, year and area
- SBiomass: Spawning biomass by simulation, age, year and area
- Removals: Removals (biomass) by simulation, age, year and area
- Landings: Landings (biomass) by simulation, age, year and area
- Discards: Discards (biomass) by simulation, age, year and area

`TSdata` A named list with population and fleet dynamics:

- Number: Total numbers; array dimensions `c(nsim, nyears, nareas)`
- Biomass: Total biomass; array dimensions `c(nsim, nyears, nareas)`
- VBiomass: Vulnerable biomass; array dimensions `c(nsim, nyears, nareas)`
- SBiomass: Spawning Biomass; array dimensions `c(nsim, nyears, nareas)`
- Removals: Removals (biomass); array dimensions `c(nsim, nyears, nareas)`
- Landings: Landings (biomass); array dimensions `c(nsim, nyears, nareas)`
- Discards: Discards (biomass); array dimensions `c(nsim, nyears, nareas)`
- Find: Historical fishing mortality (scale-free); matrix dimensions `c(nsim, nyears)`
- RecDev: Recruitment deviations (historical and projection); matrix dimensions `c(nsim, nyears+proyears+maxage)`
- SPR: Named list with Equilibrium and Dynamic SPR (both matrices iwth dimensions `c(nsim, nyears)`)
- Unfished_Equilibrium: A named list with unfished equilibrium numbers and biomass-at-age

`Ref` A named list with biological reference points:

- ByYear: A named list with asymptotic reference points (i.e., calculated annually without recruitment deviations) all matrices with dimensions `nsim` by `nyears+proyears`:
  - N0: Asymptotic unfished total number
  - SN0: Asymptotic unfished spawning number
  - B0: Asymptotic unfished total biomass
  - SSB0: Asymptotic unfished spawning biomass
  - VB0: Asymptotic unfished vulnerable biomass
  - MSY: Asymptotic MSY
  - FMSY: Fishing mortality corresponding with asymptotic MSY
  - SSBMSY: Spawning stock biomass corresponding with asymptotic MSY

  - BMSY: total biomass corresponding with asymptotic MSY
  - VBMSY: Vulnerable biomass corresponding with asymptotic MSY
  - F01: Fishing mortality where the change in yield per recruit is 10% of that at F = 0
  - Fmax: Fishing mortality that maximizes yield per recruit
  - F_SPR: Fishing mortality corresponding to spawning potential ratio of 20 - 60% in increments of 5%; array dimensions `c(nsim, 9, nyears+proyears)`
  - Fcrash: Fishing mortality corresponding to the recruits-per-spawner at the origin of the stock-recruit relationship
  - Fmed: Fishing mortality corresponding to the median recruits-per-spawner in the historical period
  - SPRcrash: SPR corresponding to the recruits-per-spawner at the origin of the stock-recruit relationship
- Dynamic_Unfished: A named list with dynamic unfished reference points for each simulation and year:
  - N0: Unfished total numbers
  - B0: Unfished total biomass
  - SN0: Unfished spawning numbers
  - SSB0: Unfished spawning biomass
  - VB0: Unfished vulnerable biomass
  - Rec: Unfished recruitment
- ReferencePoints: A data.frame with `nsim` rows with with biological reference points calculated as an average over age-of-maturity `ageM` years around the current year (i.e. `nyears`):
  - N0: Average unfished numbers
  - B0: Average unfished biomass
  - SSB0: Average unfished spawning biomass (used to calculate depletion)
  - SSN0: Average unfished spawning numbers
  - VB0: Average unfished vulnerable biomass (used to calculate depletion if `cpar$control$D='VB'`)
  - MSY: Average maximum sustainable yield (equilibrium)
  - FMSY: Average fishing mortality corresponding with MSY
  - SSBMSY: Average spawning stock biomass corresponding with MSY
  - BMSY: Average total biomass corresponding with MSY
  - VBMSY: Average vulnerable biomass corresponding with MSY
  - UMSY: Average exploitation rate corresponding with MSY
  - FMSY_M: Average FMSY/M ratio
  - SSBMSY_SSB0: Average ratio of SSBMSY to SSB0
  - BMSY_B0: Average ratio of BMSY to B0
  - VBMSY_VB0: Average ratio of VBMSY to VB0
  - RefY: Maximum yield obtained in forward projections with a fixed F

`SampPars` A named list with all sampled Stock, Fleet, Obs, and Imp parameters

`OM` The `OM` object (without cpars)

`Misc` A list for additional information

### Author(s)

A. Hordyk

---

## hist2 *Wrapper for histogram function*

---

### Description

Produces a blank plot if all values in x are equal

### Usage

```
hist2(x, col, axes = FALSE, main = "", breaks = 10, cex.main = 1)
```

### Arguments

| | |
|---|---|
| x | A vector of values |
| col | Colour of the histogram |
| axes | Logical - should axes be included? |
| main | Character - main title |
| breaks | Number of breaks. See ?hist for more details |
| cex.main | Text size of the main title |

---

## HistoricalWeight *Seasonal Allocation of TAC/Effort for OM Objects*

---

### Description

Access or replace the `HistoricalWeight` and `SeasonalAllocation` slots of an om object. Only relevant for seasonal OMs (`Seasons > 1`): together they control how a periodically-set TAC or Effort (an MP is not necessarily called every season - see `Interval()`) is split across the seasons of the interval it covers. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
HistoricalWeight(x)

HistoricalWeight(x) <- value

SeasonalAllocation(x)

SeasonalAllocation(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a named list of length 0 or the number of stock complexes. For `HistoricalWeight`, each element is a named numeric vector over fleet names with values in `[0,1]` (or a single unnamed value applied to every fleet). For `SeasonalAllocation`, each element is an `nSim` (or 1, recycled) by `Seasons` by `nFleet` array, each `[sim, , fleet]` slice summing to 1. `NULL` clears it. |

### Details

For each stock complex, `SeasonalAllocation` is resolved in this order:

1. Use `SeasonalAllocation` if specified.
2. Derive it from `HistoricalWeight`, blending, per fleet, the historical seasonal share of removals with the seasonal share of population biomass: `HistoricalWeight * historical_share + (1 - HistoricalWeight) * biomass_share`, pooled over the last `OM@Control$SeasonalAllocationYears` years (default `5`).

`HistoricalWeight = 1` (the default when unspecified) uses the historical removals pattern only; `HistoricalWeight = 0` uses the population biomass pattern only.

### Value

- `HistoricalWeight()`/`SeasonalAllocation()` return the value of the corresponding slot.
- `HistoricalWeight<-()`/`SeasonalAllocation<-()` return `x` with the corresponding slot updated.

### See Also

- `OM()` for the constructor, om for the class definition.
- `FleetAllocation()` for how a stock/complex TAC is split among fleets.
- `Interval()` for the management update frequency that determines how many seasons a single TAC/Effort value covers.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM(Seasons = 4)
HistoricalWeight(om) <- list(Stock1 = c(F1 = 0.8, F2 = 1))
HistoricalWeight(om)
```

`HockeyStickHCR` *Hockey-Stick Harvest Control Rule*

## Description

Modifies a trial rate (e.g. an exploitation or fishing mortality rate) based on the current level of some status indicator (e.g. an index, or biomass) relative to a reference (target) level.

## Usage

```
HockeyStickHCR(
  TrialRate,
  Est,
  Ref,
  ControlPointsIndex = c(0, 1),
  ControlPointsRate = c(0, 1),
  RampType = c("linear", "smooth")
)
```

## Arguments

| | |
|---|---|
| `TrialRate` | Positive number. Trial rate (or catch, as in `IndexTarget()`) before HCR adjustment. |
| `Est`, `Ref` | Positive numbers, same units. Current status level and its target/reference level. |
| `ControlPointsIndex` | Numeric vector, length `>= 2`, non-decreasing (`c(Lx, Ux)`, or `c(Lx, Mx, Ux)`, ...). Control points in units of `Est / Ref`. Equal adjacent points collapse that segment to a step (e.g. `c(0, 0)` with `ControlPointsRate = c(0, 1)` disables the HCR - a positive `Est / Ref` always resolves to the upper branch). |
| `ControlPointsRate` | Numeric vector, the same length as `ControlPointsIndex`. The rate multipliers corresponding to `ControlPointsIndex`. |
| `RampType` | Character. Shape of the interpolation between consecutive control points: `'linear'` (default), or `'smooth'` (a cubic smoothstep, avoiding the slope discontinuity a linear ramp has at each control point). |

## Details

A piecewise control rule over 2 or more control points:

- below `ControlPointsIndex[1]`, `Rate = TrialRate * ControlPointsRate[1]`
- above `ControlPointsIndex[n]`, `Rate = TrialRate * ControlPointsRate[n]`
- between consecutive control points, the multiplier is interpolated according to `RampType`.

With 2 control points this is the classic two-inflection-point ("hockey-stick") rule. With 3, a middle control point can be added, e.g. to cap the rate above a very healthy status as well as floor it below a limit.

## Value

A numeric rate.

## See Also

`IndexRate()`, `Advice()`

## Examples

```
# At or above target (Est/Ref >= 1): full trial rate applies
HockeyStickHCR(TrialRate = 0.2, Est = 1.2, Ref = 1,
               ControlPointsIndex = c(0.5, 1), ControlPointsRate = c(0, 1))

# Partway up the 0.5-1 ramp: rate scaled down accordingly
HockeyStickHCR(TrialRate = 0.2, Est = 0.75, Ref = 1,
               ControlPointsIndex = c(0.5, 1), ControlPointsRate = c(0, 1))

# At or below the lower control point: rate set to zero
HockeyStickHCR(TrialRate = 0.2, Est = 0.3, Ref = 1,
               ControlPointsIndex = c(0.5, 1), ControlPointsRate = c(0, 1))

# Three control points: ramp up to target, then taper above a high level
HockeyStickHCR(TrialRate = 0.2, Est = 1.8, Ref = 1,
               ControlPointsIndex = c(0.5, 1, 2),
               ControlPointsRate  = c(0, 1, 0.7))

# Plot the control rule across a range of current status (Est / Ref)
Level <- seq(0, 1.5, length.out = 200)
Rate  <- sapply(Level, HockeyStickHCR,
                TrialRate = 0.2, Ref = 1,
                ControlPointsIndex = c(0.5, 1), ControlPointsRate = c(0, 1))
plot(Level, Rate, type = "l", lwd = 2,
     xlab = "Current status (Est / Ref)", ylab = "Rate")
abline(v = c(0.5, 1), lty = 3, col = "grey50")

# Linear vs smooth ramp: same control points, different interpolation
# 'smooth' avoids the kink at each control point
RateLinear <- sapply(Level, HockeyStickHCR,
                     TrialRate = 0.2, Ref = 1,
                     ControlPointsIndex = c(0.5, 1), ControlPointsRate = c(0, 1),
                     RampType = 'linear')
RateSmooth <- sapply(Level, HockeyStickHCR,
                     TrialRate = 0.2, Ref = 1,
                     ControlPointsIndex = c(0.5, 1), ControlPointsRate = c(0, 1),
                     RampType = 'smooth')
plot(Level, RateLinear, type = "l", lwd = 2,
     xlab = "Current status (Est / Ref)", ylab = "Rate")
lines(Level, RateSmooth, lwd = 2, col = "dodgerblue")
```

```
abline(v = c(0.5, 1), lty = 3, col = "grey50")
legend("topleft", c("linear", "smooth"), col = c("black", "dodgerblue"), lwd = 2)

# Three control points, tapering above target
Level3 <- seq(0, 2.5, length.out = 200)
Rate3  <- sapply(Level3, HockeyStickHCR,
                 TrialRate = 0.2, Ref = 1,
                 ControlPointsIndex = c(0.5, 1, 2),
                 ControlPointsRate  = c(0, 1, 0.7))
plot(Level3, Rate3, type = "l", lwd = 2,
     xlab = "Current status (Est / Ref)", ylab = "Rate")
abline(v = c(0.5, 1, 2), lty = 3, col = "grey50")

# Same three control points, with a smooth ramp between each pair
Rate3Smooth <- sapply(Level3, HockeyStickHCR,
                      TrialRate = 0.2, Ref = 1,
                      ControlPointsIndex = c(0.5, 1, 2),
                      ControlPointsRate  = c(0, 1, 0.7),
                      RampType = 'smooth')
plot(Level3, Rate3, type = "l", lwd = 2,
     xlab = "Current status (Est / Ref)", ylab = "Rate")
lines(Level3, Rate3Smooth, lwd = 2, col = "dodgerblue")
abline(v = c(0.5, 1, 2), lty = 3, col = "grey50")
legend("topleft", c("linear", "smooth"), col = c("black", "dodgerblue"), lwd = 2)
```

---

`Imp` *Imp Constructor and Accessor*

---

## Description

Construct an imp object defining implementation error for TAC, effort, and size-based management controls, or extract the `Imp` slot from an enclosing object.

## Usage

```
Imp(Name = NULL, TAC = NULL, Effort = NULL, Size = NULL, Misc = list())

Imp(x) <- value

TACImp(x)

TACImp(x) <- value
```

## Arguments

| | |
|---|---|
| `Name` | `character(1)` or an S4 object. Unique identifier for this implementation error model. Default `NULL`.<br>If `Name` is an om object, `Imp()` returns `Name@Imp` rather than constructing a new object. See *Pass-Through Access* in Details. |

| | |
|---|---|
| `TAC` | An impslot object, or `NULL` (default). Implementation error for total allowable catch recommendations. When `NULL`, an empty impslot is created via `ImpSlot()`. |
| `Effort` | An impslot object, or `NULL` (default). Implementation error for effort-based controls. When `NULL`, an empty impslot is created via `ImpSlot()`. |
| `Size` | An impslot object, or `NULL` (default). Implementation error for size-based regulations (e.g. minimum landing size), via its `Compliance` slot only (see Details). When `NULL`, an empty impslot is created via `ImpSlot()`. |
| `Misc` | List. Miscellaneous additional objects. Default `list()`. |
| `x` | An om object, for `Imp<-`; an imp object, for `TACImp()`/`TACImp<-`. |
| `value` | For `Imp<-`: a single imp object (replicated across every complex/fleet), a flat list of imp objects of length `nFleet` (replicated across every complex), or a nested list `[[complex]][[fleet]]` of imp objects. For `TACImp<-`: an impslot object to assign to the `TAC` slot. |

## Details

### Pass-Through Access:

When `Name` is an om object, `Imp()` acts as an accessor and returns `Name@Imp` directly. This allows a consistent interface for both construction and retrieval:

```
imp_obj <- Imp(om)          # extract from OM
Imp(om) <- Imp("MyImp")    # assign to OM
```

### Sub-object initialisation:

When any of `TAC`, `Effort`, or `Size` is `NULL`, an empty impslot object is created automatically so that the returned imp object is always fully populated and valid.

### What is consumed during simulation:

`TAC@Mean`/`SD`/`Error` and `Effort@Mean`/`SD`/`Error` are applied as a multiplicative implementation-error factor to the advised TAC/effort before effort-solving (`Error` is generated from `Mean`/`SD` if not supplied directly – see `PopulateImpSlot()`). `TAC@Compliance` and `Effort@Compliance` (`[Sim x Year]` within each fleet/complex's `impslot`, in `[0, 1]`) govern how a fleet reconciles competing TAC/effort recommendations across multiple stocks or complexes – meaningless (and not consulted) for a single-stock/single-complex OM, since there is nothing to reconcile against.

For `TAC@Compliance`, a fleet that cannot satisfy every complex's TAC at once must trade off falling short of one against exceeding another. `Compliance` sets how heavily exceeding this complex's TAC is penalised, relative to falling short of it:

- `0`: exceeding this TAC costs nothing, so the fleet effectively ignores it and fishes to satisfy its other quotas.
- `0.5`: exceeding and falling short are weighted equally.
- `0.7` (the default when unset): exceeding is penalised about 2.3 times more heavily than falling short, so the fleet stops at whichever quota binds first rather than overshooting it to fill the others.
- Approaching `1`: an effective hard choke.

The default is asymmetric because a TAC is a cap rather than a target: landing under quota is routine, while exceeding it is a regulatory breach. Set `0.5` explicitly if you want the symmetric treatment.

`Size@Compliance` (in `[0, 1]`) is the fraction of the fleet that adopts a newly-advised size-based regulation (a change to `Advice@Retention` and/or `Advice@Selectivity`) in the year it changes; the remaining `1 - Compliance` fraction continues under the prior curve. Missing/`NA` defaults to `1` (full, immediate adoption). `Size@Mean/SD/Error` are not consumed.

### Value

- `Imp()` returns a new imp object when `Name` is `NULL` or a character string.
- `Imp()` returns `Name@Imp` (an imp object) when `Name` is an om object.
- `Imp<-` returns `x` with the `Imp` slot replaced.
- `TACImp()`, `Effort()`, `Size()` return the corresponding impslot slot from an imp object `x`. (`TAC()` is not used for this purpose because it is already defined elsewhere in the package as a function for running MPs against a data object.)
- `TACImp<-`, `Effort<-`, `Size<-` return `x` with the named slot replaced.

### See Also

- imp for the class definition.
- impslot and `ImpSlot()` for the sub-object constructor and slot-level accessors.
- `TACImp()`, `Effort()`, `Size()` for extracting the `TAC`, `Effort`, and `Size` sub-objects from an imp object.
- `OM()` for the operating model constructor.
- `Advice()` for the advice object connected to implementation.
- `ConvertImp()` for converting legacy implementation objects.

Other imp: `FullComplianceImp`, `ImpSlot()`, `OverageImp`, `PartialSizeComplianceImp`, `UnderageImp`, `imp-class`, `impslot-class`

---

## `imp-class` *Implementation Error*

---

### Description

The imp defines implementation error associated with management advice, controlling how management recommendations (TAC, effort, and size limits) are imperfectly applied in the operating model.

### Details

`TAC` and `Effort` each use `Mean/SD/Error`: a multiplicative bias and its stochastic realisation applied to the advised TAC/effort. `Size` instead uses only `Compliance`: the fraction of the fleet that adopts a newly-advised size-based regulation (a change to `Advice@Retention` and/or `Advice@Selectivity`) in the year it changes, with the remainder of the fleet continuing under the prior (status-quo) Retention/Selectivity curve.

An imp object can be attached to an om object and retrieved with `Imp(om)`.

### Slots

`Name character` or `NULL`. Name of the implementation error model. See `Imp()`.

`TAC` An impslot object. Implementation error for total allowable catch recommendations, via `Mean/SD/Error`. See `Imp()`.

`Effort` An impslot object. Implementation error for effort-based controls, via `Mean/SD/Error`. See `Imp()`.

`Size` An impslot object. Implementation error for size-based regulations (minimum legal size etc.), via `Compliance` only. See `Imp()`.

`Misc` List. Miscellaneous additional objects.

### See Also

- `Imp()` for the constructor and accessor.
- `ImpSlot()` for the sub-object constructor and slot-level accessors.
- `OM()` for the operating model constructor.
- `Advice()` for the advice object connected to implementation.
- `ConvertImp()` for converting legacy implementation objects.

Other imp: `FullComplianceImp`, `Imp()`, `ImpSlot()`, `OverageImp`, `PartialSizeComplianceImp`, `UnderageImp`, `impslot-class`

---

## `Imp-legacy-class` *Class* `'Imp'`

---

### Description

An operating model component that specifies the degree of adherence to management recommendations (Implementation error)

### Slots

`Name` The name of the Implementation error object. Single value. Character string.

`Name` The name of the Implementation error object. Single value. Character string.

`TACFrac` Mean fraction of recommended TAC that is actually taken. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the mean TAC fraction obtained across all years of that simulation, and a yearly TAC frac is drawn from a log-normal distribution with the simulation mean and a coefficient of variation specified by the value of `TACSD` drawn for that simulation. If the value drawn is greater than 1 the amount of catch taken is greater than that recommended by the TAC, and if it is less than 1 the amount of catch taken is less than that recommended by the TAC. Positive real numbers.

`TACSD` Log-normal coefficient of variation in the fraction of recommended TAC that is actually taken. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is used, along with the `TACFrac` drawn for that simulation, to create a log-normal distribution that yearly values specifying the actual amount of catch taken are drawn from. Positive real numbers.

`TAEFrac` Mean fraction of recommended TAE that is actually taken. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the mean TAE fraction obtained across all years of that simulation, and a yearly TAE frac is drawn from a log-normal distribution with the simulation mean and a coefficient of variation specified by the value of `TAESD` drawn for that simulation. If the value drawn is greater than 1 the amount of effort employed is greater than that recommended by the TAE, and if it is less than 1 the amount of effort employed is less than that recommended by the TAE. Positive real numbers.

`TAESD` Log-normal coefficient of variation in the fraction of recommended TAE that is actually taken. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is used, along with the `TAEFrac` drawn for that simulation, to create a log-normal distribution that yearly values speciying the actual amount of efort employed are drawn from. Positive real numbers.

`SizeLimFrac` Mean fraction of recommended size limit that is actually retained. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the mean size limit fraction obtained across all years of that simulation, and a yearly size limit fraction is drawn from a log-normal distribution with the simulation mean and a coefficient of variation specified by the value of `SizeLimSD` drawn for that simulation. If the value drawn is greater than 1 the size of fish retained is greater than that recommended by the size limit, and if it is less than 1 the amount of size of fish retained is less than that recommended by the size limit. Positive real numbers.

`SizeLimSD` Log-normal coefficient of variation in the fraction of recommended size limit that is actually retained. For each historical simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is used, along with the `SizeLimFrac` drawn for that simulation, to create a log-normal distribution that yearly values speciying the actual fraction of the size limit retained are drawn from. Positive real numbers.

### Objects from the Class

Objects can be created by calls of the form `new('Imp')#'`

### Author(s)

T. Carruthers and A. Hordyk

### Examples

```
showClass('Imp')
```

---

`ImportBAM` *Import BAM Output into an Operating Model*

---

### Description

Imports output from the Beaufort Assessment Model (BAM) and converts it into an operating model (`OM()`) object. Stock and fleet objects are populated from BAM parameters and time series, including selectivity, retention, discard mortality, recruitment deviations, and stock-recruitment relationships.

### Usage

```
ImportBAM(
  Stock = "Red Snapper",
  nSim = 48,
  pYear = 30,
  StockName = NULL,
  Source = NULL,
  DiscMortDF = NULL,
  DiscFleets = NULL,
  DiscSelFleets = NULL,
  RetSelFleets = NULL,
  incData = TRUE,
  SurveyNames = NULL,
  UnitsLandings = NULL,
  UnitsDiscards = NULL,
  MPStartYear = NULL,
  InterimAdvice = NULL,
  populate = TRUE,
  silent = FALSE
)

GetBAMOutput(Stock = "Red Snapper", type = c("rdat", "dat"))

ListBAMStocks(type = c("rdat", "dat"))
```

### Arguments

| | |
|---|---|
| `Stock` | Character string matching a stock name available in `bamExtras` (e.g., `'Red Snapper'`), or a list of BAM output objects containing elements `rdat` and `dat`. |
| `nSim` | Integer. Number of simulations. Default `48`. |
| `pYear` | Integer. Number of projection years. Default `30`. |
| `StockName` | Optional character string to override the species name taken from the BAM output. |
| `Source` | Optional. Character string specifying the source (usually a link) for the OM. |
| `DiscMortDF` | Optional data.frame with columns `Fleet`, `Value`, and `Year` specifying discard mortality rates by fleet and time block. If `NULL`, values are extracted directly from BAM parameters. |
| `DiscFleets` | Optional named character vector mapping retain fleet names to their corresponding discard fleet names in the BAM output, for cases where naming conventions differ. |
| `DiscSelFleets` | Optional named character vector mapping retain fleet names to the selectivity series to use for discards. |
| `RetSelFleets` | Optional named character vector mapping retain fleet names to alternative retention selectivity series. |
| `incData` | Logical. Include `Data` in the input? Default = `TRUE` |
| `SurveyNames` | A character vector of user-facing names for survey indices detected in `BAMdata`. Must have the same length as the number of survey indices found (i.e. indices not matched to any fleet name). Use this to override the default names derived from column names. Used by `ImportBAMData()`. |
| `UnitsLandings` | A character vector of length equal to the number of landings fleets detected. Each element must be one of:<br>• `"1000 lb"` — values will be converted from thousands of pounds to kg.<br>• `"1000 n"` — values will be multiplied by 1,000 (number in thousands). Used by `ImportBAMData()`. |
| `UnitsDiscards` | A character vector of length equal to the number of discard fleets detected. Accepts the same values as `UnitsLandings`. Used by `ImportBAMData()`. |
| `MPStartYear` | Numeric or `NULL`. First calendar year in which MPs are applied; see `OM()`/om. Default `NULL`. |
| `InterimAdvice` | A `data.frame` or `NULL`. Fixed or stochastic TAC/Effort values for years before `MPStartYear`; see `InterimAdvice()`. Default `NULL`. |
| `populate` | Logical. If `TRUE` (default), calls `PopulateOM()` to populate the OM after construction. |
| `silent` | Logical. If `TRUE`, suppresses console messages. Default `FALSE`. |
| `type` | Character. Format of BAM output to retrieve: `'rdat'` (default) returns the R data object; `'dat'` returns the raw input data. |

### Details

The numbers-at-age in the first historical year are the equilibrium unfished numbers-at-age multiplied by `SRR@RecDevInit` (the BAM initial-age deviations, `log.Nage.dev`) and `Depletion@Misc$InitialAtAge` (the remaining age-specific depletion, e.g. from initial fishing mortality). The dynamic unfished conditions start from the equilibrium unfished numbers-at-age multiplied by `SRR@RecDevInit` only.

### Value

`ImportBAM()` returns a populated OM object. `GetBAMOutput()` returns a BAM output list of class `BAMdata` (for `type = 'rdat'`). `ListBAMStocks()` returns a character vector of available stock names.

`ImportBAM()` requires the `bamExtras` package, which can be installed with `pak::pkg_install('nikolaifish/bamE`

The `DiscMortDF` argument accepts a data.frame with columns:

**Fleet** Fleet name matching the retain fleet names in the OM.

**Value** Discard mortality rate (0–1).

**Year** The year which the value takes effect (i.e., the value applies to all years after this year).

When `DiscFleets`, `DiscSelFleets`, or `RetSelFleets` are provided, they should be named character vectors where names are the retain fleet names and values are the corresponding BAM fleet/selectivity names to use instead of the defaults.

### See Also

`ImportBAMData()`

---

## `ImportBAMData` *Import BAM Assessment Data into an Operating Model*

---

### Description

Populates the `Data` and `Obs` slots of an om object with observed time-series data extracted from a BAM stock assessment. Landings, discards, fleet CPUE indices, and independent survey indices are imported and stored in the appropriate data sub-objects. Survey selectivity schedules are stored in the corresponding obs objects so that the data object holds the observed values and the obs object records how those data were generated.

### Usage

```
ImportBAMData(
  OM,
  BAMdata,
  SurveyNames = NULL,
  UnitsLandings = NULL,
  UnitsDiscards = NULL,
  DiscFleets = NULL
)
```

### Arguments

| | |
|---|---|
| `OM` | An om object. |
| `BAMdata` | A named list returned by `GetBAMOutput()` or directly from the `bamExtras` pacakge. |
| `SurveyNames` | A character vector of user-facing names for survey indices detected in `BAMdata` (i.e. `U.*` columns not matched to any fleet name, excluding discard indices). Must have the same length as the number of such surveys detected. If `NULL`, names are derived from the BAM column names directly. |
| `UnitsLandings` | A named character vector or list mapping fleet names to unit strings. Names must be a subset of `FleetNames()`; fleets not present are assigned `NA` units with no conversion applied. Supported unit strings:<br>• `"1000 lb"`: converts from thousands of pounds to kg.<br>• `"1000 n"`: multiplies by 1,000 (numbers in thousands). If `NULL`, `"1000 lb"` is assumed for all detected landings fleets with a warning. |
| `UnitsDiscards` | A named character vector or list following the same convention as `UnitsLandings`, applied to discard columns (`D.*`). When `DiscFleets` is supplied, names here must be the final (retained-fleet) names it maps to, not the raw BAM discard stub - see `DiscFleets`. |
| `DiscFleets` | A named character vector mapping final fleet names (i.e. names appearing in `FleetNames()`) to raw BAM discard column stubs, in the form `FinalName = "F.RawStub.D"` (the middle component, `"RawStub"`, is extracted and matched against detected `D.*` columns). Used when a BAM assessment's discard columns use a different fleet code than the corresponding retained fleet (e.g. a discard-only code distinct from the landings code) - matching raw discard columns are remapped to their final fleet name before import, so `UnitsDiscards` and the resulting `@Discards` data are keyed by final fleet names throughout, consistently with `@Landings`. If `NULL` (default), discard columns are matched directly against `FleetNames()` with no remapping. |

### Details

#### Fleet matching:

Fleet CPUE columns are identified by searching `U.*` column names in `BAMdata$t.series` for strings matching `FleetNames()`. Matched columns are stored in `@CPUE`; unmatched `U.*` columns (excluding those containing `.D.`, which are discard CPUE) are stored in `@Survey`.

#### Unit conversion:

Conversion is applied fleet-by-fleet using `UnitsLandings` and `UnitsDiscards`. Fleets present in the OM but absent from the units argument receive `NA` units and their values are left unconverted. For discards, any raw BAM column stub named in `DiscFleets` is remapped to its final fleet name before this matching happens (see `DiscFleets`).

**Age and length composition:**

Composition data are not yet imported. Planned sources are `BAMdata$comp.mats$*age*` (age composition) and `BAMdata$comp.mats$lcomp.*.ob` (length composition). When implemented, length composition must follow the compdata convention used by `LandingsAtSize`/`DiscardsAtSize`: `Classes` is a named list, one vector per fleet, since fleets are not required to share a length-bin grid (see compdata for the fleet-relative-position convention on `Value`'s Class dimension). Age composition can keep the flat shared `Classes` vector, since ages are shared across fleets.

## Value

The `OM` object with two slots updated:

- `OM@Data[[1]]`: a data object containing:
  - `@CPUE`: an indicesdata object with fleet-matched CPUE indices and CVs.
  - `@Survey`: an indicesdata object with unmatched (non-fleet) survey indices and CVs.
  - `@Landings`: a catchdata object with landings by fleet, converted to metric units where specified.
  - `@Discards`: a catchdata object with discards by fleet, converted to metric units where specified.
- `OM@Obs[[1]]`: updated with a named obs entry for each survey, containing the selectivity-at-age schedule matched from `BAMdata$sel.age`. If no matching selectivity is found for a survey, full selection across all age classes is assumed and a warning is issued.

## See Also

`GetBAMOutput()`, `Data()`, `CatchData()`, `Obs()`

## Examples

```
## Not run:
bam <- GetBAMOutput('Red Snapper')
OM <- ImportBAMData(
  OM            = myOM,
  BAMdata       = bam,
  SurveyNames   = c("VideoSurvey", "TrawlSurvey"),
  UnitsLandings = c(CommFleet = "1000 lb", RecFleet = "1000 lb"),
  UnitsDiscards = c(CommFleet = "1000 n"),
  DiscFleets    = c("F.CommFleet.D", "F.RecFleet.D")
)

## End(Not run)
```

`ImportiSCAM` *Import an iSCAM Model into an Operating Model*

## Description

Import an iSCAM (Integrated Statistical Catch-Age Model) MPD fit and convert it into an `OM()` object, targeting the current `om/hist/data` class system (unlike the legacy `iSCAM2OM()`, which targets the older `OM/Data` classes).

## Usage

```
ImportiSCAM(
  iSCAMdir,
  Name = "Imported iSCAM Model",
  nSim = 48,
  pYear = 30,
  Agency = "",
  Author = "",
  Email = "",
  Region = "",
  Latitude = numeric(),
  Longitude = numeric(),
  Sponsor = "",
  StockName = "iSCAM Stock",
  CommonName = NULL,
  Species = NULL,
  FleetName = "iSCAM Fleet",
  Interval = 1,
  DataLag = 0,
  MPStartYear = NULL,
  InterimAdvice = NULL,
  silent = FALSE,
  Populate = TRUE
)
```

## Arguments

| | |
|---|---|
| `iSCAMdir` | Character string giving the path to a directory containing iSCAM output files, or a list already returned by `load.iscam.files()`. |
| `Name` | Character string giving the name of the imported operating model. |
| `nSim` | Integer specifying the number of simulations. iSCAM's MPD fit is a single point estimate, so every simulation starts from the same biology/exploitation history; simulation-to-simulation variability comes from `Simulate()`'s own process/observation error, not from iSCAM. |
| `pYear` | Integer specifying the number of projection years. |

`Agency`, `Author`, `Email`, `Region`, `Latitude`, `Longitude`, `Sponsor`
: Character/ numeric metadata stored on the resulting `OM`, matching `ImportSS()`.

`StockName`, `FleetName`
: Character. Names for the single stock/fleet built from the iSCAM model.

`CommonName`, `Species`
: Optional character metadata for the stock.

`Interval`
: Integer specifying the number of years between management actions.

`DataLag`
: Integer specifying the observation lag in years.

`MPStartYear`
: Numeric or `NULL`. First calendar year in which MPs are applied; see `OM()`/om. Default `NULL`.

`InterimAdvice`
: A `data.frame` or `NULL`. Fixed or stochastic TAC/Effort values for years before `MPStartYear`; see `InterimAdvice()`. Default `NULL`.

`silent`
: Logical; if `TRUE`, suppress informational output.

`Populate`
: Logical; if `TRUE` (default), populate the OM using `PopulateOM()`. If `FALSE`, return the partially constructed OM.

## Details

`ImportiSCAM()` reads iSCAM output files from `iSCAMdir` (via `load.iscam.files()`, the same reader used by the legacy `iSCAM2OM()`) and constructs an `OM()` object:

- Growth (`Length`), weight-at-age, maturity-at-age, and natural mortality-at-age are read directly from the MPD fit (`replist$mpd$M`/`replist$mpd$d3_wt_mat`/`replist$mpd$ma`, and `replist$dat$linf`/`k`/`to` for the von Bertalanffy growth curve).
- The stock-recruit relationship uses iSCAM's fitted steepness and R0 (`replist$mpd$steepness`/`replist$mpd$`
- **Only a single, aggregate fleet is built.** iSCAM's MPD reconstruction (`replist$mpd$F`) is a single combined fishing-mortality-at-age series, not broken out by gear – this matches the scope of the legacy `iSCAM2OM()`/`iSCAM2Data()`, which also collapse multi-gear catch into one total rather than building per-gear `Fleet` dynamics. The fleet's relative selectivity-at-age is derived by normalising `replist$mpd$F` within each year (`F_a,y / max_a(F_a,y)`), and its historical `Effort` is the resulting apical F series, normalised to its historical maximum – i.e. effort is assumed proportional to apical F under constant catchability. **This approximates iSCAM's fitted exploitation history; it is not a byte-for-byte reconstruction of iSCAM's own (state-space) historical trajectory**, since `Simulate()` derives numbers-at-age from this effort series and the biology above, rather than reusing `replist$mpd$N` directly.
- `ImportiSCAMData()` (called internally, or usable standalone) populates `OM@Data` with aggregate landings and any abundance indices (`replist$dat$indices`). Age/size composition data is not yet imported – like `ImportBAMData()`'s equivalent gap, this is a documented TODO; see compdata for the per-fleet convention to follow.

**Not currently supported**: delay-difference models (`replist$mpd$F_dd`) and MCMC-based import (multiple simulation replicates drawn from `read.mcmc()`) – both are handled by the legacy `iSCAM2OM()` but are out of scope for this first version; `ImportiSCAM()` errors if either is detected.

### Value

An om object.

### See Also

OM(), PopulateOM(), CompareiSCAM(), iSCAM2OM() (legacy), ImportSS(), ImportBAM()

---

## ImportiSCAMData *Import iSCAM Data into a Data Object*

---

### Description

Populates a Data() object with observed time series extracted from an iSCAM assessment: aggregate landings (summed across gears, matching the single-fleet scope of ImportiSCAM()) and abundance indices.

### Usage

```
ImportiSCAMData(
  iSCAMdir,
  Name = "Imported by ImportiSCAMData",
  FleetName = "iSCAM Fleet",
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| iSCAMdir | Character string giving the path to a directory containing iSCAM output files, or a list already returned by load.iscam.files(). |
| Name | Character; name for the Data object. |
| FleetName | Character; name for the single aggregate fleet (must match the fleet name used by ImportiSCAM() when combining the two). |
| silent | Logical; suppress informational output. Default FALSE. |

### Value

A populated Data() object with @Landings and (if present in the iSCAM output) @Survey populated. Age/size composition data is not yet imported – see ImportiSCAM() Details.

### See Also

ImportiSCAM(), Data(), CatchData(), IndicesData()

`ImportSS` *Import a Stock Synthesis Model into an Operating Model*

## Description

Import a Stock Synthesis 3 (SS3) model and convert it into an `OM()` object.

## Usage

```
ImportSS(
  SSDir,
  Name = "Imported SS3 Model",
  nSim = 48,
  pYear = 30,
  Agency = "",
  Author = "",
  Email = "",
  Region = "",
  Latitude = numeric(),
  Longitude = numeric(),
  Sponsor = "",
  StockName = NULL,
  CommonName = NULL,
  Species = NULL,
  FleetNames = NULL,
  Interval = 1,
  DataLag = 0,
  MPStartYear = NULL,
  InterimAdvice = NULL,
  LengthUnits = NULL,
  WeightUnits = NULL,
  ROUnits = NULL,
  CatchUnits = c("SS3", "Biomass"),
  AllocationYears = 3,
  silent = FALSE,
  Populate = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `SSDir` | Character string giving the path to a directory containing SS3 output files, a vector of such paths, or a list of SS3 report objects generated by `ImportSSReport()` |
| `Name` | Character string giving the name of the imported operating model. |
| `nSim` | Integer specifying the number of simulations. Ignored if multiple SS3 runs are detected; in that case `nSim` is set to the number of SS3 runs imported. |

| | |
|---|---|
| pYear | Integer specifying the number of projection years. |
| Agency | Character string giving the management or scientific agency. |
| Author | Character string giving the model author(s). |
| Email | Character string giving the author(s) email address. |
| Region | Character string describing the geographic region. |
| Latitude | Numeric vector giving stock latitude(s). |
| Longitude | Numeric vector giving stock longitude(s). |
| Sponsor | Character string giving the project sponsor. |
| StockName | Optional character vector of stock names. Length must match the number of stocks inferred from the SS3 model. |
| CommonName | Optional character vector of common names for each stock. |
| Species | Optional character vector of species names for each stock. |
| FleetNames | Optional character vector of fleet names. If NULL, fleet names are inferred from SS3 output. |
| Interval | Integer specifying the number of years between management actions. |
| DataLag | Integer specifying the observation lag in years. |
| MPStartYear | Numeric or NULL. First calendar year in which MPs are applied; see OM()/om. Default NULL. |
| InterimAdvice | A data.frame or NULL. Fixed or stochastic TAC/Effort values for years before MPStartYear; see InterimAdvice(). Default NULL. |
| LengthUnits | Character string giving the physical unit of length measurements in the SS3 model (e.g., "cm", "mm", "inch"; see ValidUnits()). Default "cm". |
| WeightUnits | Character string giving the physical unit of weight measurements in the SS3 model (e.g., "kg", "g", "lb"; see ValidUnits()). Not reported by SS3/r4ss; "kg" is the conventional pairing with LengthUnits = "cm". Default "kg". |
| R0Units | Numeric scaling factor for unfished recruitment (R0) and related numbers-at-age quantities; see Units in srr. SS3 does not export the numeric scale of its population numbers (e.g. individuals vs. thousands) as metadata, so this cannot be inferred and must be supplied by the user if not 1 (absolute numbers of fish). Default 1. |
| CatchUnits | Character. Units of the imported catch data (Data@Landings, Data@Discards), and therefore of the catch data simulated from the OM. 'SS3' (default) uses the catch units of each SS3 fleet (biomass or numbers). 'Biomass' imports the catch of every fleet in biomass: for fleets with catch in numbers, this is the catch biomass predicted by SS3 (ret_bio, kill_bio in the CATCH report), which depends on the estimated selectivity and weight-at-age. Use 'Biomass' when a TAC for several fleets is set in biomass, or for MPs that need catch in biomass (e.g. SurplusProduction()). |
| AllocationYears | Integer specifying the number of most recent historical years used to calculate OM@FleetAllocation: each fleet's share of total removals biomass |

| | |
|---|---|
| | (landings + discards), used to split an aggregate TAC across fleets under multi-fleet TAC-based MPs. Averaged over this many years ending at the last historical year. Default `3`. |
| `silent` | Logical; if `TRUE`, suppress informational output during import. |
| `Populate` | Logical; if `TRUE` (default), populate the OM using `PopulateOM()`. If `FALSE`, return the partially constructed OM. |
| `...` | Additional arguments passed to SS3 import functions. |

## Details

`ImportSS()` reads SS3 output files from `SSDir` and constructs an `OM()` object. Biological processes, fleet structure, and historical time series are inferred directly from the SS3 report output and mapped into the corresponding OM slots.

- Multiple simulation replicates are supported when multiple SS3 runs are detected.
- Optional metadata describing authorship, region, and stock identity are stored in the resulting OM.
- By default, the OM is fully populated using `PopulateOM()`.
- Set `Populate = FALSE` to return the partially constructed OM.
- The numbers-at-age in the first historical year are the SS3 virgin numbers-at-age multiplied by `SRR@RecDevInit` and `Depletion@Misc$InitialAtAge`. `SRR@RecDevInit` holds the SS3 initial-age recruitment deviations, scaled by any recruitment offset SS3 applies to the initial equilibrium without fishing (e.g. a regime parameter). `Depletion@Misc$InitialAtAge` holds the remaining age-specific depletion of the initial numbers-at-age, e.g. from initial equilibrium fishing mortality. The dynamic unfished conditions start from the virgin numbers-at-age multiplied by `SRR@RecDevInit` only.

`ImportSSReport()` reads SS3 output generated by Stock Synthesis 3, either from one or more directories or from existing `r4ss` report objects, and returns a standardized list of SS3 report lists.

## Value

- `ImportSS()` returns an om object containing stock, fleet, data, observation, and allocation components derived from the SS3 model.
- `ImportSSReport()` returns a list of SS3 report objects.

## Functions

- `ImportSS()` Convert SS3 output into a fully specified operating model suitable for management strategy evaluation.
- `ImportSSReport()` Read SS3 output from directories or existing `r4ss` report objects and return a standardized list of SS3 reports.

### See Also

- OM()
- PopulateOM()
- CompareSS()
- https://nmfs-ost.github.io/ss3-website/ (Stock Synthesis 3)

### Examples

```
## Not run:
## Path to a directory containing SS3 output
ss_dir <- "path/to/SS3/output"

## Read SS3 output only
reps <- ImportSSReport(ss_dir)

## Build an operating model from SS3 output
OM <- ImportSS(
  SSDir = reps,
  Name = "Example SS3 Import",
  StockName = "ExampleStock",
  FleetNames = c("Commercial", "Recreational")
)

## Inspect imported stocks and fleets
StockNames(OM)
FleetNames(OM)

## End(Not run)
```

---

## ImportSSData *Import SS3 data into a Data object*

---

### Description

Reads a parsed SS3 report and populates a data object with catch, index, and composition data.

### Usage

```
ImportSSData(
  SSDir,
  Name = "Imported by ImportSSData",
  CommonName = "",
  Species = "",
  LengthUnits = "cm",
  CatchUnits = c("SS3", "Biomass"),
  FleetNames = NULL,
```

```
  silent = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| `SSDir` | Character vector of SS3 report directories, or a pre-parsed `r4ss::SS_output` list or `RepList` object returned by `ImportSSReport()`. |
| `Name` | Character; name for the `Data` object |
| `CommonName` | Character; common name of the species |
| `Species` | Character; scientific name of the species |
| `LengthUnits` | Character string giving the physical unit of length composition bins (e.g., `"cm"`, `"mm"`, `"inch"`; see `ValidUnits()`). Not reported by SS3/`r4ss`; `"cm"` is the near-universal SS3 convention. Default `"cm"`. |
| `CatchUnits` | Character. `'SS3'` (default) imports the catch of each fleet in its SS3 catch units; `'Biomass'` imports the catch of every fleet in biomass. See `ImportSS()`. |
| `FleetNames` | Optional character vector of names for the SS3 fishing fleets, in SS3 fleet order, used for their catch, CPUE, and composition data. If `NULL`, the SS3 fleet names are used. |
| `silent` | Logical; suppress progress messages (default FALSE) |
| `...` | Additional arguments passed to `ImportSSReport` |

### Value

A populated `Data()` object

---

## `ImportSSReport` *Import SS3 Report directories with optional parallel processing*

---

### Description

Import SS3 Report directories with optional parallel processing

### Usage

```
ImportSSReport(SSDir, parallel = FALSE, silent = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `SSDir` | Character vector of SS3 report directories, or a pre-parsed `r4ss::SS_output` list or `RepList` object |
| `parallel` | Logical; whether to use parallel processing (default FALSE) |
| `silent` | Logical; suppress progress messages (default FALSE) |
| `...` | Additional arguments passed to `r4ss::SS_output` |

### Value

A named `RepList` object: a list of parsed `r4ss::SS_output` lists, one per SS3 directory, with integer-character names (`"1"`, `"2"`, ...)

### See Also

`ImportSS()`, `SetupParallel()`, `DisableParallel()`

---

`ImpSlot` *ImpSlot Constructor and Accessors*

---

### Description

Construct an impslot object representing implementation uncertainty for a single management control, or access and replace individual slots.

### Usage

```
ImpSlot(Mean = NULL, SD = NULL, Compliance = NULL, Error = NULL, Misc = list())
```

### Arguments

| | |
|---|---|
| `Mean` | Numeric array or list of arrays, or `NULL` (default). Mean implemented fraction of the management recommendation (e.g. `1` = perfect compliance with the TAC). When `NULL`, the slot is left at its class default. |
| `SD` | Numeric array or list of arrays, or `NULL` (default). Standard deviation of implementation error around `Mean`. When `NULL`, the slot is left at its class default. |
| `Compliance` | Scalar, length-2 uniform bounds `c(lower, upper)`, a `[Sim x Year]` array, or `NULL` (default). Meaning depends on which control (`TAC`, `Effort`, or `Size`) this `impslot` is attached to (see Details). Values should be in $[0, 1]$. |
| `Error` | Numeric array or list of arrays, or `NULL` (default). Realised implementation error, a per simulation/year multiplicative factor applied to the advised TAC or effort before effort-solving. When supplied directly, `Mean` and `SD` are stored but `Error` is not re-derived from them during simulation. When `NULL`, `Error` is stochastically generated from `Mean` and `SD` at simulation time (see `PopulateImpSlot()`). |
| `Misc` | List. Miscellaneous additional objects. Default `list()`. |

### Details

#### Mean, SD, and Error:

`Mean` and `SD` together parameterise a lognormal distribution for `Error`, a general implementation-error multiplier applied to the advised TAC/effort regardless of how many stocks are in the OM (e.g. `Mean = 0.9` models a fleet that implements 90% of the recommended TAC on average). See `PopulateImpSlot()` for exactly how `Mean`/`SD` expand to `Error`.

**Compliance:**

For `TAC/Effort`, only meaningful for multi-stock/multi-complex OMs, where it governs how a fleet reconciles competing TAC or effort recommendations across stocks.

For `TAC` it sets how heavily exceeding this complex's TAC is penalised relative to falling short of it: `0` makes exceeding it free, `0.5` weights the two equally, and values approaching `1` act as a hard choke. Unset defaults to `0.7`, penalising overshoot about 2.3 times more than undershoot, since a TAC is a cap rather than a target. See `Imp()`.

Values close to `1` are not recommended: the penalty on falling short is bounded while the penalty on exceeding is not, so a sufficiently large weight makes zero effort the cheapest option and the fleet stops fishing altogether.

For `Size`, it is the fraction of the fleet adopting a newly-advised size regulation.

## Value

- `ImpSlot()` returns a new impslot object.
- `Mean()`, `SD()`, `Compliance()`, `Error()` return the corresponding slot from an impslot object `x`.
- `Mean<-`, `SD<-`, `Compliance<-`, `Error<-` return `x` with the named slot replaced.

## See Also

- impslot for the class definition.
- `Imp()` for the enclosing constructor and accessor.
- `TACImp()`, `Effort()`, `Size()` for slot-level accessors on imp objects.

Other imp: `FullComplianceImp`, `Imp()`, `OverageImp`, `PartialSizeComplianceImp`, `UnderageImp`, `imp-class`, `impslot-class`

## Examples

```
# Empty impslot
sl <- ImpSlot()

# With Mean and SD specified
sl <- ImpSlot(Mean = 1, SD = 0.1, Compliance = 0.9)

# Supplying Error directly (bypasses stochastic generation)
sl <- ImpSlot(Error = array(rnorm(100, 1, 0.05), dim = c(10, 10)))
```

---

`IndexFitTable` *Summarize Fitted/Estimated Index Observation Error Parameters*

---

## Description

Tabulates the hyperstability (`Beta`), Beta fit diagnostics (`R2`, `PValue`, `Status`), and residual (`SD`, `AC`, `nYears`) parameters stored on each fleet's `CPUE/Survey` indicesobs object. Companion summary for `PlotIndexFit()`.

### Usage

```
IndexFitTable(
  object,
  type = c("CPUE", "Survey"),
  Fleets = NULL,
  Stocks = NULL,
  print = TRUE
)
```

### Arguments

| | |
|---|---|
| `object` | A hist or mse object. |
| `type` | Character. One of `"CPUE"` or `"Survey"`. |
| `Fleets` | Character vector. Restrict to these fleet names. Default `NULL` (all fleets with a populated `type` index). |
| `Stocks` | Character vector. Restrict to these stock/complex names. Default `NULL` (all). |
| `print` | Logical. Print a markdown table to console with `cat()`? Default `TRUE`. |

### Details

Each parameter may vary by simulation. When more than one distinct value is present, the column reports `median [min, max]`; a single fixed value is reported as-is. `R2` and `PValue` come from the per-simulation `Beta` fit (`Misc$BetaFit`, see `EstimateBeta()`) and are `NA` where `Beta` was not freely estimated (fixed by the user or `SimControl(EstimateBeta = FALSE)`). `Status` lists the distinct fit outcomes across simulations (`"estimated"`, `"fixed_user"`, `"fixed_insufficient_data"`, `"fixed_low_variance"`, `"fixed_not_significant"`, `"fixed_bounds"`). `nYears` is the number of years used in the fit. `Beta`, `R2`, `PValue`, and `Status` are placed last since they're only meaningful when Beta estimation was attempted; a column that is `NA` for every row (e.g. `R2`/ `PValue` when `Beta` was fixed everywhere) is dropped from the result.

### Value

returns the `data.frame`.

### See Also

`PlotIndexFit()`, `IndicesObs()`

`IndexMPs` *Empirical Index-Based Management Procedures*

## Description

Management procedures that set a TAC from one or more relative abundance indices (`Survey(Data)` or `CPUE(Data)`, or both).

## Usage

```
IndexRate(
  Data,
  Indices = NULL,
  IndexSource = "auto",
  IndexFreq = NULL,
  IndexWeight = NULL,
  CalibYears = 2,
  Smooth = TRUE,
  ENPMult = 0.3,
  RecentYears = 1,
  TrendYears = NULL,
  TrendHorizon = 1,
  IndexFactor = 1,
  IndexTarget = NULL,
  HCRControlPointsIndex = c(0, 0.5),
  HCRControlPointsRate = c(0, 1),
  RampType = c("linear", "smooth"),
  Responsiveness = 1,
  DeltaDown = c(0.01, 0.5),
  DeltaUp = c(0.01, 0.5),
  TACRange = NULL,
  Allocation = NULL,
  TACType = c("Removals", "Landings"),
  IndexSeasons = NULL,
  tunepar = 1
)

IndexTarget(
  Data,
  Indices = NULL,
  IndexSource = "auto",
  IndexFreq = NULL,
  IndexWeight = NULL,
  CalibYears = 5,
  Smooth = TRUE,
  ENPMult = 0.3,
  RecentYears = 1,
```

```
  TrendYears = NULL,
  TrendHorizon = 1,
  IndexTarget = NULL,
  HCRControlPointsIndex = c(0, 0.25, 0.5, 1, 2),
  HCRControlPointsRate = c(0, 0.125, 0.5, 1, 1.5),
  RampType = c("linear", "smooth"),
  Responsiveness = 1,
  DeltaDown = c(0.01, 0.5),
  DeltaUp = c(0.01, 0.5),
  TACRange = NULL,
  Allocation = NULL,
  TACType = c("Removals", "Landings"),
  IndexSeasons = NULL,
  tunepar = 1
)
```

### Arguments

| | |
|---|---|
| `Data` | A data object. |
| `Indices` | Character (matching `Name`) or integer vector selecting which columns of the chosen `IndexSource` to use. `NULL` (default) uses every index in `IndexSource`. |
| `IndexSource` | Character. `'auto'` (default), or one or both of `'Survey'` and `'CPUE'`. `'auto'` uses the indices in `Data@Survey` if it has any positive observation, otherwise those in `Data@CPUE`, and warns once when the last `Survey` observation is more than 5 years older than the last `CPUE` observation. `'Survey'` or `'CPUE'` uses that slot, and `c('Survey', 'CPUE')` combines the indices of both into a single selection (`Survey` columns first, then `CPUE`). `Indices`, `IndexFreq`, `IndexWeight`, `IndexTarget` and `IndexSeasons` apply to the selected indices in that order; set `IndexSource` explicitly when these are given per index. |
| `IndexFreq` | Non-negative integer vector, one per selected index. How often each index is available in the projection period (`1` = every year, `2` = every 2 years, `0` = not used). Historical values are always used. `NULL` (default) assumes every year. |
| `IndexWeight` | Positive numeric vector, one per selected index, giving each index's weight when combining indices. `NULL` (default) weights indices equally. |
| `CalibYears` | Positive integer. Number of most recent historical years used to calculate the catch-per-index rate (`IndexRate()`, default `2`) or the reference catch (`IndexTarget()`, default `5`). Clamped to the number of historical years available. |
| `Smooth` | Logical. Whether to smooth each selected index with `SmoothSeries()` before calculating its status and trend. `TRUE` (default). `FALSE` uses the observed index values, e.g. for a series too short for a stable loess fit. |
| `ENPMult` | Fraction. Smoothing parameter for `SmoothSeries()`: the number of effective parameters is `length(index) x ENPMult`. Larger values mean less smoothing. Ignored when `Smooth = FALSE`. |

`RecentYears` Positive integer. Number of most recent years averaged, per index, to obtain its current status. Years without an observation are ignored. `1` (default) uses only the terminal year.

`TrendYears` `NULL` (default) disables trend projection. Otherwise an integer `>= 2` giving the number of most recent years (per index) used to estimate a log-linear trend. The status is then projected forward `TrendHorizon` years along that trend before it is compared with `IndexTarget`, so a declining index is treated more cautiously than a stable one at the same level. In `IndexRate()` the projected status is used only in the harvest control rule; the trial TAC is always calculated from the current status.

`TrendHorizon` Positive number. Number of years the estimated trend is extrapolated forward; see `TrendYears`. Default `1`. Ignored when `TrendYears` is `NULL`.

`IndexFactor` Positive number. Multiplier on the calibrated catch-per-index rate. `1` (default) keeps the rate observed over `CalibYears`; e.g. `0.75` sets it 25% lower.

`IndexTarget` `NULL` (default), or a positive number (or vector, one per selected index) giving the target index level, in the units of the (annual) index. Targets of multiple indices are combined with `IndexWeight`. When `NULL`, `Ref` on the selected `IndexSource` object (see `IndicesData()`) is used, and an index with no `Ref` uses its status at the end of the historical period: the historical index, smoothed as set by `Smooth`/`ENPMult`, averaged over the `RecentYears` ending in the last historical year with an observation. The target is fixed across management cycles. `HCRControlPointsIndex` is expressed as a fraction of `IndexTarget`. See Details for choosing a target.

`HCRControlPointsIndex`

Numeric vector of at least 2 non-decreasing values: the control points of a hockey-stick harvest control rule, as fractions of `IndexTarget` (i.e. in units of `status / IndexTarget`). At or below the first control point the catch-per-index rate (`IndexRate()`) or reference catch (`IndexTarget()`) is multiplied by the first value of `HCRControlPointsRate`; at or above the last, by the last value; in between, the multiplier is interpolated (see `RampType`). See `HockeyStickHCR()`.

- `IndexRate()`: default `c(0, 0.5)`. E.g. `c(0.2, 0.8)` with `HCRControlPointsRate = c(0, 1)` closes the fishery below 20% of `IndexTarget` and applies the full rate above 80%. `c(0, 0)` makes the multiplier `1` for any positive status, disabling the harvest control rule.
- `IndexTarget()`: default `c(0, 0.25, 0.5, 1, 2)`. The multiplier applies to the reference catch, so the control points define the TAC as a function of `status / IndexTarget`.

`HCRControlPointsRate`

Numeric vector, same length as `HCRControlPointsIndex`, giving the multiplier at each control point. Default `c(0, 1)` for `IndexRate()` and `c(0, 0.125, 0.5, 1, 1.5)` for `IndexTarget()`.

`RampType` Character. Shape of the harvest control rule between control points: `'linear'` (default) or `'smooth'` (a cubic smoothstep, with zero slope at each control point).

`Responsiveness`
: Positive number. Exponent applied to the proposed TAC ratio, `(trial TAC / previous TAC) ^` before `DeltaDown`/`DeltaUp` are applied. Values below `1` damp the change; `1` (default) applies it in full.

`DeltaDown`, `DeltaUp`
: Numeric vector, length 2 (`c(min, max)`). Limits on the fractional TAC change from the previous TAC, for decreases and increases respectively. Changes larger than `max` are capped at `max`; changes smaller than `min` leave the TAC unchanged. See `ConstrainTAC()`.

`TACRange`
: Numeric vector, length 2 (`c(min, max)`). Absolute bounds on the TAC, applied after `DeltaDown`/`DeltaUp`. `NULL` (default) uses `c(0, 100 x max(annual catch in Data` with catch of type `TACType`, which in practice imposes no limit.

`Allocation`
: `NULL` (default), or a non-negative numeric vector of length `nFleet` giving the fraction of the TAC allocated to each fleet (normalised to sum to `1`). `NULL` returns a single stock-wide TAC, which the framework then splits across fleets using `FleetAllocation(OM)` (see `Advice()`).

`TACType`
: Character. Whether the TAC applies to `'Removals'` (default; landings plus discards) or `'Landings'`. Sets the catch used for the catch-per-index rate (`IndexRate()`) or reference catch (`IndexTarget()`), for the previous TAC in the first management cycle (see `LastTAC()`), and for the default `TACRange`. See `Advice()`.

`IndexSeasons`
: Seasonal data only. `NULL` (default) averages each index over every season of the year. Otherwise an integer vector of seasons applied to every selected index, or a list with one element (integer vector, or `NULL` for every season) per selected index. See `AnnualData()`. Ignored when `Data@Seasons = 1`.

`tunepar`
: Positive number used to tune the MP (see `TuneMP()`); larger values give higher catches. `1` (default) applies the MP as specified. It multiplies the catch-per-index rate (`IndexRate()`) or the reference catch (`IndexTarget()`).

## Details

Both MPs estimate the current status of each selected index as its (optionally loess-smoothed, see `Smooth`) value averaged over the `RecentYears` most recent years. When `TrendYears` is set, that status is projected forward `TrendHorizon` years along a log-linear trend before it is compared with the index target. Per-index values are combined with `IndexWeight`.

The harvest control rules of both MPs are expressed relative to `IndexTarget`. With no `IndexTarget` or `Ref`, the target is the index at the end of the historical period, which is not a biological reference point: for a stock that is depleted at that time, the target is the depleted level and the harvest control rule does not reduce the TAC until the index falls further. Supply `IndexTarget` or `Ref` (e.g. the index level corresponding to BMSY) to set the harvest control rule relative to stock status.

Each MP then calculates a trial TAC (see the sections below). If the trial TAC is negative, infinite, or `NA`, the previous TAC (`LastTAC()`) is kept. The proposed change from the previous TAC is raised to the power `Responsiveness` and then constrained by `DeltaDown`/`DeltaUp`/`TACRange`.

For seasonal data (`Data@Seasons > 1`), both MPs first aggregate `Data` to calendar years with `AnnualData()`: catches are summed over seasons and each index is averaged over the seasons in `IndexSeasons`. All arguments given in years (`CalibYears`, `RecentYears`, `TrendYears`, `TrendHorizon`, `IndexFreq`) therefore refer to calendar years, and the TAC is annual.

**IndexRate:**

`IndexRate()` fishes at a catch-per-index rate, so the TAC follows the index.

The rate is the mean catch of type `TACType` (summed over fleets) over the last `CalibYears` historical years divided by the mean unsmoothed index over the same years, multiplied by `IndexFactor` and `tunepar`. A hockey-stick harvest control rule (`HockeyStickHCR()`), evaluated at `status / IndexTarget`, scales the rate, and the trial TAC is the scaled rate multiplied by the current (non-projected) status.

With the default control points (`HCRControlPointsIndex = c(0, 0.5)`, `HCRControlPointsRate = c(0, 1)`) the full rate applies above half of `IndexTarget`. Below half the target the rate falls linearly to zero, so the trial TAC falls quadratically with the index. `HCRControlPointsIndex = c(0, 0)` disables the harvest control rule, so the trial TAC is `rate x current status` and `IndexTarget` has no effect.

**IndexTarget:**

`IndexTarget()` sets the TAC to keep the index around `IndexTarget`.

The reference catch is the mean catch of type `TACType` (summed over fleets) over the last `CalibYears` historical years, multiplied by `tunepar`. A hockey-stick harvest control rule (`HockeyStickHCR()`), evaluated at `status / IndexTarget`, scales the reference catch to give the trial TAC. With control points passing through `(1, 1)`, the trial TAC is the reference catch when the index is at `IndexTarget`. The trial TAC does not depend on the previous TAC, so TAC changes do not accumulate while the index is away from `IndexTarget`.

With the default control points, the trial TAC as a fraction of the reference catch is:

- `status / IndexTarget` between half the target and the target;
- falling to zero below half the target along a piecewise-linear approximation of a quadratic (`0.125` at a quarter of the target);
- increasing at half that slope above the target, capped at `1.5` from twice the target.

The index settles where the trial TAC equals the stock's production, which is at `IndexTarget` only when the reference catch is sustainable there. Tune `tunepar` (see `TuneMP()`) so that the index fluctuates around `IndexTarget`.

## Value

An advice object with `TAC` and `TACType` set, and `TACUnit` taken from `Data@Landings@Units`.

## See Also

`Advice()`, `CheckCatch()`, `LastTAC()`, `FilterTAC()`, `HockeyStickHCR()`, `ConstrainTAC()`, `SmoothSeries()`, data, advice

`IndexSelectivity` *Get or Set the Selectivity of an Index*

## Description

Access or replace the selectivity-at-age of a CPUE or survey index in an om object, stored in `OM@Obs[[Complex]][[Name]]@Survey` (or `@CPUE`). The replacement form creates the index's obs entry when it does not exist, e.g. for a survey added directly to `OM@Data`.

## Usage

```
IndexSelectivity(x, Name, Type = c("Survey", "CPUE"), Complex = 1)

IndexSelectivity(x, Name, Type = c("Survey", "CPUE"), Complex = 1) <- value
```

## Arguments

| | |
|---|---|
| `x` | An om object. |
| `Name` | `character(1)`. Name of the index, as in `OM@Data`. |
| `Type` | `character(1)`. `"Survey"` (default) or `"CPUE"`. |
| `Complex` | Integer. Index of the stock complex. Default `1`. |
| `value` | Selectivity of the index: `"Biomass"`, `"SBiomass"`, a numeric vector named by age, an array, a list of these by stock, or `NULL` for the selectivity of the fleet with the same name. See `IndicesObs()`. |

## Value

`IndexSelectivity()` returns the index's selectivity specification, or `NULL` when none is set. `IndexSelectivity<-` returns `x` with the selectivity replaced.

## See Also

`AddIndex()`, `IndicesObs()`

## Examples

```
OM <- SingleStockOM
Years <- Years(OM, "Historical")
Value <- matrix(runif(length(Years)), ncol = 1, dimnames = list(Year = Years, Index = "Acoustic"))
Data(OM) <- Data(Survey = IndicesData(Name = "Acoustic", Value = Value))
IndexSelectivity(OM, "Acoustic") <- "SBiomass"
IndexSelectivity(OM, "Acoustic")
```

`IndicesData` *IndicesData Constructor*

## Description

Construct an indicesdata object storing an abundance or biomass index and associated uncertainty.

## Usage

```
IndicesData(
  Name = NULL,
  Value = NULL,
  CV = NULL,
  Units = NULL,
  Ref = NULL,
  RefCV = NULL,
  Timing = 0,
  Misc = list()
)
```

## Arguments

| | |
|---|---|
| `Name` | `character` or `NULL`. Name of the index. Default `NULL`. |
| `Value` | `array` or `NULL`. Observed index values with dimensions `[nYear x nIndex]`. Default `NULL`. |
| `CV` | `array` or `NULL`. Coefficients of variation for the index observations, matching the dimensions of `Value`. Default `NULL`. |
| `Units` | `character` or `NULL`. Units of the index: `"Biomass"`, `"Number"`, or `"Recruitment"`. Default `NULL`. |
| `Ref` | `numeric` or `NULL`. Reference value for each index (e.g., a historical mean or target level), length `nIndex`. Default `NULL`. |
| `RefCV` | `array` or `NULL`. Coefficients of variation for the reference values, matching the dimensions of `Ref`. Default `NULL`. |
| `Timing` | `numeric`. Timing of each observation as a fraction of the time step (0-1), length `nIndex`. Simulated indices are decayed by the mortality accrued up to that point. Default `0`, the start of the time step. |
| `Misc` | `list`. Named list for additional metadata. Default `list()`. |

## Details

### Usage:

All arguments are optional; an empty object is valid and slots are populated later (e.g., when `Simulate()` generates historical data).

### Attaching to a Data Object:

```
CPUE(data)   <- IndicesData(Value = myArray, CV = myCVArray)
Survey(data) <- IndicesData(Value = myArray, CV = myCVArray)
```

The selectivity of an index is set on its observation model (`IndicesObs()`). Use `AddIndex()` to add an index and its selectivity to an operating model in one step.

### Value

An indicesdata object.

### See Also

- indicesdata for the class definition.
- data and `Data()` for the enclosing data object.
- `IndicesObs()` for the observation error structure that generates indicesdata objects during simulation.

Other data: `CatchData()`, `CompData()`, `EffortData()`, `catchdata-class`, `compdata-class`, `effortdata-class`, `indicesdata-class`

### Examples

```
# Empty object
id <- IndicesData()

# With values
yrs <- 2000:2020
val <- array(runif(length(yrs)),
             dim = c(length(yrs), 1),
             dimnames = list(Year = yrs, Index = "Survey1"))
id <- IndicesData(Name  = "Survey1",
                  Value = val,
                  CV    = array(0.2, dim = dim(val), dimnames = dimnames(val)),
                  Units = "Biomass")
```

---

## indicesdata-class *The* indicesdata *S4 Class*

---

### Description

Stores abundance or biomass index observations from fishery-dependent (CPUE) or fishery-independent (Survey) sources, including observation uncertainty, reference values, timing, and selectivity mapping. Used in the `CPUE` and `Survey` slots of a data object. Objects are typically created via `IndicesData()`, which documents all parameters in detail.

### Slots

`Name character` or `NULL`. Name of the index. See `IndicesData()`.

`Value array` or `NULL`. Observed index values with dimensions `[nYear x nIndex]`. See `IndicesData()`.

`CV array` or `NULL`. Coefficients of variation for the index observations, matching the dimensions of `Value`. See `IndicesData()`.

`Units character` or `NULL`. Units of the index (e.g., `"kg/trip"`, `"numbers/tow"`). See `IndicesData()`.

`Ref numeric` or `NULL`. Reference value for each index (e.g., a historical mean or target level). See `IndicesData()`.

`RefCV array` or `NULL`. Coefficients of variation for the reference values, matching the dimensions of `Ref`. See `IndicesData()`.

`Timing numeric`. Timing of each observation as a fraction of the time step (0-1). Simulated indices are decayed by the mortality accrued up to that point; `0` (default) observes the population at the start of the time step. See `IndicesData()`.

`Misc` A named list for any additional index-level metadata.

### See Also

`IndicesData()` for the constructor and full parameter documentation. data for the enclosing data object.

Other data: `CatchData()`, `CompData()`, `EffortData()`, `IndicesData()`, `catchdata-class`, `compdata-class`, `effortdata-class`

---

## `IndicesObs` *IndicesObs Constructor and Accessors*

---

### Description

Construct an indicesobs object defining observation error for a CPUE or survey abundance index, or access and replace individual slots.

### Usage

```
IndicesObs(
  CV = NULL,
  Error = NULL,
  Areas = NULL,
  Units = NULL,
  Selectivity = NULL,
  TruncSD = 2,
  Ref = NULL,
  Beta = NULL,
  AC = NULL,
  qInc = NULL,
  Misc = list()
)
```

### Arguments

`CV` — `numeric` or `NULL`. Coefficient of variation of the lognormal index observation error. Accepted forms:

- `NULL` (default): no observation error is generated unless `Error` is supplied directly.
- Length-1 scalar: a single CV applied uniformly across all simulations and years when generating the `Error` array internally.
- Length-2 vector `c(mean, sd)`: `CV` is itself drawn stochastically — a unique CV per simulation is generated before being used to generate the `Error` array.
- Length-`nSim` named array: used directly, one CV per simulation.

When conditioning on real data, `CV` is replaced by the simulation-specific standard deviation of historical log-residuals estimated by `CalcResidualStats()`. Ignored if `Error` is supplied directly.

`Error` — `numeric` array or `NULL`. Pre-specified lognormal observation error multipliers with dimensions `nSim x nYear` and named dimnames (`Sim`, `Year`). When supplied, `Error` is used directly and `CV` is ignored for error generation. When real index data are provided in `OM@Data`, `Error` is replaced by conditioned values regardless of whether it was user-supplied. Default `NULL`, in which case `Error` is generated from `CV` internally.

`Areas` — `integer` vector or `NULL`. Indices of the spatial areas that contribute to this index (e.g., `c(1, 2)` for a two-area model where the index covers only the first two areas). Default `NULL`, in which case all areas are used.

`Units` — `character(1)` or `NULL`. Units determining how the simulated population is converted to an index value. Must be one of:

- `"Biomass"` (default): sum of numbers-at-age multiplied by weight-at-age over the selected ages and areas.
- `"Number"`: sum of numbers-at-age over the selected ages and areas.
- `"Recruitment"`: abundance of the youngest age class, summed over areas. For seasonal models, the first non-zero age class in each simulation and year is used. Default `NULL` (treated as `"Biomass"`).

`Selectivity` — Selectivity-at-age of this index, multiplied by numbers-at-age before they are summed into the index. Accepted forms:

- `NULL` (default): selectivity of the fleet in `OM@Fleet` with the same name as the index. Indices that are not fleets must set `Selectivity`.
- `"Biomass"`: flat selectivity of 1 for all age classes.
- `"SBiomass"`: maturity-at-age.
- Numeric vector named by age, e.g. `c('3' = 1, '4' = 1, '5' = 1)`. Ages not named have selectivity 0. An unnamed vector must have one value per age class.
- Array with named dimnames including `Age` and optionally `Sim`, `Year`, and `Area`, for selectivity that varies by simulation, year, or area. `Age` must cover the stock's age classes.
- List of any of the numeric forms, named by stock (or in stock order), for indices of multi-stock complexes.

`AddIndex()` adds an index and its selectivity to an operating model in one step; `AddAgeIndices()` adds an age-disaggregated index as one index per age class.

`TruncSD` `numeric(1)`. Number of standard deviations at which to truncate the lognormal residual distribution when generating observation errors for projection years via `GenResiduals()`. Default `2`.

`Ref` `numeric` or `NULL`. Reference index value. Accepted forms:

- `NULL` (default): no reference value.
- Positive scalar: interpreted as the CV of a lognormal distribution; a unique reference value is drawn per simulation as `rlnorm(nSim, mconv(1, Ref), sdconv(1, Ref))` internally.
- Length-`nSim` vector or named array: used directly.

`Beta` `numeric` or `NULL`. Hyperstability/hyperdepletion parameter: `Observed_t = Efficiency * NomIndex_t^Beta * Error_t` (Harley et al. 2001). `Beta < 1` is hyperstable, `Beta > 1` hyperdeplete. Must be positive. Default `NULL` (`Beta = 1`, proportional). A supplied value is used as-is for historical/projected data generation; only `Efficiency` is then fit when conditioning on real data. If `NULL` and `SimControl(EstimateBeta = TRUE)`, `Beta` is instead estimated per simulation by log-linear regression when conditioned – otherwise it stays at `1`.

`AC` `numeric`, `array`, or `NULL`. Lag-1 autocorrelation of index residuals, one value per simulation

`qInc` `numeric` or `NULL`. Annual percentage change in index catchability during projection years (e.g. `2` = 2% per year; negative values indicate declining catchability). Default `NULL` (no change). Accepted forms:

- Scalar: applied to all simulations.
- Length-2 vector `c(lower, upper)`: one value per simulation drawn from a uniform distribution.
- Length-`nSim` vector: used directly.

See "Projected Change in Catchability" below. Also get or set with `qInc()` and `qInc<-`.

`Misc` `list`. Miscellaneous additional objects. Default `list()`.

## Details

### Projected Change in Catchability:

In projection years, simulated index values are `Efficiency * NomIndex_t^Beta * Error_t * (1 + qInc / 100)^(t - t_H)`, where `t - t_H` is the time in years since the last historical time step (a fraction of a year for each season of a seasonal model). The multiplier is 1 in the historical period, so `qInc` has no effect on historical data, conditioning on real data, or `Ref`. Index values supplied for projection years in `OM@Data` are used as-is.

### Observation Error Generation:

The `Error` array (`nSim x nYear`) is obtained in one of three ways, in order of precedence:

1. **User-supplied** `Error`: stored directly; `CV` is not used.

2. **Stochastic from `CV`**: if `Error` is `NULL` and `CV` is specified, Internally the model generates a lognormal array where each cell is `rlnorm(1, mconv(1, CV[sim]), sdconv(1, CV[sim]))`. `CV` is first expanded to length `nSim` internally — a length-2 input triggers a stochastic draw of a unique CV per simulation.
3. **Conditioned from real data**: `Simulate()` estimates `CV`, `Beta`, `Efficiency`, `AC`, `Stats`, and `Error` from historical residuals; any user-supplied values are replaced.

`Ref` is expanded internally: a scalar is treated as the CV of a lognormal from which one reference value per simulation is drawn; a length-`nSim` input is used directly.

### Conditioning on Real Data:

When real index data are provided (via `OM@Data`), `Simulate()` conditions the observation model and the following slots are estimated:

- `Efficiency` (`q`) is always fit per simulation by log-linear regression of `log(Observed)` on `log(NomIndex)`, under `Observed_t = Efficiency * NomIndex_t^Beta`. `Beta` is jointly estimated the same way, but only when `SimControl(EstimateBeta = TRUE)` and the user has not already supplied a value for that index; otherwise `Beta` is held fixed (at the user's value, or `1`). When estimated, each simulation gets an explicit fit `Status` (`"estimated"`, `"fixed_low_variance"`, `"fixed_not_significant"`, `"fixed_bounds"`, ...) plus `SE_Beta`, a confidence interval, `R2`, and `PValue`, stored in `Misc$BetaFit` (see `EstimateBeta()`) and summarized by `IndexFitTable()`.
- `Stats`: a data frame computed by `CalcResidualStats()` with columns `Sim`, `AC` (weighted lag-1 autocorrelation of log-residuals), `SD` (standard deviation of log-residuals), and `NA_Season` (a list-column indicating which seasons, if any, have no observations in any year).
- `AC` and `CV` are set from `Stats$AC` and `Stats$SD` (the latter converted from log-scale SD to CV via the inverse of `sdconv()`), so they reflect the estimated values used to generate `Error` rather than whatever was supplied beforehand.
- `Error`: lognormal error multipliers covering both historical and projection years, with projection errors generated by `GenResiduals()` using the estimated `SD` and `AC`.

The `Beta`, `Efficiency`, `AC`, `CV`, `Stats`, and `Error` slots are replaced during conditioning regardless of whether the user supplied `Error` directly.

### Selectivity Notes:

Index selectivity is set only on the indicesobs object, and is used both when conditioning on real index data and when generating simulated index data. CPUE indices named after a fleet use that fleet's selectivity when `Selectivity` is `NULL`. Survey indices must set `Selectivity` with one of the forms described under the `Selectivity` parameter.

### Attaching to an Obs Object:

```
CPUE(obs)   <- IndicesObs(CV = 0.3)
Survey(obs) <- IndicesObs(CV = 0.2, Selectivity = "SBiomass")

# index of ages 3-5
Survey(obs) <- IndicesObs(Selectivity = c('3' = 1, '4' = 1, '5' = 1))
```

### Value

- `IndicesObs()` returns an indicesobs object.
- `CPUE()`, `Survey()` return the corresponding indicesobs slot from an obs object `x`.
- `CPUE<-`, `Survey<-` return `x` with the slot replaced.

### See Also

- indicesobs for the class definition and slot-level documentation.
- `Obs()` for the enclosing observation model constructor.
- `CalcResidualStats()` for residual statistic computation.
- `GenResiduals()`, `ApplyAC()` for projection error generation.
- `Simulate()` for conditioning the observation model on real data.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

### Examples

```
# Biomass CPUE with uniform 30% CV
io <- IndicesObs(CV = 0.3, Units = "Biomass")

# Stochastic CV - unique CV per simulation
io <- IndicesObs(CV = c(0.3, 0.05), Units = "Biomass")

# Spawning biomass survey using maturity as selectivity
survey_obs <- IndicesObs(CV = 0.2, Selectivity = "SBiomass", TruncSD = 3)

# Index covering only areas 1 and 2
io <- IndicesObs(CV = 0.25, Areas = c(1L, 2L))

# Attach to an obs object
obs <- Obs(CPUE   = IndicesObs(CV = 0.3),
           Survey = IndicesObs(CV = 0.2, Selectivity = "SBiomass"))

# Index catchability increasing 2% per year during the projection
io <- IndicesObs(CV = 0.3, qInc = 2)

# Creep drawn per simulation from a uniform distribution between 0 and 3%
io <- IndicesObs(CV = 0.3, qInc = c(0, 3))
qInc(io)

# Declining catchability
qInc(io) <- -1
```

indicesobs-class *The* indicesobs *S4 Class*

## Description

Defines the observation error structure for abundance or biomass indices (CPUE or survey). Used in the CPUE and Survey slots of an obs object. Objects are typically created via IndicesObs(), which documents all parameters in detail.

## Slots

CV numeric or NULL. Coefficient of variation of index observation error. See IndicesObs().

Error numeric array or NULL. Lognormal observation error multipliers (nSim x nYear). May be supplied directly via IndicesObs(), generated stochastically from CV, or estimated from real data during Simulate(). See IndicesObs().

Beta numeric array or NULL. Hyperstability/hyperdepletion parameter: Observed_t = Efficiency * NomIndex_t^Beta * Error_t. NULL (default) is Beta = 1 (proportional); otherwise estimated per simulation when conditioned on real data and SimControl(EstimateBeta = TRUE), unless already supplied. See IndicesObs().

AC numeric array or NULL. Lag-1 autocorrelation of index residuals, one value per simulation. When supplied by the user, overrides the value estimated during conditioning. See IndicesObs().

Years numeric or NULL. Calendar years over which the observation error is conditioned. See IndicesObs().

Areas numeric or NULL. Integer indices of the spatial areas contributing to this index. See IndicesObs().

Units character or NULL. Units of the index ("Biomass", "Number", or "Recruitment"). See IndicesObs().

Selectivity array, character, numeric, or list. Selectivity-at-age of this index. See IndicesObs().

Type character. Index type identifier. See IndicesObs().

Ref numeric array or NULL. Reference index value, one per simulation. See IndicesObs().

Efficiency numeric array or NULL. Catchability coefficient q relating the nominal index to the population quantity. Calculated internally during conditioning as the ratio of mean observed index to mean simulated index; not set by the user.

qInc numeric array or NULL. Annual percentage change in index catchability during projection years, one value per simulation. See IndicesObs().

TruncSD numeric(1) or NULL. Number of standard deviations at which to truncate the lognormal residual distribution when generating projection errors. Default 2. See IndicesObs().

Stats data.frame or NULL. Residual statistics computed during conditioning by CalcResidualStats(). Contains columns Sim, AC (weighted lag-1 autocorrelation), SD (standard deviation), and NA_Season (a list-column of seasons with no observations). Populated internally; not set by the user.

`Misc list`. Miscellaneous additional objects. When `Beta` is conditioned on real data, `Misc$BetaFit` holds the per-simulation fit diagnostics from `EstimateBeta()`: `SE_Beta`, `CI_Lower`, `CI_Upper`, `R2`, `PValue`, `nPoints`, and `Status`. See `IndexFitTable()`.

### See Also

- `IndicesObs()` for the constructor and full parameter documentation.
- obs for the enclosing observation model object.
- `CalcResidualStats()` for how `Stats` is computed.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `lifehistoryobs-class`, `obs-class`

---

## `initialize-methods` *~~ Methods for Function* `initialize` *~~*

---

### Description

~~ Methods for Function `initialize` ~~

### Methods

**list('signature(.Object = \'DLM\')')** %% ~~describe this method here~~

**list('signature(.Object = \'Fleet\')')** %% ~~describe this method here~~

**list('signature(.Object = \'MSE\')')** %% ~~describe this method here~~

**list('signature(.Object = \'Obs\')')** %% ~~describe this method here~~

**list('signature(.Object = \'OM\')')** %% ~~describe this method here~~

**list('signature(.Object = \'Stock\')')** %% ~~describe this method here~~

**list('signature(.Object = \'Fease\')')** %% ~~describe this method here~~

**list('signature(.Object = \'DLM_general\')')** %% ~~describe this method here~~

---

## `Input` *Function to run a set of input control methods*

---

### Description

Runs a set of input control methods are returns the output in a single table

### Usage

```
Input(Data, MPs = NA, reps = 100, timelimit = 10, CheckMPs = TRUE, msg = TRUE)
```

### Arguments

| | |
|---|---|
| `Data` | A Data object |
| `MPs` | A list of input MPs, if NA all available input MPs are run |
| `reps` | Number of repetitions (for those methods that use them) |
| `timelimit` | Maximum timelimit to run MP (in seconds) |
| `CheckMPs` | Logical, the Can function is run if this is TRUE |
| `msg` | Logical. Should messages be printed? |

### Author(s)

A. Hordyk

### Examples

```
## Not run:
library(MSEtool)
Input(MSEtool::Cobia)

## End(Not run)
```

---

## `InterimAdvice` *Interim Advice for OM Objects*

---

### Description

Access or replace the `InterimAdvice` slot of an om object: analyst-supplied fixed or stochastic TAC/Effort values used for "interim" projection years, i.e. years before `MPStartYear` in which the MP is not called (see `MPStartYear()`). Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
InterimAdvice(x)

InterimAdvice(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a `data.frame` as described in *Format* below, or `NULL`. |

## Details

When the historical period ends before MPs should actually start being applied (e.g. the OM is conditioned on data to 2024 but MPs will not be implemented until 2028), set `MPStartYear` to the first calendar year MPs should run. Projection years before `MPStartYear` are "interim" years: the MP is not called, `Imp` (implementation error) is bypassed since these values represent actual/plausible realised catch or effort rather than a management recommendation, and advice is instead taken from `InterimAdvice`.

### Format:

`InterimAdvice` is a `data.frame` with columns:

- `Year`: an interim timestep, i.e. a value of `Years(OM, "Projection")` before `MPStartYear`. See *Annual and seasonal values* below.
- `Complex`: stock complex name, matching `names(OM@Complexes)`. Optional when the OM has a single complex, in which case every row applies to it; required when there is more than one.
- `Fleet`: optional; fleet name, or `NA` for a complex total. A `Fleet = NA` row gives the total for the fleets not listed individually in the same Year x Complex x Type, split among them by `FleetAllocation()` (TAC) or `EffortAllocation()` (absolute Effort).
- `Type`: `"TAC"` or `"Effort"`.
- `Mean`: point estimate, natural scale. Must be `>= 0` (e.g. `0` for a fleet with no historical catch, maintained as a closure in interim years).
- `CV`: optional; coefficient of variation of a lognormal multiplier with mean 1. `NA`/`0` (default) gives a fixed, deterministic value. May differ among rows, e.g. `0` for fleets with reported catch and `> 0` for fleets with assumed catch. One standard-normal deviate is drawn per simulation for each calendar year x Complex x Type x Fleet and shared by that fleet's seasonal rows; each row converts it to a multiplier with its own `CV`. Draws are independent among fleets. A `Fleet = NA` total is one draw, split among its fleets as described above.
- `Max`: optional; upper bound on the drawn value. A fleet's deviate is truncated so that none of its rows in the calendar year exceeds its `Max`. `Mean` is the mean before truncation; the realised mean is lower. `NA` (default) gives no bound. If supplied, must be `>= Mean`.
- `TACType`: `"Removals"` (default) or `"Landings"`; only used for `"TAC"` rows.
- `TACUnit`: units of `Mean` and `Max` for `"TAC"` rows, as in `Advice()`: `"Biomass"` (default), `"Number"`, or a catch unit such as `Data@Landings@Units` (e.g. `"t"`, `"kg"`, `"n"`), which is stored as `"Biomass"` or `"Number"`. The TAC is not rescaled, so a TAC in biomass is in the OM's `Weight` units. May differ among fleets.
- `EffType`: `"Abs"` (default) or `"Rel"`; only used for `"Effort"` rows. `"Abs"` is absolute effort. `"Rel"` is a multiplier on the fleet's effort in the same season of the last historical year.

### Annual and seasonal values:

For each calendar year x Complex x Type x Fleet (with `Fleet = NA` treated as its own entry), supply either:

- one row with the integer calendar year: an annual value. For seasonal OMs (`Seasons > 1`), an annual TAC or absolute Effort is split across seasons by `SeasonalAllocation()`; an annual relative Effort applies to every season; or
- one row for every timestep of that year (the integer year is the first season): a value for each season, used as given.

Different fleets in the same year may use different forms. Every complex must have rows in every interim year, and within each calendar year x Complex x Type every fleet must be covered, either by its own rows or by a `Fleet = NA` row. `InterimAdvice` must be supplied if and only if `MPStartYear` leaves at least one interim year. These rules are checked at the start of `Project()`, and any violation is an error.

### Value

- `InterimAdvice()` returns the value of the `InterimAdvice` slot.
- `InterimAdvice<-()` returns `x` with the `InterimAdvice` slot updated.

### See Also

- `OM()` for the constructor, om for the class definition.
- `MPStartYear()` for the slot controlling when interim years end.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM(CurrentYear = 2023, MPStartYear = 2028)
InterimAdvice(om) <- data.frame(
  Year = 2024:2027, Type = "TAC", Mean = 1000, CV = 0.2, Max = 1200
)
InterimAdvice(om)
```

---

## `Interval` *Management Interval for OM Objects*

---

### Description

Access or replace the `Interval` slot of an om object: how often, in years, an MP is called to produce new advice during projection. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
Interval(x)

Interval(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a positive numeric scalar, or a named numeric vector giving a per-MP interval. |

### Details

Use `ManagementScheduleTable()` to see which years are management years for a given OM.

`Interval` is always in years, regardless of `Seasons`: `Interval = 1` calls the MP once a year, `Interval = 2` every second year, and so on. The first management year is the first projection year, or `MPStartYear` if it is set. In projection years that are not management years, the TAC or Effort from the most recent management year carries forward unchanged.

**Per-MP intervals:**

A named vector sets the interval by MP name, e.g. `c(MP1 = 1, MP2 = 4)`. An unnamed element (e.g. `c(2, MP2 = 4)`) is the default for MPs not named. For each MP, the interval is resolved in this order:

1. The element of `Interval` named for the MP.
2. `1 / Seasons` if the MP has `attr(mp, 'EverySeason') <- TRUE`.
3. The MP's `attr(mp, 'Interval')`, if set.
4. The unnamed element of `Interval`, or its first element.

**Seasonal OMs:**

For a seasonal OM (`Seasons > 1`), `Interval * Seasons` must be a whole number of timesteps. MPs that must be called at every timestep (e.g. because they return a season-specific value) should declare `attr(mp, 'EverySeason') <- TRUE`, since `1 / Seasons` depends on the OM the MP is run with. Advice set for a multi-season interval is split across seasons by `SeasonalAllocation()`.

### Value

- `Interval()` returns the value of the `Interval` slot.
- `Interval<-()` returns `x` with the `Interval` slot updated.

### See Also

- `OM()` for the constructor, om for the class definition.
- `MPStartYear()`, `DataLag()` for the other slots controlling the management timeline.
- `ManagementScheduleTable()` for previewing the resulting schedule.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM(Interval = 2)
Interval(om)
Interval(om) <- c(3, MP2 = 1)

# Preview which years are management years
om <- OM(CurrentYear = 2026, Interval = 2)
ManagementScheduleTable(om)
```

---

`iSCAM` *Reads iSCAM files into a hierarchical R list object*

---

### Description

Internal functions for reading iSCAM input and output files into R

### Usage

```
load.iscam.files(model.dir, burnin = 1000, thin = 1, verbose = FALSE)

fetch.file.names(path, filename)

read.report.file(fn)

read.data.file(file = NULL, verbose = FALSE)

read.control.file(
  file = NULL,
  num.gears = NULL,
  num.age.gears = NULL,
  verbose = FALSE
)

read.projection.file(file = NULL, verbose = FALSE)

read.par.file(file = NULL, verbose = FALSE)

read.mcmc(model.dir = NULL, verbose = TRUE)
```

### Arguments

| | |
|---|---|
| `model.dir` | Folder name |
| `burnin` | The initial mcmc samples to be discarded |
| `thin` | The degree of chain thinning 1 in every thin iterations is kept |
| `verbose` | should detailed results be printed to console |

| | |
|---|---|
| `path` | File path |
| `filename` | The filename |
| `fn` | File location |
| `file` | File location |
| `num.gears` | The number of gears |
| `num.age.gears` | The number age-gears |

### Functions

- `load.iscam.files()`: Wrapper function to generate R list
- `fetch.file.names()`: A function for returning the three types of iSCAM input and output files
- `read.report.file()`: A function for returning the results of the .rep iscam file
- `read.data.file()`: A function for returning the results of the .dat iscam file
- `read.control.file()`: A function for returning the results of the iscam control file
- `read.projection.file()`: A function for returning the results of the iscam projection file
- `read.par.file()`: A function for returning the results of the iscam .par file
- `read.mcmc()`: A function for returning the results of the iscam mcmc files

### Author(s)

Chris Grandin (DFO PBS)

### See Also

iSCAM2OM

---

`iSCAM2OM` *Reads MPD or MCMC estimates and data from iSCAM file structure into an operating model*

---

### Description

Functions for importing an iSCAM assessment. From a fitted model, `iSCAM2OM` populates the various slots of an operating model and `iSCAM2Data` generates a Data object. These functions rely on several functions written by Chris Grandin (DFO PBS).

## Usage

```
iSCAM2OM(
  iSCAMdir,
  nsim = 48,
  proyears = 50,
  mcmc = FALSE,
  spawn_time_frac = 0,
  Name = "iSCAM model",
  Source = "No source provided",
  length_timestep = 1,
  nyr_par_mu = 2,
  Author = "No author provided",
  report = FALSE,
  silent = FALSE
)

iSCAM2Data(
  iSCAMdir,
  Name = "iSCAM assessment",
  Source = "No source provided",
  length_timestep = 1,
  Author = "No author provided"
)
```

## Arguments

| | |
|---|---|
| `iSCAMdir` | A folder with iSCAM input and output files in it. Alternatively, a list returned by load.iscam.files. |
| `nsim` | The number of simulations to take for parameters with uncertainty (for OM@cpars custom parameters) |
| `proyears` | The number of MSE projection years |
| `mcmc` | Logical, whether to use mcmc samples to create custom parameters cpars. Alternatively, a list returned by read.mcmc. Set the seed for the function to sub-sample the mcmc samples. |
| `spawn_time_frac` | Numeric between 0-1 indicating when spawning occurs within the time step |
| `Name` | The name of the operating model |
| `Source` | Reference to assessment documentation e.g. a url |
| `length_timestep` | How long is a model time step in years (e.g. a quarterly model is 0.25, a monthly model 1/12) (currently only uses annual time step) |
| `nyr_par_mu` | integer, the number of recent years to estimate vulnerability over for future projections |
| `Author` | Who did the assessment |
| `report` | logical should a numbers at age reconstruction plot be produced? |
| `silent` | logical should progress reporting be printed to the console? |

### Biological parameters

The function calls `model <- load.iscam.files(iSCAMdir)` and grabs the following matrices:

- `model$mpd$d3_weight_mat` - fecundity (product of weight and maturity at age)
- `model$mpd$ma` - maturity at age

### MPD historical reconstruction

The function calls `model <- load.iscam.files(iSCAMdir)` and then grabs the following matrices:

- `model$mpd$N` - abundance at age
- `model$mpd$F` - fishing mortality at age
- `model$mpd$M` - natural mortality at age

If a delay-difference model is recognized, then the following is used instead:

- `model$mpd$F_dd` - fishing mortality at age
- `model$mpd$M_dd` - natural mortality at age

Abundance at age is reconstructed using `model$mpd$rt` (recruitment) and projected with `F_dd` and `M_dd` to match `model$mpd$numbers`.

### MCMC historical reconstruction

If `mcmc = TRUE` the function calls `mcmc_model <- read.mcmc(iSCAMdir)`, and grabs `nsim` sub-samples of the MCMC output through the following arrays:

- `mcmc_model$params` and `mcmc_model$ft` - fishing mortality at age from the fleet selectivity parameters and apical F's
- `mcmc_model$m` - year-specific natural mortality at age
- `mcmc_model$params$rinit` and `mcmc_model$rt` - recruitment and abundance

### Start age

While the iSCAM start age can be greater than zero, abundance at age is back-calculated to age zero with M, maturity, growth = 0. In this way, the stock-recruit dynamics from iSCAM are preserved.

These arrays are then passed to Assess2OM to generate the operating model.

### Reference points

iSCAM calculates the stock-recruit relationship and subsequently a single set of MSY and unfished reference points using R0, steepness, and unfished spawners per recruit from the mean M, fecundity, and growth (mean with respect to time).

R0 and h are recalculated for the operating model by obtaining the stock-recruit alpha and beta from the iSCAM parameters and the mean unfished spawners per recruit in the first `ageM` (age of 50% maturity) years.

### Author(s)

T. Carruthers, Q. Huynh

---

`iSCAMcomps` *Combines all iSCAM age composition data across fleets*

---

### Description

iSCAM assessments are often fitted to numerous fleets that have differing age selectivities. iSCAMcomps is a simple way of providing the aggregate catch at age data. It should be noted that this process is important and in a real application would require due diligence (ie peer reviewed data workshop).

### Usage

```
iSCAMcomps(replist, Year)
```

### Arguments

| | |
|---|---|
| `replist` | S3 class object: the output from a read from an iSCAM data folder |
| `Year` | Integer vector: the years of the data object ie Data@Year |

### Author(s)

T. Carruthers

---

`iSCAMinds` *Combines indices into a single index using linear modelling (** Deprecated **)*

---

### Description

iSCAM assessments often make use of multiple indices of abundance. The data object and MPs currently only make use of a single index. combiSCAMinds is a function that creates a single index from many using linear modelling. It is a simple way of providing initial calculations of management recommendations and it should be noted that this process is important and in a real application would require due diligence (ie peer reviewed data workshop).

### Usage

```
iSCAMinds(idata, Year, fleeteffect = T)
```

### Arguments

| | |
|---|---|
| `idata` | List: the indices recorded in a read from an iSCAM data folder, e.g. replist$data$indices |
| `Year` | Integer vector: the years of the data object ie Data@Year |
| `fleeteffect` | Logical: should a fleet effect be added to the linear model? |

### Author(s)

T. Carruthers

---

## `isNewObject` *Test Whether an Object is a Default-Constructed Instance*

---

### Description

Returns `TRUE` if `object` is identical to a freshly constructed default instance of its class — i.e. the object has not been modified since construction. Used internally by `EmptyObject()`.

### Usage

```
isNewObject(object)
```

### Arguments

| | |
|---|---|
| `object` | An S4 object. |

### Details

' `isNewObject()` constructs a default instance of the same class as `object` and compares the two with `identical()`. Before comparing, the `Name` slot (if present) is set to `NULL` on `object` so that a user- supplied name alone does not cause the object to be considered non-default.

### Value

A length-1 logical. Returns `FALSE` for non-S4 objects.

### See Also

`EmptyObject()`

## join_plots *Plot several plots with a shared legend*

### Description

Plot several plots with a shared legend

### Usage

```
join_plots(
  plots,
  ncol = length(plots),
  nrow = 1,
  position = c("right", "bottom"),
  legend = TRUE
)
```

### Arguments

| | |
|---|---|
| plots | list of plot objects of class gg or ggplot |
| ncol | Optional number of columns |
| nrow | Optional number of rows |
| position | position of the legend ("bottom" or "right") |
| legend | Logical. Use a legend? |

### Note

modified from https://github.com/tidyverse/ggplot2/wiki/share-a-legend-between-two-ggplot2-graphs

## joinData *Join Data objects present in a list*

### Description

A function that combined a list of data objects into a single data object (same dimensions but can have different numbers of simulations)

### Usage

```
joinData(DataList)
```

### Arguments

| | |
|---|---|
| DataList | A list of data objects of identical dimension (except for simulation) |

### Author(s)

T. Carruthers

### See Also

joinMSE joinHist

---

`JoinYear` *Join Two Objects Along the Year Dimension*

---

### Description

Recursively combines two objects of the same structure by appending `object2` after `object1` along the year dimension. Handles S4 objects, lists, named arrays, and numeric vectors. Intended for joining early and late year segments of MSEtool data objects (e.g. historical and projection periods).

### Usage

```
JoinYear(object1, object2)
```

### Arguments

| | |
|---|---|
| `object1` | The earlier object. Must have years that precede those in `object2`. Can be an S4 object, a `list`, a named `array` with a `"Year"` dimension, or a `numeric` vector. |
| `object2` | The later object with the same structure as `object1`. Its years must be strictly greater than those in `object1`. |

### Details

`JoinYear` recurses through the structure of `object1` and `object2` in parallel, dispatching on the type of each element:

- **S4 objects** — each slot is processed recursively via `Recall()`. If a slot is `NULL` on `object1` but populated on `object2` (e.g. `Advice` during the historical period, before any MP has run), the joined result is `object2`'s value rather than `NULL`.
- **Lists** — each element is processed recursively via `Recall()`.
- **Named arrays with a `"Year"` dimension** — joined along the year axis using `abind::abind()`. An error is thrown if years are not strictly increasing across `object1` and `object2`.
- **Numeric vectors** — if `object1` and `object2` are identical (e.g. static per-fleet metadata such as a `compdata`'s `Classes` bin boundaries, unchanged between the historical and projection segments), `object1` is returned as-is rather than duplicated. Otherwise the two are concatenated with `c()`. If all values exceed 1000 (i.e. are likely calendar years), strict monotonicity is checked and an error is thrown if the combined sequence is not increasing.
- **All other types** — returned unchanged from `object1`.

### Value

An object of the same class and structure as `object1`, with `object2` appended along the year dimension. Non-array, non-numeric, and non-list slots or elements are returned unchanged from `object1`.

### Error handling

If years are not strictly increasing across the two objects, `cli::cli_abort()` is called with a diagnostic message showing the year values from both objects and a hint that the arguments may be in the wrong order.

### See Also

`abind::abind()`

---

## Kplot — *KOBE plot: a projection by projection plot of F/FMSY and B/BMSY*

---

### Description

A standard KOBE plot by each method that also shows the percentage of methods that ended up in each quadrant.

### Usage

```
Kplot(
  MSEobj,
  maxsim = 60,
  MPs = NA,
  sims = NULL,
  maxMP = 9,
  nam = NA,
  cex.leg = 1.5
)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `maxsim` | Maximum number of simulations (lines) to plot on each panel. |
| `MPs` | Optional subset MSE object by MP |
| `sims` | Optional subset MSE object by simulation |
| `maxMP` | Maximum number of MPs to include in plot |
| `nam` | The name of the plot |
| `cex.leg` | Size of legend |

### Note

Apologies for the nauseating shading.

### Author(s)

T. Carruthers with some additions from A. Hordyk

---

`Lag_Data` *Lag the time-series slots in a* `Data` *object by a specified number of time-steps*

---

### Description

Lag the time-series slots in a `Data` object by a specified number of time-steps

### Usage

```
Lag_Data(Data, Data_Lag = 0, msg = FALSE)
```

### Arguments

| | |
|---|---|
| `Data` | An object of class `Data` |
| `Data_Lag` | Either a numeric vector of length 1 with a positive number specifying the number of time-steps to lag all time-series data, or a named list with numeric values (length 1). See details for more information. |
| `msg` | Logical. Display the messages? |

### Details

By default, all simulated data in the forward projections are provided up to the previous time-step. That is, in projection year `t`, the simulated data are up to and including `t-1`. This function will lag the time-series values by the specified value. For example, `Data_Lag=5` will mean in projection time-step `t` the data will be up to and including `t-6`.

*Note*: The `Data@Year` slot is *not* lagged by this function. Many built-in MPs use the length of `Data@Year` to determine the number of years of data for smoothing over recent years etc. This may not be appropriate so check the MP is behaving as you expect if you use `Lag_Data`.

### Value

An object of class `Data` with time-series slots lagged.

### Examples

```
# Lag all time-series slots by 2 time-steps (usually years)
Data <- Example_datafile
Lagged_1 <- Lag_Data(Data, 2)
length(Data@Year)
length(Lagged_1@Year)
length(Data@Cat[1,])
length(Lagged_1@Cat[1,])
length(Data@Ind[1,])
length(Lagged_1@Ind[1,])

# Lag CAA by 5 and Ind by 3 time-steps
Lagged_2 <- Lag_Data(Data, Data_Lag=list(CAA=5, Ind=3))
length(Lagged_2@Year)
length(Lagged_2@Cat[1,])
dim(Data@CAA[1,,])
dim(Lagged_2@CAA[1,,])

length(Data@Ind[1,])
length(Lagged_2@Ind[1,])
```

---

## ldim *Dimensions of a hierarchical list object*

---

### Description

Dimensions of a hierarchical list object

### Usage

```
ldim(x)
```

### Arguments

| | |
|---|---|
| x | A list |

### Author(s)

T. Carruthers

`Length` *Length Constructor and Accessors*

## Description

Construct a length object defining the length-at-age schedule for a stock, or access and replace the `Length` slot of a stock and its individual slots. A `Length` object is required for all stock objects.

## Usage

```
Length(
  Pars = list(),
  Model = NULL,
  Units = "mm",
  MeanAtAge = NULL,
  CVatAge = 0.1,
  Dist = "normal",
  TruncSD = 2,
  Timing = 0,
  Random = NULL,
  ALK = NULL,
  Classes = NULL,
  Misc = list()
)

Length(x) <- value

ALK(x)

ALK(x) <- value
```

## Arguments

| | |
|---|---|
| `Pars` | `list`. Named list of growth parameters. Element names must match the arguments of a built-in growth model (see `LengthModels()`). When `Pars` is non-empty and `Model` is `NULL`, the model is inferred automatically from the parameter names. When `Pars` is a stock object, `Length()` acts as a pass-through accessor and returns `x@Length`. See *Specifying the Length-at-Age* and Specifying Biological and Fleet Schedules for the full set of accepted input formats. Default `list()`. |
| `Model` | `character(1)` or `function`. Growth model identifier. When `NULL` (default), the model is inferred from `Pars`. May be set to a character string naming a built-in model or to a custom R function — see Specifying Biological and Fleet Schedules. |
| `Units` | `character(1)`. Physical unit of length measurements. Must be one of the strings returned by `ValidUnits()` (e.g., `"mm"`, `"cm"`). Default `"mm"`. |

`MeanAtAge` — `array` or `NULL`. Mean length at age with named dimensions `Sim × Age × Year`. Supply directly when bypassing the model-based approach (`Pars = list()`). Only the years at which values *change* need to be included; `Extend()` fills the remainder. When `Pars` contains a matched model, any values here are **overwritten** during `Populate()`. A numeric vector of length `nAge` is also accepted and is converted to a `1 × nAge × 1` array automatically. Default `NULL`.

`CVatAge` — `numeric(1)`, length-2 bounds vector, or `array`. Coefficient of variation of length-at-age within each age class, used to parameterise the within-age-class length distribution and to build the age-length key (`ALK`). A length-2 vector is treated as `Uniform(lower, upper)` bounds sampled once per simulation. Default `0.1`.

`Dist` — `character(1)`. Parametric distribution for length-at-age variability. Options are: `"normal"` (default) and `"lognormal"`).

`TruncSD` — `numeric(1)`. Number of standard deviations at which the length distribution is truncated, preventing unrealistic tails in the `ALK`. Default `2`.

`Timing` — `numeric(1)`. Fractional position within the time step at which length is measured: `0` = start of step, `1` = end of step. Shifts age class midpoints before computing the `ALK`. Relevant for seasonal models where growth occurs within a step. Default `0`.

`Random` — `array` or `NULL`. Simulation- and year-specific random multipliers applied to growth parameters, enabling inter-annual or among-simulation variation in growth. Currently stored but not applied. Default `NULL`.

`ALK` — `array` or `NULL`. Age-length key giving the probability that an individual of a given age belongs to each length class (`Sim × Age × Class × Year`). Populated automatically during `Populate()` from `MeanAtAge`, `CVatAge`, `Dist`, and `TruncSD`; supply directly only when overriding the computed key. Default `NULL`.

`Classes` — `numeric` or `NULL`. Length class midpoints in units of `Units`. Populated automatically during `Populate()` if `NULL`. Default `NULL`.

`Misc` — `list`. Used internally. Default `list()`.

`x` — A length object for slot accessors, or a stock object for `Length<-`.

`value` — For `Length<-`: a length object. For `ALK<-`: a replacement age-length key array.

## Details

### Specifying Length-at-Age:

There are two ways to define the length schedule:

1. **Model-based** (recommended): supply `Pars` with the named parameters for your chosen growth model (e.g., `list(Linf = 80, K = 0.2, t0 = -0.5)` for von Bertalanffy). If `Model` is `NULL` and the parameter names uniquely identify a model, `Model` is set automatically. The `MeanAtAge` array and `ALK` are then populated by `Populate()` when the stock is added to an `OM()`.
2. **Direct array**: supply `MeanAtAge` with `Pars = list()`. Any values already present in `MeanAtAge` are preserved and are not overwritten during `Populate()`.

See Specifying Biological and Fleet Schedules for more details.

**Length Distribution and the ALK:**

Within each age class, lengths follow the distribution given by `Dist`, centred on `MeanAtAge` with spread `CVatAge`, truncated at `TruncSD` standard deviations. These parameters are used to build the age-length key (`ALK`), which maps age-structured abundance to length-structured quantities throughout the operating model. Retrieve the computed key with `ALK(x)`.

**Pass-Through Access from a Stock:**

When `Pars` is a stock object, `Length()` returns the `Length` slot directly:

```
Length(my_stock)           # returns my_stock@Length
Length(my_stock) <- my_L   # replaces my_stock@Length
```

**Slot Accessors:**

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Pars(len)         <- list(Linf = 80, K = 0.2, t0 = -0.5)
Model(len)        <- "vonBert"
Units(len)        <- "cm"
MeanAtAge(len)    <- my_array
CVatAge(len)      <- 0.08
Dist(len)         <- "normal"
TruncSD(len)      <- 3
Timing(len)       <- 0
ALK(len)          <- my_alk
Classes(len)      <- seq(5, 80, by = 5)
```

## Value

- `Length()` returns a length object. If `Pars` is a stock, returns `x@Length`.
- `Length<-` returns the stock `x` with the `Length` slot replaced by `value` and the object re-validated.
- `ALK()` returns the age-length key array from `x`.
- `ALK<-` returns `x` with the `ALK` slot updated and the object re-validated.

## See Also

- length for the class definition and slot-level documentation.
- `LengthModels()` for available growth models and required parameter sets.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.

Other length: `length-class`

### Examples

```
# See LengthModels() for all available built-in growth models and the
# parameter names required for each.
LengthModels()

# ---- Model-based specification ----

## Constant parameters (scalar) - same value for every simulation and year
len <- Length(Pars = list(Linf = 100, K = 0.2, t0 = -0.1))

## Stochastic across simulations - Linf and K drawn from Uniform(lower, upper)
## once per simulation; t0 fixed at -0.1 in all simulations
len <- Length(Pars = list(Linf = c(90, 110),
                          K    = c(0.15, 0.25),
                          t0   = -0.1))

## Inter-annual random walk on Linf - log-normal with 5% CV
## LinfSD is removed from Pars after use
len <- Length(Pars = list(Linf    = c(90, 110),
                          LinfSD  = 0.05,
                          K       = c(0.15, 0.25),
                          t0      = -0.1))

## Time-varying parameters with named change-point arrays
## Only the years where values change need to be supplied; Extend() fills
## all intermediate and future years by forward-filling automatically.
nSim <- 48
## Linf increases in 2010 (two change points: 1990 and 2010)
Linf_arr <- array(
  c(rep(80, nSim), rep(90, nSim)),
  dim      = c(nSim, 2),
  dimnames = list(Sim = seq_len(nSim), Year = c(1990, 2010))
)
len <- Length(Pars = list(Linf = Linf_arr,
                          K    = c(0.15, 0.25),
                          t0   = -0.1))

# ---- Direct array specification ----

## Supply MeanAtAge directly (Pars left empty).
## A plain numeric vector of length nAge is accepted; it is treated as a
## single simulation, single year and replicated by Extend() as needed.
ages   <- 0:20
len_aa <- Length(MeanAtAge = 100 * (1 - exp(-0.2 * (ages + 0.5))))

## Named Sim × Age × Year array - two change-point years, single simulation.
## Extend() replicates Sim = 1 to all nSim simulations automatically.
maa <- array(
  c(100 * (1 - exp(-0.2 * (ages + 0.5))),   # 1990 schedule
    110 * (1 - exp(-0.2 * (ages + 0.5)))),   # 2010 schedule (Linf increased)
  dim      = c(1, length(ages), 2),
  dimnames = list(Sim = 1, Age = ages, Year = c(1990, 2010))
```

```
)
len_arr <- Length(MeanAtAge = maa)

# ---- Custom length class midpoints ----

## By default, Classes and the ALK are populated automatically during
## Populate().
# Supply Classes explicitly to override the default bin width.
len_cls <- Length(
  Pars    = list(Linf = 100, K = 0.2, t0 = -0.1),
  Classes = seq(from = 2.5, by = 5, length.out = 26)
)

# ---- Slot accessors ----

len <- Length(Pars = list(Linf = 100, K = 0.2, t0 = -0.1))

## Read slots
MeanAtAge(len)            # NULL until Populate() is called
CVatAge(len)
Units(len)

## Replace slots
Units(len)   <- "cm"
CVatAge(len) <- 0.08
TruncSD(len) <- 3

# ---- Attaching to a Stock ----

stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
Length(stk) <- Length(Pars = list(Linf = 100, K = 0.2, t0 = -0.1))
Length(stk)

# ---- Populate ----

## The final populated object (typically done internally)
pop_len <- Populate(Length(stk))
pop_len

MeanAtAge(pop_len)
```

---

`Length-at-Age-Models` *Length-at-Age Growth Models*

---

## Description

Length-at-age (LAA) growth models

### Usage

```
vonBert(Ages, Linf, K, t0 = 0)

Brody(Ages, L0, Linf, K)

Gompertz(Ages, Linf, g, a)

Richards(Ages, Linf, K, t0 = 0, d = 1)

LengthModels(full = TRUE, print = TRUE)
```

### Arguments

| | |
|---|---|
| `Ages` | Numeric vector of ages (always in years). |
| `Linf` | asymptotic length (used by von Bertalanffy, Gompertz, Brody) |
| `K` | growth coefficient (used by von Bertalanffy, Brody) |
| `t0` | theoretical age at zero length (von Bertalanffy) |
| `L0` | length at age 0 (Brody) |
| `g` | growth rate parameter (Gompertz) |
| `a` | age at inflection (Gompertz) |
| `d` | Richards shape parameter. `d = 1` recovers von Bertalanffy; `d < 1` gives faster early growth; `d > 1` gives slower early growth. |
| `full` | Logical. Provide a complete table (TRUE) or just the model names (FALSE)? |
| `print` | Logical. Print out the results (TRUE) or just return the data.frame (FALSE)? |

### Details

`LengthModels()` prints the list of models and parameters to the console

- **von Bertalanffy**:
$$L(a) = L_\infty(1 - e^{-K(a-t_0)})$$
- **Brody**:
$$L(a) = L_\infty - (L_\infty - L_0)e^{-Ka}$$
- **Gompertz**:
$$L(a) = L_\infty \exp(-\exp(-g(a-a)))$$
- **Richards**:
$$L(a) = L_\infty \left(1 - e^{-K(a-t_0)}\right)^{1/d}$$

### Value

- `LengthModels()` invisibly returns a data frame containing the models
- Growth models: Numeric vector of expected lengths at age.

### See Also

Ages(), Length(), Stock()

### Examples

```
Ages <- 0:15

L_vb <- vonBert(Ages, Linf = 100, K = 0.25, t0 = -0.5)
L_brody <- Brody(Ages, L0 = 10, Linf = 100, K = 0.3)
L_gompertz <- Gompertz(Ages, Linf = 100, g = 0.35, a = 3)
L_schnute <- Schnute(Ages, y0 = 12, y1 = 100, t0 = 0, t1 = 15, a = 0.5, b = 1.2)

df <- data.frame(
  Age = Ages,
  vonBert = L_vb,
  Brody = L_brody,
  Gompertz = L_gompertz,
  Schnute = L_schnute
)

matplot(df$Age, df[,-1], type = "l", lty = 1, lwd = 2,
        col = c("blue", "red", "green", "purple"),
        xlab = "Age (years)", ylab = "Length",
        main = "Length-at-Age Models")
legend("bottomright", legend = colnames(df)[-1],
       col = c("blue", "red", "green", "purple"), lty = 1, lwd = 2)
```

---

## length-class *The* length *S4 Class*

---

### Description

Defines the length-at-age schedule for a stock object. Objects are typically created via Length(), which documents all parameters, validates inputs, and infers the growth model automatically when possible.

### Details

Direct construction via methods::new() is not recommended; use Length() instead, which handles model inference and object validation.

The ALK slot is derived from MeanAtAge, CVatAge, Dist, and TruncSD during Populate() and should not be set manually.

## Slots

`Pars list`. Named list of growth parameters whose names correspond to those expected by `Model`. See `LengthModels()` for parameter sets and Specifying Biological and Fleet Schedules for accepted input formats.

`Model function` or `character(1)`. Growth model identifier, matched to one of `LengthModels()`. Set automatically from the parameter names when `Pars` is supplied without an explicit model.

`Units character(1)`. Physical unit of the length measurements (e.g., `"mm"`, `"cm"`). Must be accepted by `ValidUnits()`.

`MeanAtAge array`. Mean length at age with named dimensions `Sim`, `Age`, and `Year`. Populated automatically by `Populate()` when `Pars` and `Model` are set; may also be supplied directly when `Pars` is empty. When `Pars` contains a matched model, any existing values are overwritten. See Specifying Biological and Fleet Schedules.

`CVatAge array` or `numeric`. Coefficient of variation of length-at-age, used to generate the within-age-class length distribution.

`Dist character(1)`. Parametric distribution used for length-at-age variability (e.g., `"normal"`).

`TruncSD array` or `numeric`. Number of standard deviations at which the length distribution is truncated.

`Timing array` or `numeric`. Fractional position within the time step at which length is measured (`0` = start, `1` = end).

`Random array`. Random effects on growth parameters across simulations or years.

`ALK array`. Age-length key: the probability of belonging to each length class given age, derived from `MeanAtAge`, `CVatAge`, `Dist`, and `TruncSD`. Populated automatically; use `ALK()` to retrieve.

`Classes numeric`. Lower bounds of length bins in units of `Units`. Bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended.

`Misc list`. Used internally.

## See Also

- `Length()` for the constructor and accessor functions.
- `LengthModels()` for available growth models and their required #' parameters.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `ALK()` to retrieve the age-length key. Specifying Biological and Fleet Schedules for the full description of how `Pars`, `Model`, and `MeanAtAge` interact.

Other length: `Length()`

`LengthStructuredObs` *Length-Structured Observation Model*

## Description

An obs object representing a data-moderate scenario suitable for length-based stock assessment. Landings, discards, a fishery-independent survey, and annual length-composition samples from both landed and discarded fish are simulated.

## Usage

```
LengthStructuredObs
```

## Format

An obs object. Populated slots:

- `Landings` (catchobs): reported landings with CV in `[0.10, 0.20]` and slight negative-to-neutral bias `[0.90, 1.05]`.
- `Discards` (catchobs): less precisely reported discards with CV in `[0.15, 0.25]` and near-unbiased reporting `[0.95, 1.05]`.
- `Survey` (indicesobs): survey index assuming biomass-proportional selectivity, with CV in `[0.15, 0.30]` and autocorrelation in `[0.0, 0.2]`.
- `LandingsAtSize` (compobs): length-composition samples of landed fish from dockside measurements. Nominal sample size `[300, 600]` fish per year; ESS `[60, 120]` reflecting within-haul length clustering; `Theta` in `[0.5, 1.0]`.
- `DiscardsAtSize` (compobs): length-composition samples of discarded fish from at-sea observers. Nominal sample size `[150, 300]`; ESS `[40, 80]`; `Theta` in `[0.4, 0.8]`.

Empty slots (no data simulated): `Effort`, `CPUE`, `LandingsAtAge`, `DiscardsAtAge`.

## See Also

obs, `Obs()`, `CatchObs()`, `IndicesObs()`, `CompObs()`, CatchAndSurveyObs, CommercialFleetObs, AgeStructuredObs, DataRichObs

Other obs-examples: `AgeStructuredObs`, `CatchAndSurveyObs`, `CommercialFleetObs`, `DataRichObs`

## Examples

```
LengthStructuredObs
LandingsAtSize(LengthStructuredObs)
DiscardsAtSize(LengthStructuredObs)
```

LH2OM *Predict missing life-history parameters*

## Description

Predict missing life-history based on taxonomic information and hierarchical model fitted to FishBase life-history parameters

## Usage

```
LH2OM(
  OM,
  dist = c("unif", "norm"),
  filterMK = FALSE,
  plot = TRUE,
  Class = "predictive",
  Order = "predictive",
  Family = "predictive",
  msg = TRUE,
  db = MSEtool::LHdatabase
)

predictLH(
  inpars = list(),
  Genus = "predictive",
  Species = "predictive",
  nsamp = 100,
  db = MSEtool::LHdatabase,
  dist = c("unif", "norm"),
  filterMK = TRUE,
  plot = TRUE,
  Class = "predictive",
  Order = "predictive",
  Family = "predictive",
  msg = TRUE
)
```

## Arguments

| | |
|---|---|
| OM | An object of class 'OM' |
| dist | Character. Should parameters be sampled from a uniform (unif) or normal (norm) distribution? |
| filterMK | Logical or numeric specifying percentiles. See Details. e.g. OM@M and OM@K. Empty slots or slots with all values of 0 are considered unknown. |
| plot | Logical. Should the plot be produced? |
| Class | Optional higher order taxonomic information |

| | |
|---|---|
| `Order` | Optional higher order taxonomic information |
| `Family` | Optional higher order taxonomic information |
| `msg` | Logical. Should messages be printed? |
| `db` | Database from FishLife model with fitted model results |
| `inpars` | A named list with lower and upper bounds of provided parameters: *Linf*, *L50*, *K* and *M* (must be length 2). Unknown or missing parameters should not be included. For example, an empty list assumes that all four life history parameters are unknown and need to be estimated. See Details below for more information. |
| `Genus` | Character string specifying the Genus name. Optional. Default is 'predictive' |
| `Species` | Character string specifying the Species name. Optional. Default is 'predictive'. If full species name (Genus + Species) is not found if FishLife database (based on FishBase) higher order taxonomy will be used (e.g., Family) for the predictions. |
| `nsamp` | The number of samples to return |

## Details

**filterMK:**

If filterMK is logical: Should the predicted M and K parameters be filtered within the range specified in `inpars`or `OM`?

Otherwise, filterMK must be numeric vector of length(2) specifying lower and upper percentiles that will be applied to the predicted M or K values

The model predicts missing life-history parameters based on provided parameters and taxonomic information. If both *M* and *K* are provided in `inpars` or `OM`, *K* values are predicted and predictions filtered so that resulting *K* values are within bounds specified in `inpars$K` or `OM@K` (see `filterMK`).

If both *Linf* and *L50* are provided in `inpars` or `OM`, *L50* values are predicted and values in `inpars$L50` or `OM@L50` are ignored.

## Value

LH2OM: An OM with `OM@cpars` populated with `OM@nsim` samples of M, K, Linf and L50

predictLH: A data.frame with `nsamp` rows with `Linf`, `L50`, `K`, and `M` values.

## Functions

- `LH2OM()`: Predict missing life-history and populate `OM@cpars`
- `predictLH()`: Predict missing life-history based on taxonomic information and hierarchical model fitted to FishBase life-history parameters

## Author(s)

A. Hordyk

### Source

https://github.com/James-Thorson-NOAA/FishLife

### References

Thorson, J. T., S. B. Munch, J. M. Cope, and J. Gao. 2017. Predicting life history parameters for all fishes worldwide. Ecological Applications. 27(8): 2262–2276

---

## LHdatabase *LHdatabase*

---

### Description

Database from the FishLife package with predicted life-history parameters for all species on FishBase

### Usage

```
LHdatabase
```

### Format

An object of class `list` of length 3.

### Source

https://github.com/James-Thorson-NOAA/FishLife/

### References

Thorson, J. T., S. B. Munch, J. M. Cope, and J. Gao. 2017. Predicting life history parameters for all fishes worldwide. Ecological Applications. 27(8): 2262–2276

---

## lifehistorydata-class

`lifehistorydata` *Class*

---

### Description

Groups the biological life-history components required to describe population dynamics. Each slot holds a dedicated sub-object defining the model and parameters for that process. Used in the `LifeHistory` slot of a data object.

### Usage

```
LifeHistoryData()
```

### Value

`LifeHistoryData()` returns a `lifehistorydata` object.

### Slots

`Ages` An object of class ages defining the age structure of the population (minimum age, maximum age, plus-group).

`Length` An object of class length defining the growth model (e.g., von Bertalanffy) and length-at-age parameters.

`Weight` An object of class weight defining the weight-at-age or length–weight relationship.

`NaturalMortality` An object of class naturalmortality defining natural mortality rates, which may be age-, size-, or time-varying.

`Maturity` An object of class maturity defining maturity-at-age or maturity-at-length schedules.

`Fecundity` An object of class fecundity defining fecundity-at-age or fecundity-at-length relationships.

`SRR` An object of class srr defining the stock–recruitment relationship (e.g., Beverton–Holt, Ricker) and associated parameters.

`Spatial` An object of class spatial defining the spatial structure of the population, including movement and area allocation.

`Depletion` An object of class depletion specifying the initial depletion level relative to unfished biomass.

`Misc` A named list for any additional life-history metadata.

`LifeHistoryData()` creates a new `lifehistorydata` object.

### See Also

data, `Data()`

---

## `LifeHistoryObs` *LifeHistoryObs Constructor*

---

### Description

Construct a lifehistoryobs object defining observation error for life-history parameters.

### Usage

```
LifeHistoryObs(
  Ages = list(),
  Length = list(),
  Weight = list(),
  NaturalMortality = list(),
  Maturity = list(),
```

```
  Fecundity = list(),
  SRR = list(),
  Spatial = list(),
  Depletion = list(),
  Misc = list()
)
```

## Arguments

| | |
|---|---|
| `Ages` | `list`. Observation error specification for age structure parameters. Default `list()`. |
| `Length` | `list`. Observation error specification for growth parameters. Default `list()`. |
| `Weight` | `list`. Observation error specification for weight-at-age or length-weight parameters. Default `list()`. |
| `NaturalMortality` | `list`. Observation error specification for natural mortality. Default `list()`. |
| `Maturity` | `list`. Observation error specification for maturity schedules. Default `list()`. |
| `Fecundity` | `list`. Observation error specification for fecundity parameters. Default `list()`. |
| `SRR` | `list`. Observation error specification for stock-recruitment parameters. Default `list()`. |
| `Spatial` | `list`. Observation error specification for spatial parameters. Default `list()`. |
| `Depletion` | `list`. Observation error specification for initial depletion. Default `list()`. |
| `Misc` | `list`. Miscellaneous additional objects. Default `list()`. |

## Details

**Placeholder.** This class and constructor are reserved for future development and are not currently used. Each slot will eventually mirror the structure of the corresponding slot in lifehistorydata, providing bias and CV specifications for life-history parameter estimation. Slots currently accept untyped lists and all values are silently ignored during simulation.

When converted from a legacy Obs-legacy object via `ConvertObs()`, the life-history bias CVs (`Linfbiascv`, `Kbiascv`, `Mbiascv`, etc.) are not yet mapped to this class and are silently dropped.

## Value

A lifehistoryobs object.

### See Also

- lifehistoryobs for the class definition.
- `Obs()` for the enclosing observation model constructor.
- lifehistorydata for the complementary observed-values class.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

### Examples

```
lho <- LifeHistoryObs()
```

---

## `lifehistoryobs-class` *The* `lifehistoryobs` *S4 Class*

---

### Description

Defines the observation error structure for life-history parameters (growth, maturity, natural mortality, etc.). Used in the `LifeHistory` slot of an obs object. Objects are created via `LifeHistoryObs()`.

### Slots

`Ages list`. Observation error specification for age structure parameters. See `LifeHistoryObs()`.

`Length list`. Observation error specification for growth parameters. See `LifeHistoryObs()`.

`Weight list`. Observation error specification for weight-at-age or length-weight parameters. See `LifeHistoryObs()`.

`NaturalMortality list`. Observation error specification for natural mortality. See `LifeHistoryObs()`.

`Maturity list`. Observation error specification for maturity schedules. See `LifeHistoryObs()`.

`Fecundity list`. Observation error specification for fecundity parameters. See `LifeHistoryObs()`.

`SRR list`. Observation error specification for stock-recruitment parameters. See `LifeHistoryObs()`.

`Spatial list`. Observation error specification for spatial parameters. See `LifeHistoryObs()`.

`Depletion list`. Observation error specification for initial depletion. See `LifeHistoryObs()`.

`Misc list`. Miscellaneous additional objects.

This class is a placeholder. Each slot mirrors the corresponding sub-object of lifehistorydata and will eventually hold an error structure (bias, CV) for the associated life-history parameter. Slots are currently untyped lists and are not populated during model runs. See `LifeHistoryObs()` for further details.

### See Also

- `LifeHistoryObs()` for the constructor.
- obs for the enclosing object.
- lifehistorydata for the complementary observed-values class.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `obs-class`

---

`list-array-conversion`

*Convert Between Lists and Arrays*

---

### Description

Convert between lists of arrays and higher-dimensional arrays in a predictable and efficient way.

### Usage

```
List2Array(x, name = "Fleet", dim1 = "Sim", pos = NULL)

Array2List(x, pos = 2, index = NULL)
```

### Arguments

| | |
|---|---|
| `x` | A list (for `List2Array`) or an array (for `Array2List`). |
| `name` | Character string giving the name of the new dimension created by `List2Array`. |
| `dim1` | Character string giving the name of the first dimension when combining a list of vectors. Defaults to `"Sim"`. |
| `pos` | Integer or character specifying the position or name of the dimension to add (`List2Array`) or split (`Array2List`). |
| `index` | Optional integer vector specifying which slices to extract in `Array2List`. Defaults to all slices. |

### Details

These functions provide inverse operations for reshaping data between list and array representations while preserving dimension structure and dimnames.

**List2Array**

Combines a list of vectors or arrays into a single array by adding a new dimension corresponding to list elements. All list elements must have compatible dimensions. The new dimension has length equal to the length of the list.

**Array2List**

Splits an array into a list by extracting slices along a specified dimension. Each list element is an array with the remaining dimensions preserved.

### Value

- `List2Array()`: an array with one additional dimension
- `Array2List()`: a named list of arrays

### Examples

```
x <- list(
  Cod = array(
    1:6,
    dim = c(2, 3),
    dimnames = list(
      Sim  = 1:2,
      Year = 2023:2025
    )
  ),
  Haddock = array(
    7:12,
    dim = c(2, 3),
    dimnames = list(
      Sim  = 1:2,
      Year = 2023:2025
    )
  )
)

array <- List2Array(x, name = "Stock", pos = 2)
dimnames(array)

Array2List(array) # convert back to x
```

---

## `ListParallelWorkers` *List and Clean Up Parallel Worker Processes*

---

### Description

Clean up orphan worker processes

### Usage

```
ListParallelWorkers()

CleanupOrphanWorkers(dry_run = TRUE)
```

### Arguments

| | |
|---|---|
| `dry_run` | Logical. If `TRUE` (default), `CleanupOrphanWorkers()` reports what would be terminated without actually terminating anything. |

### Details

`future::multisession` (the backend `SetupParallel()` uses) spawns separate, persistent R processes for its workers. If the R session that created them exits abnormally rather than calling `DisableParallel()` those worker processes are not automatically cleaned up; each keeps running (holding whatever memory it had at the time) until manually killed or the machine restarts.

- `ListParallelWorkers()` finds every such worker process on this machine and reports whether its parent R session is still alive.
- `CleanupOrphanWorkers()` terminates only the ones whose parent is confirmed gone.

### Value

`ListParallelWorkers()` returns a `data.frame` with columns `pid`, `parent_pid` (`NA` if it could not be determined), `parent_alive` (`NA` if undetermined), and `orphan` (`TRUE` only when `parent_pid` is known and confirmed dead). `CleanupOrphanWorkers()` invisibly returns the same data.frame restricted to the orphaned rows that were (or, under `dry_run = TRUE`, would be) terminated.

### See Also

`SetupParallel()`, `DisableParallel()`, `CheckParallel()`

---

## LoadArgs *Load Function Arguments into an Environment*

---

### Description

Assigns the default argument values of a function to `envir`, skipping any argument that already exists in that environment as a non-function object. Useful during interactive development for populating the workspace with a function's expected inputs without overwriting variables already set.

### Usage

```
LoadArgs(fun = "Simulate", envir = .GlobalEnv, debug = FALSE)
```

### Arguments

| | |
|---|---|
| `fun` | Character string naming a function, or a function object. Default `"Simulate"`. |
| `envir` | Environment to assign default values into. Default `.GlobalEnv`. |
| `debug` | Logical. If `TRUE`, emits a message for each argument indicating whether it already exists in `envir` and whether it is being assigned. Default `FALSE`. |

### Value

`NULL` invisibly. Called for its side effect of populating `envir`.

### Examples

```
# Populate the global workspace with the default arguments of [CalcSurvival()],
# without overwriting any variables that are already defined:
## Not run:
NaturalMortality <- my_array   # already set; will not be overwritten
LoadArgs("CalcSurvival")
# FishingMortality, PlusGroup, SpawnTimeFrac, and Semelparous are now
# assigned their defaults in .GlobalEnv; NaturalMortality is unchanged.

## End(Not run)
```

---

## `Log` *Display Log Messages*

---

### Description

Prints any messages stored in the `Log` slot of an S4 object to the console using `cli` formatting. Messages are grouped into assumption, warning, and error sections; within each section, entries are further grouped by the name of the function that recorded them. Identical messages recorded for several MPs, years, or simulations are printed once, with the MPs, years, and simulations listed after the message.

### Usage

```
Log(object, type = NULL)
```

### Arguments

| | |
|---|---|
| `object` | An S4 object with a `Log` slot. See e.g. om, hist, mse, stock, data, advice. |
| `type` | Character vector. Restrict the printed sections to one or more of `'assumption'`, `'warning'`, `'error'`. Default `NULL` prints all sections. |

### Details

Log entries fall into three types:

- `'assumption'`: something was inferred or auto-corrected (e.g. a missing value defaulted, a selectivity schedule rescaled to a maximum of 1) and no further action is required.
- `'warning'`: something is likely wrong with the object or its inputs and should be reviewed.
- `'error'`: a problem serious enough that results built from the object should not be trusted.

### Value

Invisibly returns `NULL`. Called for its side effect of printing warnings to the console.

### See Also

DeleteLogs()

### Examples

```
## Not run:
  Log(my_object)
  Log(my_object, type = 'error')

## End(Not run)
```

---

`logit` *Logit and Inverse Logit (Sigmoid) Functions*

---

### Description

`logit()` transforms a probability to the log-odds scale. `ilogit()` is its inverse — the sigmoid function — mapping any real number back to [0, 1].

### Usage

```
logit(p)

ilogit(x)
```

### Arguments

| | |
|---|---|
| `p` | A numeric vector of probabilities. Values must be in the open interval (0, 1); `NaN` is returned for values outside this range. |
| `x` | A numeric vector of real-valued log-odds. |

### Details

The two functions are mutual inverses:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

$$\text{ilogit}(x) = \frac{1}{1+e^{-x}}$$

so that `ilogit(logit(p)) == p` and `logit(ilogit(x)) == x` (up to floating-point precision).

### Value

- `logit(p)` returns a numeric vector of log-odds in (-Inf, Inf).
- `ilogit(x)` returns a numeric vector of probabilities in (0, 1).

### Examples

```
logit(0.5)   # 0
logit(0.9)   # ~2.197

ilogit(0)    # 0.5
ilogit(2)    # ~0.880

# Round-trip
p <- c(0.1, 0.5, 0.9)
all.equal(p, ilogit(logit(p)))  # TRUE
```

---

## makeMOM *Utility for making multi-OMs*

---

### Description

Converts an OM to a single stock, single fleet MOM.

### Usage

```
makeMOM(..., silent = FALSE)
```

### Arguments

| | |
|---|---|
| `...` | An `OM`. |
| `silent` | Should messages be printed out to the console? |

### Value

A class `MOM` object.

### Author(s)

Q. Huynh

### Examples

```
MOM <- makeMOM(testOM)
```

---

`makemov` *Calculates movement matrices from user inputs for fraction in each area (fracs) and probability of staying in areas (prob)*

---

## Description

A function for calculating a movement matrix from user specified unfished stock biomass fraction in each area. Used by simmov to generate movement matrices for an operating model.

## Usage

```
makemov(fracs = c(0.1, 0.2, 0.3, 0.4), prob = c(0.5, 0.8, 0.9, 0.95))
```

## Arguments

| | |
|---|---|
| `fracs` | A vector nareas long of fractions of unfished stock biomass in each area |
| `prob` | A vector of the probability of individuals staying in each area or a single value for the mean probability of staying among all areas |

## Author(s)

T. Carruthers

## See Also

simmov

---

`makemov2` *Calculates movement matrices from user inputs for fraction in each area (fracs) the relative fraction moving to other areas, plus a mean probability of staying in any given area.*

---

## Description

A function for calculating a movement matrix from user specified distribution among areas (v) and relative movement to other areas (solves for positive diagonal - vector of prob staying). Used by simmov2 to generate movement matrices for an operating model. There must be a prior on the positive diagonal of the movement matrix or these will tend to 1 and hence perfectly satisfy the requirement V = MV.

### Usage

```
makemov2(
  dist = c(0.05, 0.6, 0.35),
  prob = 0.5,
  probE = 1,
  frac_other = matrix(c(NA, 2, 1, 2, NA, 1, 1, 2, NA), nrow = 3, byrow = T),
  plot = F
)
```

### Arguments

| | |
|---|---|
| `dist` | A vector nareas long of fractions of unfished stock biomass in each area |
| `prob` | A vector of the probability of individuals staying in each area or a single value for the mean probability of staying among all areas |
| `probE` | The logit CV associated with prob (used as a penalty when optimizing for diagonal) |
| `frac_other` | A matrix nareas x nareas that specifies the relative fraction moving from one area to the others. The positive diagonal is unspecified. |
| `plot` | Should the convergence to a stable distribution be plotted? |

### Author(s)

T. Carruthers

### See Also

simmov2

---

## `MakeNamedList` *Create a Named List*

---

### Description

Creates a named list of a given length, optionally filling every element with the same value. A thin convenience wrapper around `stats::setNames()` and `base::vector()`.

### Usage

```
MakeNamedList(nms, value = NULL)
```

### Arguments

| | |
|---|---|
| `nms` | `character` vector of names for the list elements. |
| `value` | Optional. A single value assigned to every element. When `NULL` (default), all elements are initialised to `NULL`. |

### Value

A named list of length `length(nms)`. If `value` is supplied, every element is set to `value`; otherwise every element is `NULL`.

### Examples

```
# Empty named list
MakeNamedList(c("a", "b", "c"))

# Named list with a common value
MakeNamedList(c("x", "y"), value = 0)
```

---

## makeRel *MICE relationships for multi-OM*

---

### Description

Generate a MICE `Rel` object, with `predict` and `simulate` methods, for multiMSE. Currently implements intra-stock dynamics via density-dependent processes.

### Usage

```
makeRel(type = "DDM", stock = 1, CV = 0, ...)

## S3 method for class 'Rel'
print(x, ...)

## S3 method for class 'Rel'
predict(object, newdata, ...)

## S3 method for class 'Rel'
simulate(object, nsim = 1, seed = 1, ...)
```

### Arguments

| | |
|---|---|
| `type` | String to indicate the type of stock interaction. `"DDM"` for density-dependent natural mortality. |
| `stock` | The index position of the stock in the MOM. |
| `CV` | Coefficient of variation of the predicted value for `simulate`. Used to pass values to the operating model. |
| `...` | Additional arguments depending on `type`. See details below. |
| `x` | For `print.Rel`, a `Rel` class object from `make_Rel`. |
| `object` | A `Rel` class object from `make_Rel`. |
| `newdata` | A data frame to provide values of predictor variables with which to calculate the response variable. |
| `nsim` | The number of simulations. |
| `seed` | Integer to specify the seed for the random number generator. |

## Value

A class "Rel" object to pass to `MOM@Rel`.

## Density-dependent M ("DDM")

Natural mortality (M) is a linear function of stock depletion in terms to total biomass (B) in year y (Forrest et al. 2018):

$$M_y = M_0 + (M_1 + M_0)(1 - B_y/B_0)$$

with a constraint that $M_y = M_0$ if $B_y > B_0$

Provide the following arguments:

- `M0`: Natural mortality as B approaches B0. Vector `[nsim]`
- `M1`: Natural mortality as B approaches zero. Vector `[nsim]`
- Optional `B0`: Unfished biomass. Calculated from stock-recruit `alpha` and `beta` and unfished biomass per recruit at M = M0. Vector `[nsim]`

## Author(s)

Q. Huynh

## References

Forrest, R., Holt, K., and Kronlund, A. 2018. Performance of alternative harvest control rules for two Pacific groundfish stocks with uncertain natural mortality: Bias, robustness and trade-offs. Fisheries Research 206: 259–286. doi:10.1016/j.fishres.2018.04.007

## Examples

```
# Depensatory natural mortality
Rel <- makeRel(type = "DDM", M0 = 0.8, M1 = 0.2, CV = 0.1)

# Predict M when B/B0 = 0.1
pred <- predict(Rel, newdata = data.frame(B_1 = 0.1, B0_1 = 1))

# Simulate values of M with CV = 0.1
Rel$fitted.values <- pred
simulate(Rel, nsim = 10, seed = 1)

# Add Rel to MOM
MOM <- makeMOM(testOM)
MOM@Rel <- list(Rel)
```

---

`makeTransparent` *Make colors transparent*

---

### Description

Make colors transparent

### Usage

```
makeTransparent(someColor, alpha = 100)
```

### Arguments

| | |
|---|---|
| `someColor` | Character string describing color |
| `alpha` | transparency |

### Author(s)

T. Carruthers

---

`ManagementScheduleTable` *Build a markdown table of the TAC calculation/implementation schedule*

---

### Description

Shows which projection years are management years (years in which an MP is actually called) and which calendar year of data the `DataLag` rule calls for at each one. See `DataLag()` for a full explanation of what the data lag means and how the data year for a given management year is worked out in practice.

### Usage

```
ManagementScheduleTable(
  FirstYear,
  DataLag,
  Interval,
  nYears = Interval * 2 + 1,
  print = TRUE
)
```

### Arguments

| | |
|---|---|
| `FirstYear` | Numeric, or an om, hist, or mse object. If numeric, the first implementation year (the first management year). If an `om` object, `FirstYear`, `DataLag`, and `Interval` are derived from it (`MPStartYear` if set, otherwise `CurrentYear + 1`; `DataLag()`; and `Interval()`), and the corresponding arguments below are ignored. |
| `DataLag` | Numeric. Years of data lag applied at each management year. |
| `Interval` | Numeric. Management update interval, in years. |
| `nYears` | Integer. Number of implementation years to show. Default shows 3 full management cycles. |
| `print` | Logical. Print the data.frame to console with `cat()`? |

### Details

The "DataYear used" column follows the `DataLag` rule described in `DataLag()`: for each management year, the data year is one year before it, shifted back by a further `DataLag` years. See `DataLag()` for how this interacts with `MPStartYear` and `InterimAdvice`.

### Value

if `print = TRUE` invisibly returns the underlying data.frame, and `cat()`s a markdown table. If `print == FALSE`, returns the data.frame

### See Also

`DataLag()` for what the data lag means and how it is applied, `Interval()` for the management update frequency.

---

## `Maturity` *Maturity Constructor and Accessors*

---

### Description

Construct a maturity object defining the maturity schedule for a stock, or access and replace the `Maturity` slot of a stock and its individual slots. A `Maturity` object is required for all stock objects.

### Usage

```
Maturity(
  Pars = list(),
  Model = NULL,
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  MeanAtWeight = NULL,
  Classes = NULL,
```

```
  Semelparous = FALSE,
  Misc = list()
)

Semelparous(x)

Semelparous(x) <- value

Maturity(x) <- value
```

### Arguments

| | |
|---|---|
| `Pars` | `list`. Named list of maturity parameters. Element names must match the arguments of a built-in maturity model (see `MaturityModels()`). When `Pars` is non-empty and `Model` is `NULL`, the model is inferred automatically from the parameter names. When `Pars` is a stock object, `Maturity()` acts as a pass-through accessor and returns `x@Maturity`; when `Pars` is a list of stock objects, `Model` is treated as an integer index selecting which stock's `Maturity` slot to return — see *Pass-Through Access*. See also Specifying Biological and Fleet Schedules for the full set of accepted input formats. Default `list()`. |
| `Model` | `character(1)`, `function`, or `integer(1)`. Maturity model identifier. When `NULL` (default), the model is inferred from `Pars`. May be set to a character string naming a built-in model or to a custom R function. When `Pars` is a list of stock objects, `Model` is an integer index — see *Pass-Through Access* and Specifying Biological and Fleet Schedules. |
| `MeanAtAge` | `array` or `NULL`. Mean maturity-at-age with named dimensions `Sim × Age × Year`. Values should range from 0 (immature) to 1 (fully mature). Supply directly when bypassing the model-based approach (`Pars = list()`). Only the years at which values *change* need to be included; `Extend()` fills the remainder. When `Pars` contains a matched model, any values here are **overwritten** during `Populate()`. A numeric vector of length `nAge` is also accepted and converted to a `1 × nAge × 1` array automatically. Default `NULL`. |
| `MeanAtLength` | `array` or `NULL`. Mean maturity-at-length with named dimensions `Sim × Class × Year`. Populated automatically during `Populate()` when an at-length maturity model is used; may also be supplied directly. When `MeanAtLength` is populated and `MeanAtAge` is not, `Populate()` converts it to `MeanAtAge` via the age-length key (`ALK`) from a length object. A numeric vector is also accepted and converted to a `1 × nClass × 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`. Default `NULL`. |
| `MeanAtWeight` | `array` or `NULL`. Mean maturity-at-weight with named dimensions `Sim × Class × Year`. Populated automatically during `Populate()` when an at-weight maturity model is used; may also be supplied directly. When `MeanAtWeight` is populated and `MeanAtAge` is not, `Populate()` converts it to `MeanAtAge` via the age-weight key (`AWK`) from a weight object. Requires a populated |

| | |
|---|---|
| | weight with a non-NULL CVatAge slot so that the AWK exists. A numeric vector is also accepted and converted to a 1 × nClass × 1 array automatically, using Classes if supplied or the weight object's own Classes otherwise; if Classes differs from the weight object's, the AWK is recalculated on Classes. Default NULL. |
| Classes | numeric or NULL. Age, length, or weight class midpoints corresponding to the MeanAt* array in use. Default NULL. |
| Semelparous | logical(1) or array. Controls post-spawning mortality. FALSE (default) disables post-spawn mortality. TRUE sets post-spawn mortality equal to the maturity ogive after Populate() — see *Semelparity* for full details. A custom Sim × Age × Year array may also be supplied directly to specify age- and time-varying post-spawn mortality that differs from the maturity schedule. |
| Misc | list. Named list of optional user-defined inputs stored on the object and passed through to custom model functions. Default list(). |
| x | A maturity object for slot accessors, or a stock object for Maturity<-. |
| value | For Maturity<-: a maturity object. For Semelparous<-: a logical(1) or a Sim × Age × Year array. |

## Details

### Specifying Maturity:

There are two ways to define the maturity schedule; see Specifying Biological and Fleet Schedules for full details on input formats and the rules that govern how Pars, Model, and MeanAt* arrays interact.

**Model-based** (recommended): supply Pars with named parameters matching a built-in model (see MaturityModels()). If Model = NULL and the parameter names uniquely match a model, the model is resolved automatically. The relevant MeanAt* array is then populated by Populate() when the stock is added to an OM():

```
# Logistic length-based ogive - model inferred from Pars names
mat <- Maturity(Pars = list(L50 = 40, L50_95 = 8))

# Stochastic L50 across simulations
mat <- Maturity(Pars = list(L50 = c(35, 45), L50_95 = 8))

# Time-varying L50 (shifts in 2010; Extend() fills the rest)
L50_arr <- array(c(40, 45), dim = c(1, 2),
                 dimnames = list(Sim = 1, Year = c(1990, 2010)))
mat <- Maturity(Pars = list(L50 = L50_arr, L50_95 = 8))

# Inter-annual random walk on L50 (log-normal, SD = 0.05)
mat <- Maturity(Pars = list(L50 = c(35, 45), L50SD = 0.05, L50_95 = 8))
```

**Direct array**: supply any of MeanAtAge, MeanAtLength, or MeanAtWeight with Pars = list(). MeanAtAge takes precedence if more than one is populated:

```
ages   <- 0:20
mat_aa <- Maturity(MeanAtAge = 1 / (1 + exp(-0.5 * (ages - 5))))

lens   <- seq(5, 120, by = 5)
mat_al <- Maturity(
  MeanAtLength = array(
    1 / (1 + exp(-log(19) * (lens - 40) / 8)),
    dim      = c(1, length(lens), 1),
    dimnames = list(Sim = 1, Class = lens, Year = 1990)
  )
)

# A plain numeric vector is also accepted for `MeanAtLength`/`MeanAtWeight`
# and converted automatically, using `Classes` for the class labels:
mat_al2 <- Maturity(
  MeanAtLength = 1 / (1 + exp(-log(19) * (lens - 40) / 8)),
  Classes      = lens
)
```

**Semelparity:**

Post-spawning mortality is controlled by the `Semelparous` slot, which after `Populate()` is always a `Sim × Age × Year` array. Each cell gives the fraction of individuals at that age dying immediately after spawning. This mortality is applied as a multiplier `(1 - Semelparous)` on top of standard `exp(-Z)` survival — it is an *additional* source of mortality, not a substitute for natural mortality.

The default (`Semelparous = FALSE`) sets all cells to 0: no post-spawn mortality, appropriate for iteroparous species.

Setting `Semelparous = TRUE` sets the array equal to `MeanAtAge` after population:

- Fully mature ages (maturity  1): post-spawn survival  0 — complete die-off.
- Ages on the maturity ogive (0 < maturity < 1): proportional post-spawn mortality.
- Immature ages (maturity  0): post-spawn survival  1 — unaffected.

This assumption — that the post-spawn mortality schedule equals the maturity ogive — is appropriate for obligate semelparous species such as Pacific salmon, where all maturing individuals spawn once and die. For species with partial semelparity, or where post-spawn mortality affects only older or larger mature fish, supply a custom `Sim × Age × Year` array directly:

```
ages <- 0:20
# Post-spawn mortality applies only to ages >= 5, ramping from 0 to 0.8
sem_arr <- array(
  pmin(pmax((ages - 5) / 10, 0), 0.8),
  dim      = c(1, length(ages), 1),
  dimnames = list(Sim = 1, Age = ages, Year = 1990)
)
mat <- Maturity(Pars = list(L50 = 40, L50_95 = 8), Semelparous = sem_arr)
```

Note: after `Populate()`, `Semelparous` is always an array.

**At-Length and At-Weight Models:**

Models whose class contains `"at-Length"` produce `MeanAtLength`, which is converted to `MeanAtAge` via the `ALK` — a populated `Length()` object must be passed to `Populate()`. Models whose class contains `"at-Weight"` produce `MeanAtWeight`, converted via the `AWK` — a populated `Weight()` object with non-`NULL` `CVatAge` (so the `AWK` exists) must be passed to `Populate()`. At-weight maturity models are rare in practice.

The same conversions apply when `MeanAtLength` or `MeanAtWeight` are supplied directly by the user.

**Pass-Through Access from a Stock:**

When `Pars` is a stock object, `Maturity()` returns the `Maturity` slot directly:

```
Maturity(my_stock)              # returns my_stock@Maturity
Maturity(my_stock) <- my_mat    # replaces my_stock@Maturity
```

When `Pars` is a list of stock objects, `Model` is an integer index selecting which stock's slot to return:

```
Maturity(stock_list, 2)   # returns stock_list[[2]]@Maturity
```

**Slot Accessors:**

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Pars(mat)           <- list(L50 = 40, L50_95 = 8)
Model(mat)          <- "LogisticLength"
MeanAtAge(mat)      <- my_array
MeanAtLength(mat)   <- my_length_array
MeanAtWeight(mat)   <- my_weight_array
Semelparous(mat)    <- TRUE
Classes(mat)        <- seq(0, 80, by = 5)
```

## Value

- `Maturity()` returns a maturity object. If `Pars` is a stock, returns `x@Maturity`.
- `Maturity<-` returns the stock `x` with the `Maturity` slot replaced
- `Semelparous()` returns the `Semelparous` slot from `x` (a scalar before `Populate()`, a `Sim × Age × Year` array after).
- `Semelparous<-` returns `x` with the `Semelparous` slot updated and the object re-validated.

## See Also

- maturity for the class definition and slot-level documentation.
- `MaturityModels()` for available maturity models and required parameter sets.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.
- `Length()` for the companion length schedule, required when using at-length maturity models.

- `Weight()` for the companion weight schedule, required when using at-weight maturity models.

Other maturity: `maturity-class`

### Examples

```
# See MaturityModels() for all available built-in maturity models and the
# parameter names required for each.
MaturityModels()

# ---- Model-based specification ----

## Constant parameters (scalar) - same value for every simulation and year
mat <- Maturity(Pars = list(L50 = 40, L50_95 = 8))

## Stochastic across simulations - L50 drawn from Uniform(lower, upper)
## once per simulation; L50_95 fixed in all simulations
mat <- Maturity(Pars = list(L50    = c(35, 45),
                            L50_95 = 8))

## Inter-annual random walk on L50 - log-normal with 5% CV
## L50SD is removed from Pars after use
mat <- Maturity(Pars = list(L50    = c(35, 45),
                            L50SD  = 0.05,
                            L50_95 = 8))

## Time-varying parameters with named change-point arrays
## Only the years where values change need to be supplied; Extend() fills
## all intermediate and future years by forward-filling automatically.
nSim <- 48

## L50 shifts in 2010 (two change points: 1990 and 2010)
L50_arr <- array(
  c(rep(40, nSim), rep(45, nSim)),
  dim      = c(nSim, 2),
  dimnames = list(Sim = seq_len(nSim), Year = c(1990, 2010))
)
mat <- Maturity(Pars = list(L50    = L50_arr,
                            L50_95 = 8))

# ---- Direct array specification ----

## Supply MeanAtAge directly (Pars left empty).
## A plain numeric vector of length nAge is accepted; treated as a single
## simulation, single year and replicated by Extend() as needed.
ages   <- 0:20
mat_aa <- Maturity(MeanAtAge = 1 / (1 + exp(-0.5 * (ages - 5))))

## Named Sim x Age x Year array - two change-point years, single simulation
maa <- array(
  c(1 / (1 + exp(-0.5 * (ages - 5))),    # 1990 ogive
    1 / (1 + exp(-0.5 * (ages - 7)))),   # 2010 ogive (L50 shifted later)
```

```
  dim      = c(1, length(ages), 2),
  dimnames = list(Sim = 1, Age = ages, Year = c(1990, 2010))
)
mat_arr <- Maturity(MeanAtAge = maa)

## Supply MeanAtLength directly - converted to MeanAtAge via the ALK
## during Populate() when a Length object is provided.
lens   <- seq(5, 120, by = 5)
mat_al <- Maturity(
  MeanAtLength = array(
    1 / (1 + exp(-log(19) * (lens - 40) / 8)),
    dim      = c(1, length(lens), 1),
    dimnames = list(Sim = 1, Length = lens, Year = 1990)
  )
)

# ---- Semelparity ----

## Semelparous = TRUE: post-spawn mortality equals the maturity ogive.
## Fully mature fish die with probability 1; fish on the ogive die
## proportionally; immature fish are unaffected.
## Appropriate for obligate semelparous species (e.g. Pacific salmon).
mat_sem <- Maturity(Pars = list(L50 = 40, L50_95 = 8), Semelparous = TRUE)

## Custom post-spawn mortality array - partial semelparity by age.
## Here mortality ramps from 0 at age 5 to 0.8 at age 15, then is
## constant. This differs from the maturity ogive and must be supplied
## as a named array.
sem_arr <- array(
  pmin(pmax((ages - 5) / 10, 0), 0.8),
  dim      = c(1, length(ages), 1),
  dimnames = list(Sim = 1, Age = ages, Year = 1990)
)
mat_psem <- Maturity(Pars = list(L50 = 40, L50_95 = 8), Semelparous = sem_arr)

## After Populate(), Semelparous is always an array - do not test == TRUE.
## Use any(Semelparous(mat) > 0) to check whether post-spawn mortality
## is active on a populated object.

# ---- Slot accessors ----

mat <- Maturity(Pars = list(L50 = 40, L50_95 = 8))

## Read slots
MeanAtAge(mat)      # NULL until Populate() is called
MeanAtLength(mat)   # NULL unless an at-length model or direct array is used
Semelparous(mat)    # scalar FALSE before Populate(); array after

## Replace slots
Semelparous(mat) <- TRUE
Classes(mat)     <- seq(0, 80, by = 5)

# ---- Attaching to a Stock ----
```

```
stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
Maturity(stk) <- Maturity(Pars = list(L50 = 40, L50_95 = 8))
Maturity(stk)

# ---- Populate ----

## The final populated object (typically done internally).
## At-length models require a Length object passed to Populate().
pop_mat <- Populate(Maturity(stk),
                    Ages   = Ages(MaxAge = 20),
                    Length = Length(Pars = list(Linf = 100, K = 0.2, t0 = -0.1))
)
pop_mat
```

---

`maturity-class` *The* `maturity` *S4 Class*

---

## Description

Defines the maturity schedule for a stock object. Maturity may be expressed as a function of age, length, or weight. Objects are typically created via `Maturity()`, which documents all parameters, validates inputs, and infers the maturity model automatically when possible.

## Details

Direct construction via `methods::new()` is not recommended; use `Maturity()` instead, which handles model inference and object validation.

Only one of `MeanAtAge`, `MeanAtLength`, or `MeanAtWeight` need be populated; `MeanAtAge` takes precedence if more than one is present.

After `Populate()`, `Semelparous` is always a `Sim × Age × Year` array. Do not test `Semelparous(mat) == TRUE` on a populated object; check `any(Semelparous(mat) > 0)` instead.

## Slots

`Pars list`. Named list of maturity model parameters whose names correspond to those expected by `Model`. See `MaturityModels()` for parameter sets and Specifying Biological and Fleet Schedules for accepted input formats.

`Model function` or `character(1)`. Maturity model identifier, matched to one of `MaturityModels()`. Set automatically from the parameter names when `Pars` is supplied without an explicit model.

`MeanAtAge array`. Mean maturity-at-age with named dimensions `Sim`, `Age`, and `Year`. Values range from 0 (immature) to 1 (fully mature). Populated automatically by `Populate()` when `Pars` and `Model` are set; may also be supplied directly when `Pars` is empty. When `Pars` contains a matched model, any existing values are overwritten. See Specifying Biological and Fleet Schedules.

`MeanAtLength array`. Mean maturity-at-length with named dimensions `Sim`, `Length`, and `Year`. Populated automatically when an at-length maturity model is used, or may be supplied directly. Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). Converted to `MeanAtAge` via the `ALK` during `Populate()` when `MeanAtAge` is not already populated. See Specifying Biological and Fleet Schedules.

`MeanAtWeight array`. Mean maturity-at-weight with named dimensions `Sim`, `Weight`, and `Year`. Populated automatically when an at-weight maturity model is used, or may be supplied directly. Values are evaluated at the midpoint of each bin; the `Class` dimension is labelled by bin lower bounds (see `Classes`). Converted to `MeanAtAge` via the `AWK` during `Populate()` when `MeanAtAge` is not already populated. Requires a populated weight object with a non-`NULL` `CVatAge` slot (so that the `AWK` exists). See Specifying Biological and Fleet Schedules.

`Classes numeric`. Age classes (in years) or lower bounds of length or weight bins, corresponding to the `MeanAt*` array in use. For size bins, bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength`/`MeanAtWeight` are evaluated at bin midpoints, not at these lower bounds.

`Semelparous logical` or `array`. Controls post-spawning mortality. Before `Populate()`, a scalar `TRUE` or `FALSE`. After `Populate()`, always a `Sim × Age × Year` array where each cell gives the fraction of individuals at that age dying immediately after spawning. The default (`FALSE`) sets all cells to 0 (no post-spawn mortality). `TRUE` sets the array equal to `MeanAtAge`, so that post-spawn mortality tracks the maturity ogive — fully mature fish die with probability 1, fish on the ogive die proportionally, and immature fish are unaffected. A custom array may also be supplied directly. See `Maturity()` for details.

`Misc list`. Used internally.

## See Also

- `Maturity()` for the constructor and accessor functions.
- `MaturityModels()` for available maturity models and their required parameters.
- `Populate()` for array population.
- `Semelparous()` to retrieve or set the post-spawn mortality array.
- `Length()` and `Weight()` for the companion schedules required by at-length and at-weight maturity models.
- Specifying Biological and Fleet Schedules for the full description of how `Pars`, `Model`, and `MeanAt*` arrays interact.

Other maturity: `Maturity()`

`Maturity-Models` *Maturity Models: Age, Length, and Weight*

## Description

Logistic maturity models

## Usage

```
MaturityModelsWeight(full = TRUE, print = TRUE)

MaturityAtAge(Ages, A50, A50_95)

MaturityAtLength(Length, L50, L50_95)

MaturityAtWeight(Weight, W50, W50_95)

MaturityModels(full = TRUE, print = TRUE)

MaturityModelsLength(full = TRUE, print = TRUE)

MaturityModelsAge(full = TRUE, print = TRUE)
```

## Arguments

| | |
|---|---|
| `full` | Logical. Provide a complete table (TRUE) or just the model names (FALSE)? |
| `print` | Logical. Print out the results (TRUE) or just return the data.frame (FALSE)? |
| `Ages` | numeric vector of age classes (for age-based models) |
| `A50` | age at 50% maturity (Logistic-at-Age) |
| `A50_95` | interval between `A50` and age at 95% maturity (Logistic-at-Age) |
| `Length` | numeric vector of length classes (for length-based models) |
| `L50` | length at 50% maturity (Logistic-at-Length) |
| `L50_95` | interval between `L50` length at 95% maturity (Logistic-at-Length) |
| `Weight` | numeric vector of weight classes (for weight-based models) |
| `W50` | weight at 50% maturity (Logistic-at-Weight) |
| `W50_95` | interval between `W50` weight at 95% maturity (Logistic-at-Weight) |

## Details

Logistic models produce standard increasing maturity curves.

- Age-based models return proportion mature by age.

- Length-based models return proportion mature by length.
- Weight-based models return proportion mature by weight.

All maturity models use a logistic 50/95 parameterization:

$$M(x) = \frac{1}{1 + \exp\left(-\ln(19) \cdot \frac{x - x_{50}}{x_{95} - x_{50}}\right)}$$

at-length and at-weight schedules are converted internally to at-age using the age-length age-weight key respectively.

## Value

Each function returns a numeric vector of proportion mature at each age, length, or weight. `MaturityModels()` invisibly returns a data frame describing available models.

Prints to console and invisible data.frame or model names

## See Also

`Ages()`, `Length()`, `Maturity()`, `Weight()`, `Stock()`

## Examples

```
Ages <- 0:15
Length <- seq(10, 100, 10)
Weight <- seq(1, 50, 5)

M_age <- MaturityAtAge(Ages, A50 = 5, A50_95 = 3)

M_length <- MaturityAtLength(Length, L50 = 50, L50_95 = 20)

M_weight <- MaturityAtWeight(Weight, W50 = 20, W50_95 = 15)


plot(Ages, M_age, type = "l", lwd = 2,
     xlab = "Age (years)", ylab = "Proportion Mature",
     main = "Maturity-at-Age")

plot(Length, M_length, type = "l", lwd = 2,
     xlab = "Length", ylab = "Proportion Mature",
     main = "Maturity-at-Length")

plot(Weight, M_weight, type = "l", lwd = 2,
     xlab = "Weight", ylab = "Proportion Mature",
     main = "Maturity-at-Weight")
```

## ML2D *Depletion and F estimation from mean length of catches*

### Description

A highly dubious means of getting very uncertain estimates of current stock biomass and (equilibrium) fishing mortality rate from growth, natural mortality rate, recruitment and fishing selectivity.

### Usage

```
ML2D(OM, ML, nsim = 100, ploty = T, Dlim = c(0.05, 0.6))
```

### Arguments

| | |
|---|---|
| OM | An object of class 'OM' |
| ML | A estimate of current mean length of catches |
| nsim | Number of simulations |
| ploty | Produce a plot of depletion and F |
| Dlim | Limits on the depletion that is returned as a fraction of unfished biomass. |

### Value

An object of class 'OM' with 'D' slot populated

### Author(s)

T. Carruthers

## MMSE-legacy-class *Class* 'MMSE'

### Description

A Multi Management Strategy Evaluation object that contains information about simulation conditions and performance of MPs for a multi-stock, multi-fleet operating model.

**Slots**

`Name` Name of the MMSE object. Single value. Character string

`nyears` The number of years for the historical simulation. Single value. Positive integer

`proyears` The number of years for the projections - closed loop simulations. Single value. Positive integer

`nMPs` Number of management procedures simulation tested. Single value. Positive integer.

`MPs` The names of the MPs that were tested. Vector of length nMPs. Character strings.

`MPcond` The MP condition. Character ('bystock': an MP per stock, 'byfleet' and MP per stock and fleet, 'MMP' an MP for all stocks and fleets)

`MPrefs` The names of the MPs applied for each stock (row) and fleet (column). An array.

`nsim` Number of simulations. Single value. Positive integer

`nstocks` Number of stocks. Single value. Positive integer

`nfleets` Number of fleets. Single value. Positive integer

`Snames` Names of the stocks

`Fnames` Names of the fleets (matrix nstocks x nfleets)

`Stocks` The stock operating model objects. List of Stocks

`Fleets` The fleet operating model objects. Hierarchical list, fleets nested in stocks.

`Obss` The fleet specific observation error operating model objects. Hierarchical list, fleets nested in stocks.

`Imps` The fleet specific implementation error operating model objects. Hierarchical list, fleets nested in stocks.

`OM` A table of sampled parameters of the operating model. Data frame of nsim rows.

`Obs` A table of sampled parameters of the observation model. Data frame of nsim rows.

`SB_SBMSY` Simulated spawning biomass relative to SBMSY over the projection. An array with dimensions: nsim, nStocks, nMPs, proyears. Non-negative real numbers

`F_FMSY` Simulated fishing mortality rate relative to FMSY over the projection. An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`N` Simulated stock numbers over the projection. An array with dimensions: nsim, nStocks, `maxage`+1, nMPs, proyears, nareas. Non-negative real numbers

`B` Simulated stock biomass over the projection. An array with dimensions: nsim, nStocks, nMPs, proyears. Non-negative real numbers

`SSB` Simulated spawning stock biomass over the projection. An array with dimensions: nsim, nStocks, nMPs, proyears. Non-negative real numbers

`VB` Simulated vulnerable biomass over the projection. An array with dimensions: nsim, nStocks, nMPs, proyears. Non-negative real numbers

`FM` Simulated fishing mortality rate over the projection. An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`SPR` A list of SPR values. Currently not used.

`Catch` Simulated catches (landings) over the projection. An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`Removals` Simulated removals (landings+discards) over the projection. An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`Effort` Simulated relative fishing effort in the projection years. An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`TAC` Simulated Total Allowable Catch (prescribed) over the projection (this is NA for input controls). An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`TAE` Simulated Total Allowable Effort (prescribed) over the projection (this is NA for output controls). An array with dimensions: nsim, nStocks, nFleets, nMPs, proyears. Non-negative real numbers

`BioEco` A named list of bio-economic output. Not currently used.

`RefPoint` Named list of annual MSY reference points MSY, FMSY, and SBMSY. Array with dimensions: nsim, nstocks, nMPs, nyears+proyears. Will be the same as `multiHist@Ref$ByYear` unless selectivity is changed by MP

`multiHist` The object of class `multiHist` containing information from the spool-up period.

`PPD` Posterior predictive data. List of `Data` objects at the end of the projection period (length `nMPs`)

`Misc` Miscellaneous output such as posterior predictive data

### Objects from the Class

Objects can be created by calls of the form `new('MMSE', Name, nyears, proyears, nMPs, MPs, nsim, OMtable, Obs,B_BMSYa, F_FMSYa, Ba, FMa, Ca, OFLa, Effort, PAA, CAA, CAL, CALbins)`

### Author(s)

T. Carruthers

---

## model-helpers *Selectivity, Retention, Weight, Fecundity, and Maturity Model Helpers*

---

### Description

Low-level curve functions used internally by `FecundityModels()`, `MaturityModels()`, `RetentionModels()`, `SelectivityModels()`, and `WeightModels()`.

### Usage

```
allometric(x, scale, exponent)

dnormal(lens, lfs, sl, sr)

double_normal(x, x5, xF, xMax)

logistic_50_95(x, x50, x50_95, asymp = 1)
```

**Arguments**

| | |
|---|---|
| `x` | Numeric vector of values (ages, lengths, or weights) at which to evaluate the curve. |
| `scale` | Multiplicative scale parameter (`allometric`). |
| `exponent` | Power exponent (`allometric`). |
| `lens` | Numeric vector of lengths (`dnormal`). |
| `lfs` | Length at full selection, the peak of the `dnormal` curve. |
| `sl` | Standard deviation of the ascending limb of `dnormal`. |
| `sr` | Standard deviation of the descending limb of `dnormal`. Use `Inf` for a flat-topped (one-sided) curve. |
| `x5` | Value of `x` at 5% of the maximum (ascending limb of `double_normal`). |
| `xF` | Value of `x` at the peak of the `double_normal` curve. |
| `xMax` | Maximum value on the descending limb of `double_normal`, expressed as a proportion between 0 and 1. |
| `x50` | Value of `x` at 50% of the asymptote (`logistic_50_95`). |
| `x50_95` | Interval between the 50% and 95% points (`logistic_50_95`). |
| `asymp` | Asymptote of the logistic curve. Defaults to `1`. |

**Details**

`allometric(x, scale, exponent)`

Returns `scale * x ^ exponent`. Suitable for length-weight and length-fecundity relationships.

`dnormal(lens, lfs, sl, sr)`

Double-normal (dome-shaped) selectivity curve evaluated at `lens`. The ascending limb uses standard deviation `sl` and the descending limb uses `sr`. Set `sr = Inf` for a flat-topped curve.

$$S(l) = 2^{-\left(\frac{l - l_{fs}}{s}\right)^2}$$

where $s = sl$ if $l \leq l_{fs}$ and $s = sr$ otherwise.

`double_normal(x, x5, xF, xMax)`

Parameterises and evaluates `dnormal` from three inputs: the 5% ascending point `x5`, the peak `xF`, and the descending plateau `xMax`. Returns a flat vector of ones if both `x5` and `xF` are zero.

`logistic_50_95(x, x50, x50_95, asymp)`

Standard logistic curve parameterised by the 50% point and the 50-to-95% interval:

$$S(x) = \frac{a}{1 + \exp\left(-\frac{\ln 19}{x_{50,95}}(x - x_{50})\right)}$$

where $a$ is `asymp`.

### Value

Each function returns a numeric vector of the same length as the primary input (`x` or `lens`).

### See Also

`FecundityModels()`, `MaturityModels()`, `RetentionModels()`, `SelectivityModels()`, `WeightModels()`

---

`MOM-legacy-class` *Class* `'MOM'`

---

### Description

An object containing all the parameters needed to control a multi-stock, multi-fleet MSE which can be build from component Stock, Fleet, Obs, and Imp objects.

### Details

Almost all of these inputs are a vector of length 2 which describes the upper and lower bounds of a uniform distribution from which to sample the parameter.

### Slots

`Name` Name of the operating model

`Agency` Name of the agency responsible for the management of the fishery. Character string

`Region` Name of the general geographic region of the fishery. Character string

`Sponsor` Name of the organization who sponsored the OM. Character string

`Latitude` Latitude (decimal degrees). Negative values represent the South of the Equator. Numeric. Single value

`Longitude` Longitude (decimal degrees). Negative values represent the West of the Prime Meridian. Numeric. Single value

`nsim` The number of simulations

`proyears` The number of projected years

`interval` The assessment interval - how often would you like to update the management system?

`pstar` The percentile of the sample of the management recommendation for each method

`maxF` Maximum instantaneous fishing mortality rate that may be simulated for any given age class

`reps` Number of samples of the management recommendation for each method. Note that when this is set to 1, the mean value of the data inputs is used.

`cpars` A hierarchical list nstock then nfleet long of custom parameters. Time series are a matrix nsim rows by nyears columns. Single parameters are a vector nsim long. See `validcpars()`

`seed` A random seed to ensure users can reproduce results exactly

`Source` A reference to a website or article from which parameters were taken to define the operating model

`Stocks` List of stock objects

`Fleets` List of Fleet objects

`Obs` Hierarchical List of Observation model objects Level 1 is stock, level 2 is fleet

`Imps` Hierarchical List of Implementation model objects Level 1 is stock, level 2 is fleet

`CatchFrac` A list nstock long, of matrices nsim x nfleet representing the fraction of current catches of the various fleets to each stock (each matrix is nsim by nfleet long and rows sum to 1 for each stock)

`Allocation` A list nstock long, of matrices nsim x nfleet representing the fraction of future TACs of the various fleets to each stock (each matrix is nsim by nfleet long and rows sum to 1 for each stock).

`Efactor` A list nstock long, of current effort factors by fleet (default is 1 - same as current effort)

`Complexes` A list of stock complexes. Each position is a vector of stock numbers (as they appear in StockPars) for which data should be aggregated and TAC recommendations split among stocks according to vulnerable biomass

`SexPars` A list of slots that control sex-specific dynamics, i.e., sex-specific spawning and hermaphroditism. More generally, controls spawning and moving abundance between stocks. See details.

`Rel` A list of biological / ecological relationships among stocks over-ridden if an MP of class 'MP_F" is supplied that is a multi-fleet MP.

## Objects from the Class

Objects can be created by calls of the form `new('MOM', Stock_list, Fleet_list, Obs_list, Imp_list)`.

## SexPars

- `SSBfrom` A nstock x nstock matrix that specifies the proportion of the spawning output of the row p stock for the column p' stock. A diagonal matrix means each stock is responsible for its own recruitment.
- `Herm` A list with each entry containing a matrix (nsim x maxage + 1) that specifies the proportion at age that moves from stock p to p' (sequential hermaphroditism). The names of the list should be of the form "H_p'_p" where p and p' are integers that identify the stock. Arrays can also be used (nsim x maxage + 1 x nyears + proyears) for time-varying values.
- `share_par` Optional. Logical to indicate whether stock-recruit, depletion, and observation/implementation parameters are mirrored between stocks. By default, TRUE.

## Author(s)

T. Carruthers and A. Hordyk

## See Also

Article on MOM and multiMSE: https://openmse.com/features-multimse/

---

`movestockCPP` *Apply the movement model to the stock for one time-step*

---

## Description

Apply the movement model to the stock for one time-step

## Usage

```
movestockCPP(nareas, maxage, mov, Number)
```

## Arguments

| | |
|---|---|
| `nareas` | The number of spatial areas |
| `maxage` | The maximum age |
| `mov` | Numeric matrix (nareas by nareas) with the movement matrix |
| `Number` | A numeric matrix (maxage+1, nareas) with current numbers-at-age in each area |

## Author(s)

A. Hordyk

---

`movfit_Rcpp` *Rcpp version of the Optimization function that returns the squared difference between user specified and calculated movement parameters.*

---

## Description

The user specifies the probability of staying in the same area and spatial heterogeneity (both in the unfished state). This function returns the squared difference between these values and those produced by the three logit movement model.

## Usage

```
movfit_Rcpp(par, prb, frac)
```

### Arguments

| | |
|---|---|
| `par` | Three parameters in the logit space that control the four probabilities of moving between 2 areas |
| `prb` | User specified probability that individuals in area 1 remain in that area (unfished conditions) |
| `frac` | User specified fraction of individuals found in area 1 (unfished conditions) |

### Details

This is paired with getmov to find the correct movement model.

### Author(s)

T. Carruthers with an amateur attempt at converting to Rcpp by A. Hordyk (but it works!)

---

## `MPCalcsNAs` *Fill any NAs arising from MPCalcs (hermaphroditism mode)*

---

### Description

Fill any NAs arising from MPCalcs (hermaphroditism mode)

### Usage

```
MPCalcsNAs(MPCalcs)
```

### Arguments

| | |
|---|---|
| `MPCalcs` | A list of arrays arising fromt the DLMtool function CalcMPDynamics() |

### Author(s)

T. Carruthers

## MPFailureRate *Per-MP Effort/TAC Solver Failure Rate*

### Description

Summarizes `SummarizeConvergence()` into a failure rate per MP, to help decide whether an MP's results are trustworthy enough to keep in the analysis, or whether it failed to converge too often and should be excluded.

### Usage

```
MPFailureRate(MSE)
```

### Arguments

| | |
|---|---|
| `MSE` | An mse object. |

### Value

A `data.frame` with one row per MP that has projection results: `MP`, `N_Failed` (number of distinct sim/year combinations with at least one logged convergence failure), `N_Total` (`nSim(MSE) * length(Years(MSE, 'Projection'))`), `FailureRate` (`N_Failed / N_Total`). Sorted descending by `FailureRate`.

### See Also

`SummarizeConvergence()`

## MPStartYear *MP Start Year for OM Objects*

### Description

Access or replace the `MPStartYear` slot of an om object: the first calendar year in which MPs are called during projection. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
MPStartYear(x)

MPStartYear(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; a single calendar year after `CurrentYear`, or `NULL`. |

### Details

`MPStartYear = NULL` (the default) calls MPs from the first projection year (`CurrentYear + 1`).

When the historical period ends before MPs will actually be applied (e.g. the OM is conditioned on data to 2024 but MPs will not be implemented until 2028), set `MPStartYear` to the first calendar year MPs should run. Projection years before `MPStartYear` are "interim" years:

- the MP is not called, and the TAC or Effort is instead taken from `InterimAdvice()`, which must then be supplied for every interim year;
- implementation error (`Imp`) is not applied;
- the OM still simulates the fishery and generates data for those years, so the MP's first call at `MPStartYear` sees data from the interim period (subject to `DataLag()`).

`MPStartYear` is the first management year; later management years follow every `Interval()` years from it. Performance metrics exclude interim years (see `PM()`).

### Value

- `MPStartYear()` returns the value of the `MPStartYear` slot.
- `MPStartYear<-()` returns `x` with the `MPStartYear` slot updated.

### See Also

- `OM()` for the constructor, om for the class definition.
- `InterimAdvice()` for the TAC/Effort used in interim years.
- `Interval()`, `DataLag()` for the other slots controlling the management timeline.
- `ManagementScheduleTable()` for previewing the resulting schedule.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM(CurrentYear = 2023, MPStartYear = 2028)
MPStartYear(om)
InterimAdvice(om) <- data.frame(Year = 2024:2027, Type = "TAC", Mean = 1000)
ManagementScheduleTable(om)
```

## MPtype *Management Procedure Type*

### Description

Management Procedure Type

### Usage

```
MPtype(MPs = NA)
```

### Arguments

| | |
|---|---|
| MPs | A vector of MP names. If none are provided function is run on all available MPs |

### Value

A data.frame with MP names, management type (e.g "Input", "Output") and management recommendations returned by the MP (e.g, TAC (total allowable catch), TAE (total allowable effort), SL (size-selectivity), and/or or Spatial)

### See Also

Required()

### Examples

```
## Not run:
library(openMSE)
MPtype(c("AvC", "curE", "matlenlim", "MRreal", "FMSYref"))

## End(Not run)
```

## mse-class *mse Class*

### Description

The mse class stores the complete results of a Management Strategy Evaluation. It extends timeseries with the operating model, management procedures, unfished reference states, biological reference points, and the historical time-series used to initialize the projections.

### Slots

`OM` The om object used as the operating model.

`MPs` Named list of management procedures evaluated in the MSE. Each element corresponds to one MP applied across all simulations.

`Unfished` An unfished object containing unfished equilibrium and dynamic reference trajectories. See unfished for details.

`Reference` A reference object containing biological and management reference points derived from the historical period. See reference for details.

`Hist` A timeseries object containing the historical time-series of population and fishery dynamics used to initialize the MSE projections. Use `Hist()` to recover the complete hist object, e.g. to pass to `Project()` with additional management procedures.

`PPD` List containing posterior predictive distribution outputs and diagnostics generated during the MSE.

`Log` `list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

`Misc` Named list for carrying arbitrary additional objects alongside the MSE results.

The `mse` class inherits all time-series slots from timeseries, covering the projection period. See timeseries for full slot descriptions including `Number`, `Biomass`, `Landings`, `Discards`, `Effort`, and fishing mortality arrays.

### See Also

hist, timeseries, om, unfished, reference, `Hist()`, `Advice()`

---

## `MSE-legacy-class` *Class* `'MSE'`

---

### Description

A Management Strategy Evaluation object that contains information about simulation conditions and performance of data-limited methods

### Slots

`Name` Name of the MSE object. Single value. Character string

`nyears` The number of years for the historical simulation. Single value. Positive integer

`proyears` The number of years for the projections - closed loop simulations. Single value. Positive integer

`nMPs` Number of management procedures simulation tested. Single value. Positive integer.

`MPs` The names of the MPs that were tested. Vector of length nMPs. Character strings.

`nsim` Number of simulations. Single value. Positive integer

`OM` Operating model parameters (last historical year used for time-varying parameters). Data.frame with `nsim` rows

`Obs` Observation parameters (last historical year used for time-varying parameters). Data.frame with `nsim` rows

`SB_SBMSY` Simulated spawning biomass relative to spawning BMSY over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`F_FMSY` Simulated fishing mortality rate relative to FMSY over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`N` Simulated total numbers over the projection. An array with dimensions: nsim, max-age+1, nMPs, proyears, nareas. Non-negative real numbers.

`B` Simulated stock biomass over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`SSB` Simulated spawning stock biomass over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`VB` Simulated vulnerable biomass over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`FM` Simulated fishing mortality rate over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`SPR` Named list with equilibrium and dynamic SPR. Each element is an array with dimensions: nsim, nMPs, proyears. Non-negative real numbers.

`Catch` Simulated catches (landings) over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`Removals` Simulated removals (catch + discards) over the projection. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`Effort` Simulated relative fishing effort in the projection years. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`TAC` Simulated Total Allowable Catch prescribed by MPs. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`TAE` Simulated Total Allowable Effort prescribed by MPs. An array with dimensions: nsim, nMPs, proyears. Non-negative real numbers

`BioEco` Named list with bio-economic output Only used if bio-economic parameters are included in OM

`RefPoint` Named list of annual MSY reference points MSY, FMSY, and SBMSY. Array with dimensions: nsim, nMPs, nyears+proyears. Will be the same as `Hist@Ref$ByYear` unless selectivity is changed by MP

`CB_hist` Simulated catches (landings) from the spool-up period. An array with dimensions: nsim, nyears. Non-negative real numbers

`FM_hist` Simulated fishing mortality rate from the spool-up period. An array with dimensions: nsim, nyears Non-negative real numbers

`SSB_hist` Simulated spawning stock biomass from the spool-up period. An array with dimensions: nsim, nyears. Non-negative real numbers

`Hist` Information from the historical spool-up period. Object of class `Hist`. Only contains slots `AtAge` and `TSdata` unless `extended=TRUE` in `runMSE`

`PPD` Posterior predictive data. List of `Data` objects at the end of the projection period (length `nMPs`)

`Misc` Miscellaneous output

### Author(s)

T. Carruthers and A. Hordyk

---

`MSE2Slick` *Convert an* `mse` *Object to a Slick Object*

---

### Description

Converts an mse object (or a list of mse objects) into a `Slick::Slick()` data object for interactive visualization in the Slick App.

### Usage

```
MSE2Slick(
  MSE,
  Title = NULL,
  Subtitle = "",
  Author = "",
  Email = "",
  Institution = "",
  Introduction = "",
  Date = NULL,
  PMs = NULL,
  BoxplotPMs = NULL,
  QuiltPMs = NULL,
  SpiderPMs = NULL,
  TradeoffPMs = NULL,
  TimeseriesCode = NULL,
  TimeseriesLabel = NULL,
  TimeseriesDescription = NULL,
  TimeseriesTarget = NULL,
  TimeseriesLimit = NULL,
  SeasonalBasis = c("RefSeason", "Mean"),
  KobeComplex = NULL,
  KobeYears = NULL,
  KobeLimit = c(0.4, 1),
  MPCode = NULL,
  MPLabel = NULL,
  MPDescription = NULL,
  Design = NULL,
  AutoPreset = TRUE,
  OMsPreset = NULL,
  MPsPreset = NULL,
  BoxplotPreset = NULL,
  QuiltPreset = NULL,
  SpiderPreset = NULL,
  TradeoffPreset = NULL
)
```

### Arguments

| | |
|---|---|
| MSE | An mse object or a named list of mse objects, where each element represents a different OM. All objects in the list must have identical MPs, number of simulations, number of time steps, and stock complexes. |
| Title | Character, or NULL (default). NULL uses MSE@OM@Name (or the first list element's, when MSE is a list). |
| Subtitle, Author, Email, Institution, Introduction | Character. Optional metadata, all default "". |
| Date | Date, or NULL (default) for Sys.Date(). |
| PMs | A list of performance-metric specs, shared by default across Boxplot/Quilt/Spider/Tradeoff. NULL (default) uses DefaultSlickPMs(). See Details. |
| BoxplotPMs, QuiltPMs, SpiderPMs, TradeoffPMs | Per-panel overrides for PMs, or NULL (default) to fall back to it. See Details. |
| TimeseriesCode | Character. The variables of the Timeseries panel, or NULL (default) for c('SB_SBMSY', 'F_FMSY', 'Landings'). See Details. |
| TimeseriesLabel, TimeseriesDescription | Character, one per TimeseriesCode, or NULL (default) for the codes and built-in descriptions. |
| TimeseriesTarget, TimeseriesLimit | Numeric vectors named by TimeseriesCode, or NULL (default). The target and limit lines of the Timeseries panel. The defaults are a target of 1 for SB_SBMSY and F_FMSY, and limits of 0.4 for SB_SBMSY and 1 for F_FMSY; entries given here replace them. |
| SeasonalBasis | Character. For seasonal OMs, the annual value of the Timeseries variables that are not summed over the seasons: 'RefSeason' (default) or 'Mean'. See Details. |
| KobeComplex | Character, or NULL (default: the first complex). Which stock/complex the Kobe plot shows, when MSE has more than one. |
| KobeYears | Numeric, or NULL (default: the years the MPs are active, PMYears()). The calendar years of the Kobe plot. |
| KobeLimit | Numeric, length 2. The limits of SB/SBMSY and F/FMSY in the Kobe plot. Default c(0.4, 1). |
| MPCode, MPLabel, MPDescription | Overrides for the per-MP display text, or NULL (default). See Details. |
| Design | The OM factorial design, or NULL (default) for a single Scenario factor built from the MSE list names. See Details. |
| AutoPreset | Logical. Auto-populate OMs@Preset from Design? Default TRUE. See Details. |
| OMsPreset, MPsPreset, BoxplotPreset, QuiltPreset, SpiderPreset, TradeoffPreset | Preset buttons for the App, or NULL (default) for none (except OMs, which still gets AutoPreset's buttons). See Details. |

### Details

A `Slick::Slick()` object is a standardized data structure containing MSE results, MP metadata, and performance indicators. It can be passed directly to `Slick::App()` for interactive exploration, or saved and uploaded to the online Slick App.

When `MSE` is a list, each element should represent a different Operating Model (OM) tested with the same set of MPs.

All mse objects in the list must have identical MPs, number of simulations, number of time steps, and stock complexes.

The resulting `Slick::Slick()` object includes:

- **MPs**: management procedure codes, labels, and display colours.
- **OMs**: one factor level per list element of `MSE` (see `Design`).
- **Kobe**: annual SB/SBMSY and F/FMSY in the years the MPs are active (`KobeYears`). Kobe is single-stock so for a multi-complex OM, `KobeComplex` selects which one to show (default: the first).
- **Timeseries**: annual time series for the historical and projection periods (see `TimeseriesCode`).
- **Boxplot/Quilt/Spider/Tradeoff**: driven by `PMs`, a flexible list of performance-metric specs (see `PMs` below). Each is evaluated against every complex in every OM.

`PMs` (and the per-panel `BoxplotPMs`/`QuiltPMs`/`SpiderPMs`/ `TradeoffPMs` overrides) is a list of performance-metric specifications. Each element is one of:

- a `PM_*` function, e.g. `PM_Status` for `PM_Status()`;
- a list `list(PM, Args = list(), Code = NULL, Label = NULL, Description = NULL)` (`fun` and `args` are accepted for `PM` and `Args`), e.g. `list(PM = PM_Yield, Args = list(Years = 2030:2039), Code = 'Yield_Short')`;
- a call, e.g. `quote(PM_Status(Years = 2030:2039))`, evaluated in the environment `MSE2Slick()` is called from.

`Code`, `Label`, and `Description` default to the `Name` and `Caption` of the pm object. The codes must be unique, so give a `Code` when the same PM function is used more than once (e.g. with different `Years`; see `PMYears()`). Each per-panel override falls back to `PMs` when `NULL`; a spec list is only evaluated once even when shared across panels.

A PM is a probability if its `Prob` is not all `NA` (e.g. `PM_Status()`, `PM_Safety()`, `PM_FFMSY()` with a `Ref`), and a statistic otherwise (e.g. `PM_Yield()`, `PM_MinStatus()`, `PM_AAVY()`). The `Boxplot` and `Quilt` show the per-simulation `Prob` of a probability and the per-simulation `Stat` of a statistic, and the `Tradeoff` and `Spider` the `Mean` over simulations. The `Quilt` colour scale of a probability spans 0 to 1. In the `Spider`, which needs values from 0 to 1, a statistic is divided by its largest absolute value among the MPs in each OM.

`TimeseriesCode` selects the variables of the `Timeseries` panel, one value per calendar year: `'SB_SBMSY'`, `'F_FMSY'` (see `SB_SBMSY()`, `F_FMSY()`), `'SB_SB0'` (spawning biomass relative to equilibrium unfished spawning biomass), `'SBiomass'`, `'Landings'`, `'Removals'`, and `'TAC'` (the TAC in effect, projection years only). Other entries must name functions that accept an mse object and return a tidy data frame with columns `Sim`, `Year`, `MP`, `Period`, and `Value`. In seasonal OMs, catches and F are summed over the seasons of each calendar year and the TAC is the mean over the seasons; other variables are taken in the reference season

of each year (`SeasonalBasis = 'RefSeason'`, the season of the MSY reference points; see `RefSeason()`) or are the mean over the seasons (`SeasonalBasis = 'Mean'`).

`MPCode/MPLabel/MPDescription` set the resulting `Slick` object's `MPs@Code/Label/Description`. Each is `NULL` (default `MPCode/MPLabel` fall back to `names(MSE@MPs)`, `MPDescription` is left unset).

`Design` gives each OM's level along one or more factorial-design factors (e.g. `M`, steepness), as a `data.frame` (one row per OM, one column per factor) or an equivalent named list of length-`nOM` vectors. The per-factor level catalog Slick needs (`OMs@Factors`) is derived automatically from the unique values in each column; `NULL` (default) uses a single `Scenario` factor built from the `MSE` list names (or `"OM1"`, `"OM2"`, ... if unnamed).

When `Design` is supplied and `MSE` has more than one element, `MSE` must be a *named* list.

`AutoPreset` (default `TRUE`) auto-populates `OMs@Preset` with one button per unique level of each `Design` column, unless `OMsPreset` overrides it entirely; ignored when `Design` is `NULL`. `MPsPreset/BoxplotPreset/ QuiltPreset/SpiderPreset/TradeoffPreset` are pass-throughs (a named list of MP or PI indices into that object's own `Code`; see `Slick::Preset()`).

## Value

A `Slick::Slick()` object ready for use with `Slick::App()`.

## See Also

`Slick::Slick()`, `Slick::App()`, mse, `DefaultSlickPMs()`, PM, `PMYears()`

## Examples

```
## Not run:
Slick <- MSE2Slick(MSE)
Slick::App(slick=Slick)

# Multiple OMs, custom PMs
Slick <- MSE2Slick(
  list(Base = MSE_Base, LowM = MSE_LowM, HighM = MSE_HighM),
  Design = list(M = c('Base', 'Low', 'High')),
  PMs = list(
    PM_Status, PM_Safety,
    list(fun = PM_Yield, args = list(Years = 2000:2009), Code = 'Yield_early'),
    list(fun = PM_Yield, args = list(Years = 2010:2019), Code = 'Yield_late')
  )
)
Slick::App(slick=Slick)

# The same PM over windows of years, probabilities and statistics
Slick <- MSE2Slick(
  MSE,
  PMs = list(
    list(PM = PM_Status, Code = 'PGK', Label = 'P(Kobe green)'),
    list(PM = PM_Status, Args = list(Years = PMYears(MSE, 'first', 10)),
         Code = 'PGK_Short', Label = 'P(Kobe green) (years 1-10)'),
    list(PM = PM_Red, Code = 'PRed', Label = 'P(Kobe red)'),
```

```
    list(PM = PM_MinStatus, Code = 'SB_SBMSY_Min', Label = 'Minimum SB/SBMSY'),
    list(PM = PM_FFMSY, Args = list(Ref = 1.4), Code = 'PLim_F',
         Label = 'P(F < 1.4 FMSY)'),
    list(PM = PM_Yield, Code = 'Yield', Label = 'Mean catch'),
    list(PM = PM_AAVY, Args = list(IncludeFirst = TRUE), Code = 'AAV_TAC',
         Label = 'Mean TAC change'),
    list(PM = PM_TACLimited, Code = 'P_TACLimited', Label = 'P(TAC change limited)')
  ),
  TimeseriesCode = c('SB_SBMSY', 'F_FMSY', 'SB_SB0', 'Removals', 'TAC'),
  TimeseriesLimit = c(SB_SBMSY = 0.4, F_FMSY = 1.4),
  KobeLimit = c(0.4, 1.4)
)

## End(Not run)
```

---

`MSEextra` *Load more data from MSEextra package*

---

## Description

Downloads the MSEextra package from GitHub

## Usage

```
MSEextra(silent = FALSE)
```

## Arguments

| | |
|---|---|
| `silent` | Logical. Should messages to printed? |

---

`MSY-accessors` *Access MSY Reference Point Slots*

---

## Description

Accessor functions for MSY-based biological reference points stored in `Hist@Reference@MSY`. Each function returns the corresponding array from a hist or mse object.

## Usage

```
MSYRefs(Hist)

FMSY(Hist)

BMSY(Hist)
```

```
SBMSY(Hist)

SPMSY(Hist)

SPRMSY(Hist)

MSYLandings(Hist)

MSYDiscards(Hist)
```

### Arguments

| | |
|---|---|
| `Hist` | A hist or mse object. |

### Details

The available accessor functions and the quantities they return are:

- `MSYRefs()`: the full refpointsMSY object containing all slots.
- `FMSY()`: apical fishing mortality at MSY, defined at the complex level (`Sim × Complex × Year`).
- `BMSY()`: total biomass at MSY.
- `SBMSY()`: spawning biomass at MSY.
- `SPMSY()`: spawning production at MSY.
- `SPRMSY()`: spawning potential ratio at MSY.
- `MSYLandings()`: landed catch at MSY.
- `MSYDiscards()`: dead discards at MSY.

### Value

A numeric array with dimensions `Sim × Stock × Year`, except `FMSY()` which returns `Sim × Complex × Year`. Returns `NULL` if the slot has not yet been populated (i.e. `CalcMSY()` has not been run).

### See Also

`CalcMSY()`, refpointsMSY

---

## `MSYCalcs` *Internal function to calculate MSY Reference Points*

---

### Description

Internal function to calculate MSY Reference Points

### Usage

```
MSYCalcs(
  logF,
  M_at_Age,
  Wt_at_Age,
  Mat_at_Age,
  Fec_at_Age,
  V_at_Age,
  Wt_at_Age_C,
  maxage,
  relRfun,
  SRRpars,
  R0x = 1,
  SRrelx = 3L,
  hx = 1,
  SSBpR = 0,
  opt = 1L,
  plusgroup = 1L,
  spawn_time_frac = 0
)
```

### Arguments

| | |
|---|---|
| `logF` | log fishing mortality |
| `M_at_Age` | Vector of M-at-age |
| `Wt_at_Age` | Vector of stock weight-at-age |
| `Mat_at_Age` | Vector of maturity-at-age |
| `Fec_at_Age` | Vector of mature weight-at-age |
| `V_at_Age` | Vector of selectivity-at-age |
| `Wt_at_Age_C` | Vector of fishery weight-at-age |
| `maxage` | Maximum age |
| `relRfun` | Optional. A function used to calculate reference points if `SRrelc =3` |
| `SRRpars` | Optional. A named list of arguments for `SRRfun` |
| `R0x` | R0 for this simulation. Set = 1 if SRrelx = 4 for per-recruit calculations |
| `SRrelx` | SRR type for this simulation. Use 4 for per-recruit calculations, i.e. constant recruitment. |
| `hx` | numeric. Steepness value for this simulation. Not used if SRrelx = 4. |
| `SSBpR` | numeric. Unfished spawners per recruit for this simulation. Not used if SRrelx = 4. |
| `opt` | Option. 1 = return -Yield, 2= return all MSY calcs |
| `plusgroup` | Integer. Default = 0 = no plus-group. Use 1 to include a plus-group |
| `spawn_time_frac` | Numeric. Fraction of the year when spawning occurs. Default = 0. |

### Value

See `opt`

---

`multiData` *Combine data among fleets*

---

### Description

Catches, CAA, CAL are summed. LFC and LFS are weighted averages. ML, Lc and Lbar are recalculated from summed CAL. All other observations are for fleet 1 (indicative)

### Usage

```
multiData(MSElist, StockPars, p, mm, nf)
```

### Arguments

| | |
|---|---|
| `MSElist` | A hierarchical list of data objects stock then fleet then MP |
| `StockPars` | A list of stock parameters |
| `p` | Integer the Stock number |
| `mm` | Integer the MP number |
| `nf` | The number of fleets |

### Author(s)

T. Carruthers

---

`multiDataS` *Combine data among stocks*

---

### Description

Catches, CAA, CAL are summed. Indices, LFC and LFS are weighted averages. ML, Lc and Lbar are recalculated from summed CAL. All other observations are for fleet 1 and weighted average across stocks

### Usage

```
multiDataS(MSElist, Real.Data.Map, np, mm, nf, realVB)
```

### Arguments

| | |
|---|---|
| `MSElist` | A hierarchical list of data objects stock then fleet then MP |
| `Real.Data.Map` | Matrix describing which data are mapped across stocks |
| `np` | The number of stocks |
| `mm` | Integer the MP number |
| `nf` | The number of fleets |
| `realVB` | A matrix of real vulnerable biomass `[nsim,np, year]` |

### Author(s)

T. Carruthers

---

## `multidebug` *A basic comparison of runMSE output (MSE) and multiMSE (MMSE)*

---

### Description

A basic comparison of runMSE output (MSE) and multiMSE (MMSE)

### Usage

```
multidebug(MSEsingle, MSEmulti, p = 1, f = 1, MPno = 1, maxsims = 4)
```

### Arguments

| | |
|---|---|
| `MSEsingle` | An object of class MSE arising from a run of runMSE(OM, ...) |
| `MSEmulti` | An object of class MMSE arising from a run of multiMSE(MOM, ...) |
| `p` | Integer. The stock number from the MSEmulti object (to be plotted) |
| `f` | Integer. The fleet number from the MSEmulti object (to be plotted) |
| `MPno` | Integer. The MP number from the MSEmulti and MSEsingle object (to be plotted) |
| `maxsims` | Integer. The maximum number of simulations to plot. |

### Author(s)

T.Carruthers

`MultiStockOM` *Multi-Stock Multi-Fleet Example Operating Model*

## Description

An example om object representing the most complex structural case: two biologically contrasting stocks each exploited by two fleets, with observation models varying by both stock and fleet. Intended to illustrate multi-stock, multi-fleet OM construction in `MSEtool`. Pairings of stocks, fleets, and observation models are for illustration only and should not be taken as ecologically meaningful. See also SingleStockOM, TwoFleetOM, and ComplexOM for simpler structural examples.

## Usage

```
MultiStockOM
```

## Format

An om object with the following slots populated:

- `Name`: `"Multi Stock - Multi Fleet"`.
- `nSim`: `8`. Number of stochastic simulations.
- `nYear`: `20`. Number of historical years.
- `pYear`: `30`. Number of projection years.
- `Stock`: a length-2 list of `[[AlbacoreExStock, ButterfishExStock]]`.
- `Fleet`: a length-2 list, each containing a length-2 list of `[[AsympExFleet, DomeExFleet]]`. Structure is `[[stock]][[fleet]]`. Both stocks are fished by the same two fleet types.
- `Obs`: a length-2 list with structure `[[stock]][[fleet]]`:
  - AlbacoreExStock: `[[AgeStructuredObs, CommercialFleetObs]]`
  - ButterfishExStock: `[[CatchAndSurveyObs, LengthStructuredObs]]`
- `Complexes`: `NULL`. Stocks are managed independently.
- `Imp`: a length-2 list with structure `[[stock]][[fleet]]`, each entry FullComplianceImp. Perfect TAC/effort implementation and full immediate size-regulation compliance, for both stocks and fleets.

## Details

**Stocks:**

Two biologically contrasting stocks are included:

- AlbacoreExStock: long-lived, slow-growing, low natural mortality.
- ButterfishExStock: short-lived, fast-growing, high natural mortality.

See AlbacoreExStock and ButterfishExStock for full parameter details.

**Fleets:**

Both stocks are fished by the same two fleet types:

- AsympExFleet: asymptotic length-based selectivity; effort stabilises early in the historical period.
- DomeExFleet: dome-shaped selectivity; effort increases gradually across the historical period.

See AsympExFleet and DomeExFleet for full parameter details.

**Observation Models:**

Observation models vary by stock and fleet, illustrating a range of monitoring intensities across the two stocks:

- AlbacoreExStock, Fleet 1 (AsympExFleet): AgeStructuredObs — landings, a spawning-biomass survey, and age-composition samples from landed and discarded fish.
- AlbacoreExStock, Fleet 2 (DomeExFleet): CommercialFleetObs — landings, discards, effort, and a commercial CPUE index; no survey or composition data.
- ButterfishExStock, Fleet 1 (AsympExFleet): CatchAndSurveyObs — reported landings and a biomass-proportional survey index only.
- ButterfishExStock, Fleet 2 (DomeExFleet): LengthStructuredObs — landings, a survey, and annual length-composition samples from landed and discarded fish.

**Implementation Error:**

FullComplianceImp models perfect implementation: `TAC@Mean = 1/ SD = 0` and `Effort@Mean = 1/SD = 0` mean realised TAC/effort always equal advice exactly, and `Size@Compliance = 1` means any size-regulation advice (a change to `Advice@Retention/Advice@Selectivity`) is adopted immediately by the whole fleet. See `Imp()` for details, and OverageImp, UnderageImp, PartialSizeComplianceImp for contrasting examples.

## See Also

SingleStockOM, TwoFleetOM, ComplexOM for simpler structural variants. AlbacoreExStock, ButterfishExStock, AsympExFleet, DomeExFleet, AgeStructuredObs, CommercialFleetObs, CatchAndSurveyObs, LengthStructuredObs, FullComplianceImp for the component objects. `OM()`, om, `runMSE()`, `PopulateOM()`, `ExampleMPs()`

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
MultiStockOM
nStock(MultiStockOM)
nFleet(MultiStockOM)

## Not run:
Hist <- runMSE(MultiStockOM, MPs = ExampleMPs())

## End(Not run)
```

`name-accessors` *Access Stock or Fleet Names*

## Description

Access Stock or Fleet Names

## Usage

```
StockNames(object)

FleetNames(object, IncSurvey = FALSE)
```

## Arguments

| | |
|---|---|
| `object` | An om, hist, stock, fleet, or mse object. |
| `IncSurvey` | Logical. Include names of survey (non-fishing) fleets? Default: FALSE |

## Details

These functions extract stock or fleet names from objects used in the MSE framework, including om, hist, stock, fleet, and mse objects

**StockNames**

Returns the names of stocks contained in the object.

**FleetNames**

Returns the names of fleets contained in the object. For objects containing multiple stocks, a list of character vectors is returned.

## Value

- `StockNames()`: a character vector of stock names
- `FleetNames()`: a character vector of fleet names (always taken from the first stock)

## Examples

```
StockNames(SingleStockOM)
FleetNames(SingleStockOM)
```

`NaturalMortality` *NaturalMortality Constructor and Accessors*

## Description

Construct a naturalmortality object defining the natural mortality schedule for a stock, or access and replace the `NaturalMortality` slot of a stock and its individual slots. A `NaturalMortality` object is required for all stock objects.

## Usage

```
NaturalMortality(
  Pars = list(),
  Model = NULL,
  Units = "year",
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  Random = NULL,
  Classes = NULL,
  Misc = list()
)

NaturalMortality(x) <- value
```

## Arguments

| | |
|---|---|
| `Pars` | `list`. Named list of natural mortality parameters. Element names must match the arguments of a built-in mortality model (see `NaturalMortalityModels()`). When `Pars` is non-empty and `Model` is `NULL`, the model is inferred automatically from the parameter names. When `Pars` is a stock object, `NaturalMortality()` acts as a pass-through accessor and returns `x@NaturalMortality`. See also Specifying Biological and Fleet Schedules for the full set of accepted input formats. Default `list()`. |
| `Model` | `character(1)` or `function`. Natural mortality model identifier. When `NULL` (default), the model is inferred from `Pars`. May be set to a character string naming a built-in model or to a custom R function — see Specifying Biological and Fleet Schedules. |
| `Units` | `character(1)`. Time unit in which mortality rates are expressed. Must be one of the strings returned by `ValidUnits()` (e.g., `"year"` for instantaneous annual mortality). For seasonal models, set `Units` to match the season length so that within-step rates are scaled correctly during `Populate()`. Default `"year"`. |
| `MeanAtAge` | `array` or `NULL`. Mean instantaneous natural mortality at age with named dimensions `Sim × Age × Year`. Supply directly when bypassing the model-based approach (`Pars = list()`). Only the years at which values *change* need to be included; `Extend()` fills the remainder. When `Pars` contains |

| | |
|---|---|
| | a matched model, any values here are **overwritten** during `Populate()`. A numeric vector of length `nAge` is also accepted and converted to a `1 × nAge × 1` array automatically. Default `NULL`. |
| `MeanAtLength` | `array` or `NULL`. Mean natural mortality at length with named dimensions `Sim × Class × Year`. Populated automatically during `Populate()` when an at-length mortality model is used; may also be supplied directly. When `MeanAtLength` is populated and `MeanAtAge` is not, `Populate()` converts it to `MeanAtAge` via the age-length key (`ALK`) from a length object. A numeric vector is also accepted and converted to a `1 × nClass × 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`. Default `NULL`. |
| `Random` | `array` or `NULL`. Reserved for future use. Intended to hold simulation- and year-specific multipliers that add stochastic variation around the average mortality schedule in `MeanAtAge`. Currently stored but not applied during `Populate()`. Default `NULL`. |
| `Classes` | `numeric` or `NULL`. Age or length class midpoints corresponding to the `MeanAt*` array in use. Default `NULL`. |
| `Misc` | `list`. Used internally. Default `list()`. |
| `x` | A naturalmortality object for slot accessors, or a stock object for `NaturalMortality<-`. |
| `value` | For `NaturalMortality<-`: a naturalmortality object. |

## Details

### Specifying Natural Mortality:

There are two ways to define the mortality schedule; see Specifying Biological and Fleet Schedules for full details on input formats and the rules that govern how `Pars`, `Model`, and `MeanAt*` arrays interact.

**Model-based** (recommended): supply `Pars` with named parameters matching a built-in model (see `NaturalMortalityModels()`). If `Model = NULL` and the parameter names uniquely match a model, the model is resolved automatically. `MeanAtAge` is then populated by `Populate()` when the stock is added to an `OM()`:

```
# Constant M - model inferred from Pars names
nm <- NaturalMortality(Pars = list(M = 0.2))

# Stochastic M across simulations
nm <- NaturalMortality(Pars = list(M = c(0.1, 0.3)))

# Time-varying M (increases in 2010; Extend() fills the rest)
M_arr <- array(c(0.2, 0.3), dim = c(1, 2),
               dimnames = list(Sim = 1, Year = c(1990, 2010)))
nm <- NaturalMortality(Pars = list(M = M_arr))

# Inter-annual random walk on M (log-normal, SD = 0.1)
nm <- NaturalMortality(Pars = list(M = c(0.1, 0.3), MSD = 0.1))
```

**Direct array**: supply `MeanAtAge` (or `MeanAtLength`) with `Pars = list()`. Values are preserved during `Populate()`:

```
ages  <- 0:20
# Age-varying M declining from 0.5 at age 0 to 0.1 at the plus group
nm_aa <- NaturalMortality(
  MeanAtAge = array(
    seq(0.5, 0.1, length.out = length(ages)),
    dim      = c(1, length(ages), 1),
    dimnames = list(Sim = 1, Age = ages, Year = 1990)
  )
)
```

### Units and Seasonal Models:

Rates in `MeanAtAge` are instantaneous mortality values ($M$) expressed over the period defined by `Units`. The default `Units = "year"` means annual $M$. For seasonal models, set `Ages@Units` to the season length (e.g., `"quarter"` for quarterly steps) and make sure the rate parameters (e.g., M and K) are in the same temporal units.

### At-Length Models:

Models whose class contains `"at-Length"` produce `MeanAtLength`, which is converted to `MeanAtAge` via the `ALK` — a populated `Length()` object must be passed to `Populate()`. The same conversion applies when `MeanAtLength` is supplied directly by the user.

### Pass-Through Access from a Stock:

When `Pars` is a stock object, `NaturalMortality()` returns the `NaturalMortality` slot directly:

```
NaturalMortality(my_stock)           # returns my_stock@NaturalMortality
NaturalMortality(my_stock) <- my_M   # replaces it
```

### Slot Accessors:

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Pars(nm)          <- list(M = 0.2)
Model(nm)         <- "Constant"
Units(nm)         <- "year"
MeanAtAge(nm)     <- my_array
MeanAtLength(nm)  <- my_length_array
Classes(nm)       <- 0:20
```

## Value

- `NaturalMortality()` returns a naturalmortality object. If `Pars` is a stock, returns `x@NaturalMortality`.
- `NaturalMortality<-` returns the stock `x` with the `NaturalMortality` slot replaced and the object re-validated.

## See Also

- naturalmortality for the class definition and slot-level documentation.
- `NaturalMortalityModels()` for available models and required parameter sets.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.
- `Length()` for the companion length schedule, required when using at-length mortality models.

Other naturalmortality: `naturalmortality-class`

## Examples

```
# See NaturalMortalityModels() for all available built-in natural mortality
# models and the parameter names required for each.
NaturalMortalityModels()

# ---- Model-based specification ----

## Constant M (scalar) - same value for every simulation and year
nm <- NaturalMortality(Pars = list(M = 0.2))

## Stochastic across simulations - M drawn from Uniform(lower, upper)
## once per simulation
nm <- NaturalMortality(Pars = list(M = c(0.1, 0.3)))

## Inter-annual random walk on M - log-normal with 10% CV
## MSD is removed from Pars after use
nm <- NaturalMortality(Pars = list(M   = c(0.1, 0.3),
                                   Msd = 0.1))

## Time-varying M with named change-point arrays
## Only the years where values change need to be supplied; Extend() fills
## all intermediate and future years by forward-filling automatically.
nSim <- 48

## M increases in 2010 (two change points: 1990 and 2010)
M_arr <- array(
  c(rep(0.2, nSim), rep(0.3, nSim)),
  dim      = c(nSim, 2),
  dimnames = list(Sim = seq_len(nSim), Year = c(1990, 2010))
)
nm <- NaturalMortality(Pars = list(M = M_arr))

# ---- Direct array specification ----

## Supply MeanAtAge directly (Pars left empty).
## A plain numeric vector of length nAge is accepted; treated as a single
## simulation, single year and replicated by Extend() as needed.
ages  <- 0:20
nm_aa <- NaturalMortality(
```

```
  MeanAtAge = seq(0.5, 0.1, length.out = length(ages))
)

## Named Sim x Age x Year array - age-varying M that shifts in 2010.
## Extend() replicates Sim = 1 to all nSim simulations automatically.
maa <- array(
  c(seq(0.5, 0.1, length.out = length(ages)),   # 1990 schedule
    seq(0.6, 0.15, length.out = length(ages))),  # 2010 schedule (M higher)
  dim      = c(1, length(ages), 2),
  dimnames = list(Sim = 1, Age = ages, Year = c(1990, 2010))
)
nm_arr <- NaturalMortality(MeanAtAge = maa)

## Supply MeanAtLength directly - converted to MeanAtAge via the ALK
## during Populate() when a Length object is provided.
lens  <- seq(5, 120, by = 5)
nm_al <- NaturalMortality(
  MeanAtLength = array(
    0.4 * exp(-0.02 * lens),          # M declining with length
    dim      = c(1, length(lens), 1),
    dimnames = list(Sim = 1, Length = lens, Year = 1990)
  )
)

# ---- Slot accessors ----

nm <- NaturalMortality(Pars = list(M = 0.2))

## Read slots
MeanAtAge(nm)     # NULL until Populate() is called
Units(nm)

## Replace slots
Units(nm)  <- "quarter"
Classes(nm) <- 0:20

# ---- Attaching to a Stock ----

stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
NaturalMortality(stk) <- NaturalMortality(Pars = list(M = 0.2))
NaturalMortality(stk)

# ---- Populate ----

## The final populated object (typically done internally).
## At-length models require a populated Length object passed to Populate().
pop_nm <- Populate(NaturalMortality(stk), Ages = Ages(MaxAge = 20))
pop_nm

MeanAtAge(pop_nm)
```

---

`naturalmortality-class`

*The* `naturalmortality` *S4 Class*

---

## Description

Defines the natural mortality schedule for a stock object. Mortality may be expressed as a function of age or length, and may vary across simulations and years. Objects are typically created via `NaturalMortality()`, which documents all parameters, validates inputs, and infers the mortality model automatically when possible.

## Details

Direct construction via `methods::new()` is not recommended; use `NaturalMortality()` instead, which handles model inference and object validation.

Rates in `MeanAtAge` are instantaneous mortality values ($M$) expressed over the period defined by `Units`. For seasonal models, set `Units` to match the season length so that within-step rates are scaled correctly during `Populate()`.

## Slots

`Pars list`. Named list of natural mortality parameters whose names correspond to those expected by `Model`. See `NaturalMortalityModels()` for parameter sets and Specifying Biological and Fleet Schedules for accepted input formats.

`Model function` or `character(1)`. Natural mortality model identifier, matched to one of `NaturalMortalityModels()`. Set automatically from the parameter names when `Pars` is supplied without an explicit model.

`Units character(1)`. Time unit in which mortality rates are expressed (e.g., `"year"` for instantaneous annual mortality). Must be accepted by `ValidUnits()` and match those used in `Ages()`.

`MeanAtAge array`. Mean instantaneous natural mortality at age with named dimensions `Sim`, `Age`, and `Year`. Populated automatically by `Populate()` when `Pars` and `Model` are set; may also be supplied directly when `Pars` is empty. When `Pars` contains a matched model, any existing values are overwritten. See Specifying Biological and Fleet Schedules.

`MeanAtLength array`. Mean natural mortality at length with named dimensions `Sim`, `Length`, and `Year`. Populated automatically when an at-length mortality model is used, or may be supplied directly. Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). Converted to `MeanAtAge` via the `ALK` during `Populate()` when `MeanAtAge` is not already populated. See Specifying Biological and Fleet Schedules.

`Random array`. Reserved for future use. Intended to hold simulation- and year-specific multipliers that add stochastic variation around the average mortality schedule in `MeanAtAge`. Currently stored but not applied during `Populate()`.

`Classes numeric`. Age classes (in years) or lower bounds of length bins, corresponding to the `MeanAt*` array in use. For length bins, bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength` are evaluated at bin midpoints, not at these lower bounds.

`Misc list`. Used internally.

### See Also

- `NaturalMortality()` for the constructor and accessor functions.
- `NaturalMortalityModels()` for available models and their required parameters.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `Length()` for the companion length schedule, required when using at-length mortality models.
- Specifying Biological and Fleet Schedules for the full description of how `Pars`, `Model`, and `MeanAt*` arrays interact.

Other naturalmortality: `NaturalMortality()`

---

## `NaturalMortality-Models`

*Natural Mortality Models*

---

### Description

Natural mortality models for age, length, and weight.

### Usage

```
MortalityAtAge(Ages, M)

LorenzenMortalityLength(Length, Mref, Lref, c = -0.288)

LorenzenMortalityWeight(Weight, Mref, Wref, c = -0.288)

NaturalMortalityModels(full = TRUE, print = TRUE)
```

### Arguments

| | |
|---|---|
| `Ages` | numeric vector of age classes |
| `M` | natural mortality rate. Recycled if not `length(Ages)` |
| `Length` | numeric vector of length classes |
| `Mref` | reference mortality |
| `Lref` | reference length for Lorenzen M-at-length |
| `c` | exponent for Lorenzen models (default -0.288) |

| | |
|---|---|
| `Weight` | numeric vector of weight classes |
| `Wref` | reference weight for Lorenzen M-at-weight |
| `full` | logical; provide a complete table of models (TRUE) or just model names (FALSE) |
| `print` | logical; print results (TRUE) or return data frame invisibly (FALSE) |

## Details

at-length and at-weight schedules are converted internally to at-age using the age-length age-weight key respectively.

- **M-at-age**:

$$M(a) = M$$

- **Lorenzen M-at-length**:

$$M(L) = M_{ref} \cdot \left( \frac{L}{L_{ref}} \right)^{-0.288}$$

- **Lorenzen M-at-weight**:

$$M(W) = M_{ref} \cdot \left( \frac{W}{W_{ref}} \right)^{-0.288}$$

`NaturalMortalityModels()` prints a list of available models.

## Value

Numeric vector of natural mortality at each age, length, or weight. `NaturalMortalityModels()` invisibly returns a data frame describing available models.

## See Also

`Ages()`, `Length()`, `Weight()`, `Stock()`

## Examples

```
Ages <- 0:15
Length <- seq(10, 100, 10)
Weight <- seq(1, 50, 5)

M_age <- MortalityAtAge(Ages, M = 0.2)

M_length_lor <- LorenzenMortalityLength(Length, Mref = 0.3, Lref = 50)

M_weight_lor <- LorenzenMortalityWeight(Weight, Mref = 0.3, Wref = 20)

# Plot M-at-age
plot(Ages, M_age, type = "l", lwd = 2, col = "blue",
     xlab = "Age", ylab = "M", main = "Constant M-at-Age",
     ylim=c(0,0.25))
```

```
# Plot M-at-length
plot(Length, M_length_lor, type = "l", lwd = 2, col = "green",
     xlab = "Length", ylab = "M", main = "Lorenzen M-at-Length",
     ylim=c(0, 0.5))

# Plot M-at-weight
plot(Weight, M_weight_lor, type = "l", lwd = 2, col = "purple",
     xlab = "Weight", ylab = "M", main = "Lorenzen M-at-Weight",
     ylim=c(0,0.8))
```

---

`NewPM` *Create a Custom Performance Metric*

---

## Description

Builds a pm object from a data frame of per-simulation, per-year values, so any metric can be used as a compatible performance metric (PM), e.g. as the objective or a constraint in `TuneMP()`.

## Usage

```
NewPM(df, Name, Caption = Name, Ref = NULL, Op = NULL, Years = NULL, OM = NULL)
```

## Arguments

| | |
|---|---|
| `df` | A data frame with columns `Sim`, `Stock`, `Year`, `MP`, and `Value` (and optionally `Period`; only `'Projection'` rows are used). |
| `Name` | Character. Short name of the metric. |
| `Caption` | Character. Description of the metric. Default `Name`. |
| `Ref` | `NULL` (default) or the reference value compared with `Value` by `Op`. |
| `Op` | `NULL` (default) or a comparison function, e.g. `` `>` ``. When given, `Prob` is, for each simulation, the proportion of years in which `Op(Value, Ref)` is `TRUE`, and `Mean` is `Prob` averaged over simulations; otherwise `Prob` is `NA` and `Mean` is `Stat` (the per-simulation mean `Value`) averaged over simulations. |
| `Years` | `NULL` (default, all projection years) or the years to include. |
| `OM` | `NULL`, or the om object, to exclude interim years before `OM@MPStartYear`. |

## Details

A custom PM function takes an mse object, calculates the metric (e.g. from `Landings()` or `SB_SBMSY()` with `df = TRUE`), and returns `NewPM(...)`.

## Value

A pm object.

### See Also

PM, TuneMP()

### Examples

```
## Not run:
PM_MinSB <- function(object, Years = NULL) {
  df <- SB_SBMSY(object, df = TRUE, Reduce = FALSE)
  df <- df[df$Period == 'Projection', ]
  df <- dplyr::summarise(dplyr::group_by(df, Sim, Stock, MP),
                         Value = min(Value), Year = max(Year), .groups = 'drop')
  NewPM(df, Name = 'MinSB', Caption = 'P(min SB/SBMSY > 0.5)', Ref = 0.5, Op = `>`)
}
class(PM_MinSB) <- 'pm'

## End(Not run)
```

---

## NIL *Item in list: get the list values from a list of lists*

---

### Description

Create of vector of values that correspond with a slot in a list of objects

### Usage

```
NIL(listy, namey, lev1 = T)
```

### Arguments

| | |
|---|---|
| listy | A list of objects |
| namey | A character vector representing the list item's name |
| lev1 | Logical, should NIL default to the first level of the list? |

### Author(s)

T. Carruthers

## NOAA_plot — *National Oceanographic and Atmospheric Administration default plot 1*

### Description

A preliminary plot for returning trade-offs plots and performance table for total yield, variability in yield, probability of overfishing and likelihood of biomass dropping below 50 per cent BMSY

### Usage

```
NOAA_plot(MSEobj, nam = NA, type = NA, panel = T)
```

### Arguments

| | |
|---|---|
| MSEobj | An object of class MSE |
| nam | Title of plot |
| type | Plots full range of data if NA. Plots a subset that meet thresholds if not NA. |
| panel | Should a two panel plot be made or should plots be made in sequence. |

### Value

A table of performance metrics.

### Author(s)

T. Carruthers

## Number — *Extract Number-at-Age time series*

### Description

Returns the number of individuals (abundance) from a hist or mse object, either as a list of length nStock() or a data.frame.

### Usage

```
Number(object, df = TRUE, byAge = FALSE, byArea = FALSE)
```

### Arguments

| | |
|---|---|
| `object` | A hist or mse object. |
| `df` | Logical. If `FALSE` (default) the raw array slot is returned. If `TRUE` (default) a tidy `data.frame` is returned. |
| `byAge` | Logical. If `TRUE` the data frame retains the `Age` dimension rather than summing over ages. Ignored when `df = FALSE`. |
| `byArea` | Logical. If `TRUE` the data frame retains the `Area` dimension rather than summing over areas. Ignored when `df = FALSE`. |

### Value

- `df = FALSE` — the raw `Number` array slot (list of arrays, one per stock).
- `df = TRUE` — a `data.frame` (subclass `"number.df"`) with columns `Sim`, `Stock`, `Year`, `Period`, `MP` (MSE only), and optionally `Age` / `Area`, plus `Value`, `Variable`, and `Units`.

### See Also

`Biomass()`, `SBiomass()`, `SProduction()`

### Examples

```
Hist <- Simulate(SingleStockOM)
Number(Hist)
Number(Hist, byAge = TRUE)
Number(Hist, byAge = TRUE, byArea = TRUE)
```

---

## `Obs` *Obs Constructor and Accessor*

---

### Description

Construct an obs object defining the observation error structure for each data type in the operating model, or extract the `Obs` slot from an enclosing object.

### Usage

```
Obs(
  Name = NULL,
  LifeHistory = NULL,
  Exploitation = NULL,
  Effort = NULL,
  Landings = NULL,
  Discards = NULL,
  CPUE = NULL,
  Survey = NULL,
  LandingsAtAge = NULL,
  DiscardsAtAge = NULL,
```

```
  LandingsAtSize = NULL,
  DiscardsAtSize = NULL,
  Misc = list()
)

Obs(x) <- value
```

## Arguments

| | |
|---|---|
| `Name` | `character(1)` or an S4 object. Unique identifier for this observation model. Default `NULL`.<br>If `Name` is an om object, `Obs()` returns `Name@Obs` (the top-level observation model list) rather than constructing a new object. If `Name` is a hist or mse object, `Obs()` returns the `Obs` slot of the embedded OM (`Name@OM@Obs`). See *Pass-Through Access* in Details. |
| `LifeHistory` | A lifehistoryobs object, or `NULL` (default). When `NULL`, an empty lifehistoryobs is created via `LifeHistoryObs()`. Currently a placeholder; see `LifeHistoryObs()`. |
| `Exploitation` | An exploitationobs object, or `NULL` (default). When `NULL`, an empty exploitationobs is created via `ExploitationObs()`. Currently a placeholder; see `ExploitationObs()`. |
| `Effort` | An effortobs object, or `NULL` (default). When `NULL`, an empty effortobs is created via `EffortObs()`. |
| `Landings` | A catchobs object, or `NULL` (default). When `NULL`, an empty catchobs is created via `CatchObs()`. Defines observation error for landed catch. |
| `Discards` | A catchobs object, or `NULL` (default). When `NULL`, an empty catchobs is created via `CatchObs()`. Defines observation error for discarded catch. |
| `CPUE` | An indicesobs object, or `NULL` (default). When `NULL`, an empty indicesobs is created via `IndicesObs()`. Defines observation error for CPUE indices. |
| `Survey` | An indicesobs object, or `NULL` (default). When `NULL`, an empty indicesobs is created via `IndicesObs()`. Defines observation error for fishery-independent survey indices. |
| `LandingsAtAge` | A compobs object, or `NULL` (default). When `NULL`, an empty compobs is created via `CompObs()`. Defines observation error for landed catch-at-age composition. |
| `DiscardsAtAge` | A compobs object, or `NULL` (default). When `NULL`, an empty compobs is created via `CompObs()`. Defines observation error for discarded catch-at-age composition. |
| `LandingsAtSize` | A compobs object, or `NULL` (default). When `NULL`, an empty compobs is created via `CompObs()`. Defines observation error for landed catch-at-length composition. |
| `DiscardsAtSize` | A compobs object, or `NULL` (default). When `NULL`, an empty compobs is created via `CompObs()`. Defines observation error for discarded catch-at-length composition. |

| | |
|---|---|
| `Misc` | `list`. Miscellaneous additional objects. Default `list()`. |
| `x` | An om object. |
| `value` | A two-level named list of obs objects, or a single obs object, to assign to the `Obs` slot. |

## Details

### Sub-Object Initialisation:

All sub-object slots are initialised to empty objects of the appropriate class when not supplied. Empty sub-objects cause the model to skip observation error for that data type (e.g., an empty `Landings` slot means catch is reported without observation error). To activate observation error for a data type, supply a sub-object with at least `CV` specified.

### Pass-Through Access:

When `Name` is an om, hist, or mse object, `Obs()` extracts the `Obs` slot of the embedded OM rather than constructing a new object:

- `Obs(om)` → `om@Obs`
- `Obs(hist)` → `hist@OM@Obs`
- `Obs(mse)` → `mse@OM@Obs`

The result in all cases is the two-level named list `[[stock_complex]][[fleet_name]]` of obs objects stored in the OM.

### Attaching to an OM:

A single obs object can be assigned to a specific stock complex and fleet slot with:

```
Obs(om) <- MyObs            # replaces the full Obs list
```

Individual sub-objects are more commonly set directly on the `obs` object before it is inserted into the OM.

## Value

- `Obs()` returns an obs object. When `Name` is an om, hist, or mse object, returns the `Obs` slot of the embedded OM (a two-level named list of obs objects).
- `Obs<-` returns `x` with the `Obs` slot replaced by `value`.

## See Also

- obs for the class definition and slot-level documentation.
- `CatchObs()`, `EffortObs()`, `IndicesObs()`, `CompObs()` for sub-object constructors.
- `LifeHistoryObs()`, `ExploitationObs()` for placeholder sub-objects.
- data and `Data()` for the complementary observed-values object.
- `OM()` for the operating model constructor.
- `ConvertObs()` for converting legacy Obs-legacy objects.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`, `obs-class`

### Examples

```
# Empty obs object
obs <- Obs()
obs

# Obs with catch observation error specified
obs <- Obs(
  Name     = "MyObs",
  Landings = CatchObs(CV = 0.2, Bias = 1.0),
  CPUE     = IndicesObs(CV = 0.3)
)
```

---

## obs-class *The* obs *S4 Class*

---

### Description

Defines the observation error structure applied to each data type in the operating model. Each slot contains a sub-object specifying bias, CV, and sampling structure for one category of observed data. Objects are typically created via the `Obs()` constructor, which documents all parameters in detail.

### Details

An `obs` object defines how each data type is observed, i.e., the error structure (CV, bias, selectivity). Observed values are held in data objects stored in `OM@Data`, `Hist@Data`, and `MSE@PPD`.

`OM@Obs` is a two-level named list indexed first by stock complex, then by fleet or index name:

```
OM@Obs[[stock_complex]][[fleet_name]]  →  obs-class object
```

Stock complexes aggregate stocks whose data are reported together (e.g., combined landings across species). This indexing mirrors `OM@Data`.

The slots `Effort`, `Landings`, `Discards`, `CPUE`, `Survey`, `LandingsAtAge`, `DiscardsAtAge`, `LandingsAtSize`, and `DiscardsAtSize` are populated with empty sub-objects by `Obs()` when not supplied. The `LifeHistory` and `Exploitation` slots are reserved for future use and are not currently populated during model runs; see `LifeHistoryObs()` and `ExploitationObs()`.

Direct construction via `methods::new()` is not recommended; use `Obs()` instead, which initialises all sub-objects automatically.

### Slots

`Name` `character` or `NULL`. Unique identifier for this observation model. See `Obs()`.

`LifeHistory` A lifehistoryobs object. Observation error on life-history parameters (growth, maturity, natural mortality, etc.). See `LifeHistoryObs()`.

`Exploitation` An exploitationobs object. Observation error on exploitation processes (selectivity, retention, discard mortality). See `ExploitationObs()`.

`Effort` An effortobs object. Observation error on fishing effort. See `EffortObs()`.

`Landings` A catchobs object. Observation error on landed catch. See `CatchObs()`.

`Discards` A catchobs object. Observation error on discarded catch. See `CatchObs()`.

`CPUE` An indicesobs object. Observation error on catch-per-unit-effort indices. See `IndicesObs()`.

`Survey` An indicesobs object. Observation error on fishery-independent survey indices. See `IndicesObs()`.

`LandingsAtAge` A compobs object. Observation error on landed catch-at-age composition. See `CompObs()`.

`DiscardsAtAge` A compobs object. Observation error on discarded catch-at-age composition. See `CompObs()`.

`LandingsAtSize` A compobs object. Observation error on landed catch-at-length composition. See `CompObs()`.

`DiscardsAtSize` A compobs object. Observation error on discarded catch-at-length composition. See `CompObs()`.

`Misc list`. Miscellaneous additional objects.

## See Also

- `Obs()` for the constructor and accessor.
- `CatchObs()`, `EffortObs()`, `IndicesObs()`, `CompObs()` for sub-object constructors.
- `LifeHistoryObs()`, `ExploitationObs()` for placeholder sub-objects.
- data and `Data()` for the complementary observed-values object.
- `OM()` for the operating model constructor.
- `ConvertObs()` for converting legacy Obs-legacy objects.

Other obs: `CatchObs()`, `CompObs()`, `EffortObs()`, `ExploitationObs()`, `IndicesObs()`, `LifeHistoryObs()`, `Obs()`, `catchobs-class`, `compobs-class`, `effortobs-class`, `exploitationobs-class`, `indicesobs-class`, `lifehistoryobs-class`

---

`Obs-legacy-class` *Class* `'Obs'`

---

## Description

An operating model component that controls the observation model

**Slots**

`Name` The name of the observation model object. Single value. Character string.

`Name` The name of the Observation error object. Single value. Character string.

`Cobs` Observation error around the total catch. Observation error in the total catch is expressed as a coefficient of variation (CV). Cobs requires upper and lower bounds of a uniform distribution, and for each simulation a CV is sampled from this distribution. Each CV is used to specify a log-normal error distribution with a mean of 1 and a standard deviation equal to the sampled CV. The yearly observation error values for the catch data are then drawn from this distribution. For each time step the simulation model records the true catch, but the observed catch is generated by applying this yearly error term (plus any bias, if specified) to the true catch.

`Cbiascv` Log-normally distributed coefficient of variation controlling the sampling bias in observed catch for each simulation. Bias occurs when catches are systematically skewed away from the true catch level (for example, due to underreporting of catch or undetected illegal catches). Cbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years.

`CAA_nsamp` Number of catch-at-age observations collected per time step. For each time step a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Positive integers.

`CAA_ESS` Effective sample size of catch-at-age observations collected per time step. For each time step a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. CAA_ESS should not exceed CAA_nsamp. If greater than 1, then this is the multinomial distribution sample size. If less than 1, this is the coefficient of variation for the logistic normal distribution (see help doucmentation for simCAA for details).

`CAL_nsamp` Number of catch-at-length observations collected per time step. For each time step a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Positive integers.

`CAL_ESS` Effective sample size. For each time step a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. CAL_ESS should not exceed CAL_nsamp. Positive integers.

`Iobs` Observation error in the relative abundance index expressed as a coefficient of variation (CV). Iobs requires upper and lower bounds of a uniform distribution, and for each simulation a CV is sampled from this distribution. Each CV is used to specify a log-normal error distribution with a mean of 1 and a standard deviation equal to the sampled CV. The yearly observation error values for the index of abundance data are then drawn from this distribution. For each time step the simulation model records the true change in abundance, but the observed index is generated by applying this yearly error term (plus any bias, if specified) to the true relative change in abundance. Positive real numbers.

`Btobs` Observation error in the absolute abundance expressed as a coefficient of variation (CV). Btobs requires upper and lower bounds of a uniform distribution, and for each simulation a CV is sampled from this distribution. Each CV is used to specify a

log-normal error distribution with a mean of 1 and a standard deviation equal to the sampled CV. The yearly observation error values for the absolute abundance data are then drawn from this distribution. For each time step the simulation model records the true abundance, but the observed abundance is generated by applying this yearly error term (plus any bias, if specified) to the true abundance. Positive real numbers.

`Btbiascv` Log-normally distributed coefficient (CV) controlling error in observations of the current stock biomass. Bias occurs when the observed index of abundance is is systematically higher or lower than the true relative abundance. Btbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`beta` A parameter controlling hyperstability/hyperdepletion in the measurement of abundance. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Values below 1 lead to hyperstability (the observed index decreases more slowly than the true abundance) and values above 1 lead to hyperdepletion (the observed index decreases more rapidly than true abundance). Positive real numbers.

`LenMbiascv` Log-normal coefficient of variation for sampling bias in observed length at 50 percent maturity. LenMbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Mbiascv` Log-normal coefficient of variation for sampling bias in observed natural mortality rate. Mbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Kbiascv` Log-normal coefficient of variation for sampling bias in observed growth parameter K. Kbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`t0biascv` Log-normal coefficient of variation for sampling bias in observed t0. t0biascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Linfbiascv` Log-normal coefficient of variation for sampling bias in observed maximum length. Linfbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`LFCbiascv` Log-normal coefficient of variation for sampling bias in observed length at first capture. LFCbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For

each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`LFSbiascv` Log-normal coefficient of variation for sampling bias in length-at-full selection. LFSbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`FMSY_Mbiascv` Log-normal coefficient of variation for sampling bias in estimates of the ratio of the fishing mortality rate that gives the maximum sustainable yield relative to the assumed instantaneous natural mortality rate. FMSY/M. FMSY_Mbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`BMSY_B0biascv` Log-normal coefficient of variation for sampling bias in estimates of the BMSY relative to unfished biomass (BMSY/B0). BMSY_B0biascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Irefbiascv` Log-normal coefficient of variation for sampling bias in the observed relative index of abundance (Iref). Irefbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Brefbiascv` Log-normal coefficient of variation for sampling bias in the observed reference biomass (Bref). Brefbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Crefbiascv` Log-normal coefficient of variation for sampling bias in the observed reference catch (Cref). Crefbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Dbiascv` Log-normal coefficient of variation for sampling bias in the observed depletion level. Dbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Dobs` Log-normal coefficient of variation controlling error in observations of stock depletion among years. Observation error in the depletion expressed as a coefficient of variation (CV). Dobs requires the upper and lower bounds of a uniform distribution, and for each simulation a CV is sampled from this distribution. Each CV is used to specify a log-normal error distribution with a mean of 1 and a standard deviation equal to the sampled CV. The yearly observation error values for the depletion data are then drawn from this distribution. For each time step the simulation model records the

true depletion, but the observed depletion is generated by applying this yearly error term (plus any bias, if specified) to the true depletion.

`hbiascv` Log-normal coefficient of variation for sampling persistent bias in steepness. hbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Recbiascv` Log-normal coefficient of variation for sampling persistent bias in recent recruitment strength. Recbiascv requires the upper and lower bounds of a uniform distribution, and for each simulation a CV is sampled from this distribution. Each CV is used to specify a log-normal error distribution with a mean of 1 and a standard deviation equal to the sampled CV. The yearly bias values for the depletion data are then drawn from this distribution. Positive real numbers.

`sigmaRbiascv` Log-normal coefficient of variation for sampling persistent bias in recruitment variability. sigmaRbiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years. Positive real numbers.

`Eobs` Observation error around the total effort. Observation error in the total effort is expressed as a coefficient of variation (CV). Eobs requires upper and lower bounds of a uniform distribution, and for each simulation a CV is sampled from this distribution. Each CV is used to specify a log-normal error distribution with a mean of 1 and a standard deviation equal to the sampled CV. The yearly observation error values for the effort data are then drawn from this distribution. For each time step the simulation model records the true effort, but the observed effort is generated by applying this yearly error term (plus any bias, if specified) to the true effort.

`Ebiascv` Log-normally distributed coefficient of variation controlling the sampling bias in observed effort for each simulation. Bias occurs when effort is systematically skewed away from the true effort level. Ebiascv is a single value specifying the standard deviation of a log-normal distribution with a mean of 1 and a standard deviation equal to the sampled CV. For each simulation a bias value is drawn from this distribution, and that bias is applied across all years.

## Objects from the Class

Objects can be created by calls of the form `new('Obs')`

## Note

Its questionable whether the hyperstability/hyperdepletion should be categorised as an observation model characteristic as it is most often driven by fleet dynamics (and therefore should be in the fleet object). Oh well its here and you might want to make it hyperstable beta $< 1$ or hyperdeplete beta $> 1$, only.

## Author(s)

T. Carruthers and A. Hordyk

### Examples

```
showClass('Obs')
```

---

`OM` *OM Constructor and Accessor*

---

### Description

Construct an om object defining the complete specification of an operating model (OM) for use in Management Strategy Evaluation (MSE), or extract the `OM` slot from an enclosing object.

### Usage

```
OM(
  Name = "A new OM object",
  Agency = "",
  Author = "",
  Email = "",
  Region = "",
  Latitude = NULL,
  Longitude = NULL,
  Sponsor = "",
  nSim = 48,
  nYear = 20,
  pYear = 30,
  CurrentYear = as.numeric(format(Sys.Date(), "%Y")),
  Seasons = 1,
  RefSeason = NULL,
  RefEffortYears = NULL,
  Stock = NULL,
  Fleet = NULL,
  Obs = NULL,
  Imp = NULL,
  Data = NULL,
  DataLag = 0,
  CatchFrac = NULL,
  FleetAllocation = NULL,
  HistoricalWeight = NULL,
  SeasonalAllocation = NULL,
  EffortAllocation = NULL,
  Complexes = NULL,
  Herm = NULL,
  Relations = NULL,
  StockTargeting = NULL,
```

```
  Interval = 1,
  MPStartYear = NULL,
  InterimAdvice = NULL,
  nReps = 1,
  pStar = 0.5,
  maxF = 3,
  Seed = 101,
  Control = NULL,
  Misc = list(),
  Source = NULL
)
```

## Arguments

| | |
|---|---|
| `Name` | Character. Name of the operating model. Default `"A new OM object"`. If `Name` is an S4 object with an `OM` slot (e.g. a hist or mse object), `OM()` returns that slot directly rather than constructing a new object. See *Pass-Through Access* in Details. |
| `Agency` | Character. Name of the agency responsible for management. Supports Markdown. Default `""`. |
| `Author` | Character vector. Name(s) of the author(s) of the operating model. Default `""`. |
| `Email` | Character vector. Email address(es) corresponding to `Author`. Supports Markdown. Default `""`. |
| `Region` | Character. Name of the geographic region of the fishery. Default `""`. |
| `Latitude` | Numeric. Latitude (decimal degrees) of the centre of `Region`. Default `NULL`. |
| `Longitude` | Numeric. Longitude (decimal degrees) of the centre of `Region`. Default `NULL`. |
| `Sponsor` | Character. Organisation sponsoring development of the operating model. Supports Markdown. Default `""`. |
| `nSim` | Positive integer. Number of stochastic simulations. Default `48`. |
| `nYear` | Numeric. Number of historical years. Default `20`. |
| `pYear` | Numeric. Number of projection years. Default `30`. |
| `CurrentYear` | Integer. Final historical calendar year of the operating model. Default is the current system year. |
| `Seasons` | Integer. Number of seasons per year. Default `1`. |
| `RefSeason` | Integer vector or `NULL`. Only used for `Seasons > 1`. Reference season(s) for reporting equilibrium/MSY reference points. Default `NULL`. See `RefSeason()`. |
| `RefEffortYears` | Numeric vector or `NULL`. Only used for `Seasons > 1`. Historical year(s) fixing the seasonal shape of fishing mortality for reference point calculation. Default `NULL`. See `RefSeason()`. |
| `Stock` | A stock object or named list of stock objects. Default `NULL`. See `Stock()` for construction and pass-through access. |

`Fleet` A hierarchical named list of fleet objects indexed by stock then fleet. Each stock must have the same number of fleets. Default `NULL`. See `Fleet()` for construction and pass-through access.

`Obs` A hierarchical named list of obs objects indexed by stock and fleet. Default `NULL`. See `Obs()` for construction and pass-through access.

`Imp` A hierarchical named list of imp objects indexed by stock and fleet. Default `NULL`. See `Imp()` for construction and pass-through access.

`Data` A data object or list of data objects associated with the operating model. Default `NULL`. See `Data()` for construction and pass-through access.

`DataLag` Integer. Number of years that data are lagged relative to management implementation. Default `0`. See `DataLag()`.

`CatchFrac` List. Named list of length 0 or `nStock(OM)`, giving the historical fleet split of catch used as a catchability calibration target. Default `NULL`. See `FleetAllocation()`.

`FleetAllocation` List. Named list of length 0 or the number of stock complexes, controlling how the TAC is split among fleets during projection. Default `NULL`. See `FleetAllocation()`.

`HistoricalWeight` List. Named list of length 0 or the number of stock complexes, giving the weight placed on the historical seasonal pattern (vs. the population abundance pattern) when `SeasonalAllocation` is derived. Default `NULL`. See `SeasonalAllocation()`.

`SeasonalAllocation` List. Named list of length 0 or the number of stock complexes, controlling how a periodically-set TAC/Effort is split across the seasons of the interval it covers. Default `NULL`. See `SeasonalAllocation()`.

`EffortAllocation` List. Named list of length 0 or the number of stock complexes, splitting a scalar absolute Effort recommendation across fleets. Default `NULL`. See `EffortAllocation()`.

`Complexes` List. Defines stock complexes for data aggregation and management. Default `NULL`.

`Herm` A list of stocktransition objects (built via `Herm()`), one per `From`/`To` stock pair, defining age-dependent reclassification of individuals between stocks (e.g. sequential hermaphroditism). Default `NULL`.

`Relations` List. Biological or ecological relationships among stocks (e.g. predator-prey). Default `NULL`. Currently not used.

`StockTargeting` A stocktargeting object. Defines fleet-level stock targeting weights and deviations. Populated internally. See `StockTargeting()`.

`Interval` Numeric scalar or named numeric vector. Management update interval in years. Default `1`. A named vector gives a per-MP interval, e.g. `c(MP1 = 1, MP2 = 4)`. See `Interval()`.

`MPStartYear` Numeric or `NULL`. First calendar year in which MPs are applied. Default `NULL`. See `MPStartYear()`.

| | |
|---|---|
| InterimAdvice | A data.frame or NULL. Fixed or stochastic TAC/Effort values for years before MPStartYear. Default NULL. See InterimAdvice() for the required columns and how it is applied. |
| nReps | Positive integer. Number of stochastic replicates used for generating management advice. Default 1. Not currently used. |
| pStar | Numeric in $[0, 1]$. Percentile applied to stochastic management advice. Default 0.5. Not currently used. |
| maxF | Numeric. Maximum allowable instantaneous fishing mortality. Applies to nominal fishing mortality for fish that interact with the fishing gear; actual effective F may be lower if some fish are discarded and survive. Default 3. |
| Seed | Integer. Random number generator seed for reproducibility. Default 101. |
| Control | Named list of optional operating model control settings. Default list(). See OMControl for the available settings. |
| Misc | List. Miscellaneous objects or developer-use components. Default list(). |
| Source | Character. References to data sources or documentation. Supports Markdown. Default NULL. |

## Details

### Pass-Through Access:

When Name is an S4 object with an OM slot, OM() returns that slot directly rather than constructing a new object. This allows a consistent interface for both construction and retrieval:

```
om  <- OM(hist_obj)    # extract from `hist` or `mse`
```

### Derived Slots:

The Years slot is not a constructor parameter. It is derived automatically from nYear, pYear, CurrentYear, and Seasons via CalcYears() and is read-only. Use Years() to extract it from a constructed object.

### Interim Advice:

When the historical period ends before MPs should actually start being applied (e.g. the OM is conditioned on data to 2024 but MPs will not be implemented until 2028), set MPStartYear to the first calendar year MPs should run. Projection years before MPStartYear are "interim" years: the MP is not called, and advice is instead taken from InterimAdvice. See InterimAdvice() for how interim years are resolved and the data.frame format used to populate them.

### Sub-Object Slots:

Stock, Fleet, Obs, Imp, and Data each have dedicated constructor functions that also act as pass-through accessors when called with an om argument. Replacement is handled by the corresponding replacement functions in each constructor file:

```
Stock(om)  <- Stock("MyStock", ...)
Fleet(om)  <- list(s1 = list(f1 = Fleet("MyFleet", ...)))
```

```
Obs(om)    <- Obs("MyObs", ...)
Imp(om)    <- Imp("MyImp", ...)
Data(om)   <- Data(...)
```

**Accessing and Assigning Slots:**

All remaining slots in om objects can be accessed or assigned using functions matching the slot names. See OM-accessors for the full list:

```
Agency(om)
Agency(om) <- "DFO"
```

## Value

- `OM()` returns a new om object when `Name` is a character string or `NULL`.
- `OM()` returns `Name@OM` (an om object) when `Name` is an S4 object with an `OM` slot.

## See Also

- om for the class definition.
- OM-accessors for slot accessor and replacement functions.
- `InterimAdvice()` for populating interim TAC/Effort.
- `DataLag()` for the data-lag applied to the MP's `Data`.
- `FleetAllocation()` for splitting a stock/complex TAC among fleets.
- `SeasonalAllocation()` for splitting a periodic TAC/Effort across seasons.
- `RefSeason()` for reference point season handling in seasonal OMs.
- `EffortAllocation()` for splitting a scalar absolute Effort recommendation among fleets.
- `Stock()`, `Fleet()`, `Obs()`, `Imp()`, `Data()` for sub-object constructors and their pass-through accessors.
- `Years()`, `CalcYears()` for the derived time-step vector.
- `StockTargeting()` for the stock targeting sub-object.
- `Log()` for accessing runtime diagnostics.
- `PopulateOM()`, `runMSE()` for downstream use of the operating model.
- `ConvertOM()`, `ConvertMOM()` for converting legacy objects.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
om <- OM()
Agency(om)
Agency(om) <- "DFO"

nSim(om)
nSim(om) <- 100
```

```
Years(om)
```

---

`OM-accessors` *Access and Modify OM Slots*

---

## Description

Accessor and replacement functions for slots in om objects. Each function retrieves or replaces the value of the corresponding slot. All accessors also accept hist and mse objects, extracting the embedded `OM` slot transparently.

## Usage

```
Agency(x)

Agency(x) <- value

Author(x)

Author(x) <- value

Email(x)

Email(x) <- value

Region(x)

Region(x) <- value

Latitude(x)

Latitude(x) <- value

Longitude(x)

Longitude(x) <- value

Sponsor(x)

Sponsor(x) <- value

Source(x)

Source(x) <- value
```

```
nYear(x)

nYear(x) <- value

pYear(x)

pYear(x) <- value

CurrentYear(x)

CurrentYear(x) <- value

Seasons(x)

Seasons(x) <- value

Complexes(x)

Complexes(x) <- value

Herm(x) <- value

Relations(x)

Relations(x) <- value

nReps(x)

nReps(x) <- value

pStar(x)

pStar(x) <- value

maxF(x)

maxF(x) <- value

Seed(x)

Seed(x) <- value

Control(x)

Control(x) <- value
```

**Arguments**

| | |
|---|---|
| `x` | An om, hist, or mse object. |

| | |
|---|---|
| `value` | Replacement value for the corresponding slot. |

## Details

### Sub-Object Slots:

The `Stock`, `Fleet`, `Obs`, `Imp`, and `Data` slots are accessed and replaced via their own constructor functions, which act as pass-through accessors when called with an om argument. They are not listed here. See `Stock()`, `Fleet()`, `Obs()`, `Imp()`, `Data()`.

### Read-Only Slots:

`Years` is derived automatically from `nYear`, `pYear`, `CurrentYear`, and `Seasons` and is read-only. Use `Years()` to extract it.

### Replacing Slots in `hist` and `mse` Objects:

Replacement functions called on a hist or mse object modify its embedded `OM`, subject to when the slot is used:

- Metadata (`Agency`, `Author`, `Email`, `Region`, `Latitude`, `Longitude`, `Sponsor`, `Source`): can be replaced in om, hist, and mse objects.
- Settings used in projections (`Control`, `Interval`, `MPStartYear`, `InterimAdvice`, `DataLag`, `FleetAllocation`, `SeasonalAllocation`, `HistoricalWeight`, `EffortAllocation`, `nReps`, `pStar`): can be replaced in om and hist objects, and take effect in the next `Project()`.
- All other slots (e.g. `Seed`, `nYear`, `Seasons`, `maxF`, `Complexes`) are used to simulate the historical dynamics and can only be replaced in om objects; re-run `Simulate()` after changing them.

## Value

- Accessor functions return the value of the named slot.
- Replacement functions return `x` with the named slot updated.

## See Also

- `OM()` for the constructor.
- om for the class definition.
- `Stock()`, `Fleet()`, `Obs()`, `Imp()`, `Data()` for sub-object pass-through accessors.
- `Years()` for the read-only time-step vector.
- `Log()` for runtime diagnostics.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM()

Agency(om)
Agency(om) <- "DFO"

nSim(om)
nSim(om) <- 100

maxF(om)
maxF(om) <- 5
```

---

## om-class *The* om *S4 Class*

---

### Description

The om defines a complete operating model (OM) specification for use in Management Strategy Evaluation (MSE).

### Details

Direct construction via `methods::new()` is not recommended; use `OM()` instead, which populates defaults and derives `Years`.

All slots can be accessed or replaced using functions matching the slot name. See OM-accessors for the full list. Slots belonging to sub-objects (`Stock`, `Fleet`, `Obs`, `Imp`, `Data`) are accessed via their own constructors, which also serve as pass-through accessors when called with an om argument.

### Slots

`Name` Character. Name of the operating model. See `OM()`.

`Agency` Character. Name of the agency responsible for management. Supports Markdown. See `OM()`.

`Author` Character vector. Name(s) of the author(s). See `OM()`.

`Email` Character vector. Email address(es) corresponding to `Author`. Supports Markdown. See `OM()`.

`Region` Character. Name of the geographic region of the fishery. See `OM()`.

`Latitude` Numeric. Latitude (decimal degrees) of the centre of `Region`. See `OM()`.

`Longitude` Numeric. Longitude (decimal degrees) of the centre of `Region`. See `OM()`.

`Sponsor` Character. Organisation sponsoring development of the operating model. Supports Markdown. See `OM()`.

`nSim` Positive integer. Number of stochastic simulations. See `OM()`.

`nYear` Numeric. Number of historical years. See `OM()`.

`pYear` Numeric. Number of projection years. See `OM()`.

`CurrentYear` Integer. Final historical calendar year. See `OM()`.

`Seasons` Integer. Number of seasons per year. See `OM()`.

`Years` Numeric vector. Model time steps (including seasonal resolution if applicable). Derived from `nYear`, `pYear`, `CurrentYear`, and `Seasons` via `CalcYears()`. Read-only; see `Years()`.

`RefSeason` Integer vector or `NULL`. Only used for seasonal models (`Seasons > 1`). Season index/indices (`1..Seasons`) used as the reference snapshot(s) for reporting equilibrium reference points. `NULL` (default) auto-detects, independently per simulation, which season(s) have nonzero spawning contribution and averages the cross-sectional snapshot across them when more than one is detected. See `RefSeason()`.

`RefEffortYears` Numeric vector or `NULL`. Only used for seasonal models (`Seasons > 1`). One or more historical calendar years whose relative seasonal effort/catchability pattern is used to fix the seasonal shape of fishing mortality during per-recruit and MSY reference point optimization, decoupled from the year used for biological parameters. `NULL` (default) reuses the same year as the biological parameters. When more than one year is given, the per-season effort is averaged across those years before use. See `RefSeason()`.

`Stock` A stock object or named list of stock objects. See `Stock()`.

`Fleet` A hierarchical named list of fleet objects indexed by stock then fleet. Each stock must have the same number of fleets. See `Fleet()`.

`Obs` A hierarchical named list of obs objects indexed by stock and fleet. See `Obs()`.

`Imp` A hierarchical named list of imp objects indexed by stock and fleet. See `Imp()`.

`Data` A data object or list of data objects associated with the operating model. See `Data()`.

`DataLag` Integer. Number of years that data are lagged relative to management implementation. See `DataLag()`.

`CatchFrac` List. Named list of length 0 or `nStock(OM)`. Each element is an `nSim` by `nFleet` matrix (or a single row recycled across sims) giving the fraction of catch taken by each fleet, with rows summing to

1. Only used when there is more than one fleet and a historical `Depletion@Final` target is set for at least one stock, in which case it is the target fleet split that catchability is calibrated to reproduce. If left unspecified for a stock, it is derived from relative Effort times Catchability in the final historical year. See `FleetAllocation()`.

`FleetAllocation` List. Named list of length 0 or the number of stock complexes. Each element is an `nSim` by `nFleet` matrix, with rows summing to 1, controlling how the TAC is split among fleets during projection. If unspecified it falls back to `Allocation` (retained for backwards compatibility - see below), then `CatchFrac`, then the mean of removals over the last five historical years. See `FleetAllocation()`.

`Allocation` List. Deprecated alias for `FleetAllocation`, retained only so that objects saved before the rename still carry their data. If `FleetAllocation` is unset and `Allocation` is, its value is copied over during `Simulate()`/`Project()`. Not user-facing, set `FleetAllocation` directly instead.

`HistoricalWeight` List. Named list of length 0 or the number of stock complexes. Each element is a named numeric vector over fleet names, values in `[0,1]`, giving the weight placed on the historical seasonal pattern (vs. the population abundance pattern) when `SeasonalAllocation` is derived. `1` (default) uses the historical pattern only. See `SeasonalAllocation()`.

`SeasonalAllocation` List. Named list of length 0 or the number of stock complexes. Each element is an `nSim` (or 1, recycled) by `Seasons` by `nFleet` array, each `[sim, , fleet]` column summing to 1, controlling how a periodically-set TAC/Effort is split across the seasons of the interval it covers. If unspecified it is derived from `HistoricalWeight`. See `SeasonalAllocation()`.

`EffortAllocation` List. Named list of length 0 or the number of stock complexes. Each element is an `nSim` (or 1, recycled) by `nFleet` matrix, with rows summing to 1, splitting a scalar absolute Effort recommendation among fleets. If unspecified it falls back to to mean relative effort over the last five historical years. See `EffortAllocation()`.

`EFactor` List. Deprecated alias for `EffortAllocation`, retained only so that objects saved before the rename still carry their data. If `EffortAllocation` is unset and `EFactor` is, its value is copied over during `Simulate()`/`Project()`. Set `EffortAllocation` directly instead.

`Complexes` List. Defines stock complexes for data aggregation and management. See `OM()`.

`Herm` A list of stocktransition objects (built via `Herm()`), one per `From`/`To` stock pair, defining age-dependent reclassification of individuals between stocks (e.g. sequential hermaphroditism). See `OM()`.

`Relations` List. Biological or ecological relationships among stocks. See `OM()`.

`StockTargeting` A stocktargeting object. Defines fleet-level stock targeting weights and deviations. Built by `StockTargeting()` when the OM is populated.

`Interval` Numeric scalar or named numeric vector. Management update interval in years (regardless of `Seasons` - e.g. `Interval = 1` always means "once a year"), optionally per-MP via names. MPs that need to be called at every timestep of a seasonal OM (e.g. because they return a season-specific value derived directly from history) should declare `attr(mp, 'EverySeason') <- TRUE` instead of relying on a small `Interval` value.

`MPStartYear` Numeric or `NULL`. First calendar year in which MPs are applied. Projection years before `MPStartYear` are "interim" years - the MP is not called, and advice is instead built from `InterimAdvice`. `NULL` (default) means MPs start in the first projection year. See `InterimAdvice()`.

`InterimAdvice` A `data.frame` or `NULL`. Analyst-supplied fixed or stochastic TAC/Effort values for interim years (before `MPStartYear`). See `InterimAdvice()` for the required columns.

`nReps` Positive integer. Number of stochastic replicates for management advice. See `OM()`. Not currently used.

`pStar` Numeric. Percentile applied to stochastic management advice. See `OM()`. Not currently used.

`maxF` Numeric. Maximum allowable instantaneous fishing mortality. See `OM()`.

`Seed` Integer. Random number generator seed. See `OM()`.

`Control` Named list of operating model control settings. See OMControl.

`Misc` List. Miscellaneous objects or developer-use components.

`Log` `list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

`Source` Character. References to data sources or documentation. Supports Markdown. See `OM()`.

### See Also

- `OM()` for the constructor and OM-accessors for slot accessors.
- `Stock()`, `Fleet()`, `Obs()`, `Imp()`, `Data()` for sub-object constructors and their pass-through accessors.
- `Years()`, `CalcYears()` for the derived time-step vector.
- `StockTargeting()` for the stock targeting sub-object.
- `Log()` for accessing runtime diagnostics.
- `runMSE()`, `PopulateOM()` for downstream use of the operating model.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `stocktransition-class`

---

## `OM-legacy-class` *Class* `'OM'`

---

### Description

An object containing all the parameters needed to control the MSE which can be build from component Stock, Fleet, Obs, and Imp objects.

### Details

Almost all of these inputs are a vector of length 2 which describes the upper and lower bounds of a uniform distribution from which to sample the parameter.

### Slots

`Name` Name of the operating model

`Agency` Name of the agency responsible for the management of the fishery. Character string

`Region` Name of the general geographic region of the fishery. Character string

`Sponsor` Name of the organization who sponsored the OM. Character string

`Latitude` Latitude (decimal degrees). Negative values represent the South of the Equator. Numeric. Single value

`Longitude` Longitude (decimal degrees). Negative values represent the West of the Prime Meridian. Numeric. Single value

`nsim` The number of simulations

`proyears` The number of projected years

`interval` The assessment interval - how often would you like to update the management system?

`pstar` The percentile of the sample of the management recommendation for each method

`maxF` Maximum instantaneous fishing mortality rate that may be simulated for any given age class

`reps` Number of samples of the management recommendation for each method. Note that when this is set to 1, the mean value of the data inputs is used.

`cpars` A list of custom parameters. Time series are a matrix nsim rows by nyears columns. Single parameters are a vector nsim long. See `validcpars()`

`seed` A random seed to ensure users can reproduce results exactly

`Source` A reference to a website or article from which parameters were taken to define the operating model

### Objects from the Class

Objects can be created by calls of the form `new('OM', Stock, Fleet, Obs, Imp)`.

### Author(s)

T. Carruthers and A. Hordyk

---

`OMControl` *Operating Model Control Settings*

---

### Description

`OM@Control` is a named list of optional settings that modify how an om operating model is simulated and projected. Every setting has a default, so `OM@Control` can be empty.

### Details

Add or change individual settings with `Control(OM)$Setting <- value`. `Control(OM) <- list(...)` replaces the whole list. Unrecognised names produce a warning in `Simulate()` and are otherwise ignored.

### Simulation settings

Used by `Simulate()`:

| Setting | Default | Description |
|---|---|---|
| `MSYType` | `'Removals'` | `'Removals'` or `'Landings'`. Whether MSY-based and other F referen |
| `RefYears` | `NULL` | Year(s) of biological and fishery parameters used to calculate referenc |
| `CorrelatedRecDevs` | `TRUE` | Multi-stock OMs only. Whether projection recruitment deviations are |

| | | |
|---|---|---|
| HistRel | TRUE | Whether MICE Relations are applied in the historical period. |
| BackCalcEffort | TRUE | Whether effort is replaced by the effort implied by realised F when th |
| RefYield | list(lastnTS = 5) | Used when reference yield is calculated (see SimControl()). lastnTS |
| DataOM | NULL | Operating model quantities passed to MPs; see the DataOM section. |

### Projection settings

Used by Project() and runMSE():

| Setting | Default | Description |
|---|---|---|
| SeasonalAllocationYears | 5 | Seasonal OMs only. Nun |
| StockTargeting | list(n_recent = 5) | Multi-stock OMs only. n |
| EffortOptim | list(lambda_scale = 1, n_recent = 5, maxEval = 500) | Multi-complex OMs only |
| ProjectChunks | NULL | Number of simulation ch |
| DataOM | NULL | See the DataOM section. |

### DataOM

DataOM passes true operating model quantities to MPs, for example to build reference or "perfect information" MPs. The selected slots of the hist object are added to Data@Misc$DataOM (a hist object subset to the current simulation) each time the MP is called.

Any slot of hist except Data can be requested, e.g. Reference, Unfished, Number, Biomass, SBiomass, Landings, Effort, FDead. DataOM accepts:

- TRUE: all slots.
- A character vector of slot names, e.g. c('Biomass', 'Number').
- A named logical list, e.g. list(Biomass = TRUE, Number = TRUE). Elements set to FALSE are not included.

In OM@Control$DataOM, elements named after a hist slot apply to every MP. Any other element name is treated as the name of an MP, and its value (in any of the forms above, or FALSE) applies to that MP only:

```
Control(OM)$DataOM <- TRUE                              # all slots, every MP
Control(OM)$DataOM <- c('Biomass', 'Number')            # these slots, every MP
Control(OM)$DataOM <- list(Reference = TRUE,            # every MP
                           myMP = c('Biomass', 'Number'),# myMP only
                           MP2  = FALSE)                # nothing for MP2
```

MP functions can also request slots with a DataOM attribute, in the same forms:

```
attr(myMP, 'DataOM') <- c('Reference', 'Biomass')
```

The slots passed to an MP are:

- the MP's entry in OM@Control$DataOM, if there is one;

- otherwise, the union of the OM-wide slots in `OM@Control$DataOM` and the MP's `DataOM` attribute.

MP-specific entries are matched against the MP names used in `Project()`. `TuneMP()` projects renamed copies of the MP, so use the MP attribute or OM-wide slots for MPs that are tuned.

During projections, time-series slots (e.g. `Number`, `Biomass`, `Landings`, `Effort`) include only the time steps before the one the advice applies to (`Data@Misc$AdviceYear`). `OM`, `Unfished`, and `Reference` are passed in full.

The reference MPs (see ReferenceMPs) set `attr(MP, 'DataOM') <- list(Reference = TRUE)`. An MP that calls one of them internally needs `Reference` too, either from its own `DataOM` attribute or from `OM@Control$DataOM`.

### Internal settings

`Clone`, `CalcCatchAtSizeNeeded`, and `CalcCatchAtSizeCpp` are set internally by `Simulate()` and should not be modified.

### See Also

`OM()`, `Simulate()`, `Project()`, `SimControl()`

### Examples

```
## Not run:
OM <- SingleStockOM
Control(OM)$MSYType <- 'Landings'
Control(OM)$DataOM  <- list(Reference = TRUE)

myMP <- function(Data) {
  B <- Data@Misc$DataOM@Biomass
  refFMSY(Data)
}
class(myMP) <- 'mp'
attr(myMP, 'DataOM') <- 'Biomass'

Hist <- Simulate(OM)
MSE  <- Project(Hist, MPs = c('refFMSY', 'myMP'))

## End(Not run)
```

---

## `OMdoc` *Generate OM Documentation Report (deprecated)*

---

### Description

`OMdoc` has been deprecated and is no longer available. Install an earlier version of MSEtool (e.g. `remotes::install_version('MSEtool', version = '3.7.0')`) to use it.

### Usage

```
OMdoc(
  OM = NULL,
  rmd.source = NULL,
  overwrite = FALSE,
  out.file = NULL,
  inc.plot = TRUE,
  render = TRUE,
  output = "html_document",
  openFile = TRUE,
  quiet = FALSE,
  dir = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| `OM` | An object of class 'OM' or the name of an OM xlsx file |
| `rmd.source` | Optional. Name of the source.rmd file corresponding to the 'OM'. Default assumption is that the file is 'OM@Name.Rmd' |
| `overwrite` | Logical. Should existing files be overwritten? |
| `out.file` | Optional. Character. Name of the output file. Default is the same as the text file. |
| `inc.plot` | Logical. Should the plots be included? |
| `render` | Logical. Should the document be compiled? May be useful to turn off if there are problems with compiling the Rmd file. |
| `output` | Character. Output file type. Default is 'html_document'. 'pdf_document' is available but may require additional software and have some formatting issues. |
| `openFile` | Logical. Should the compiled file be opened in web browser? |
| `quiet` | TRUE to suppress printing of the pandoc command line. |
| `dir` | Optional file path to read the xlsx and rmd files. Default is `getwd()` |
| `...` | Optional additional named arguments provided to `runMSE` |

### Value

Nothing. Always errors.

### Author(s)

A. Hordyk

## OMexample *Copy example OM XL and OM Documentation*

### Description

Copy example OM XL and OM Documentation

### Usage

```
OMexample(dir = getwd())
```

### Arguments

| | |
|---|---|
| dir | the file path to copy the files to. |

### Examples

```
## Not run:
OMexample()

## End(Not run)
```

## OMinit *Initialize Operating Model*

### Description

Generates an Excel spreadsheet and a source.rmd file in the current working directory for specifying and documenting a MSEtool Operating Model.

### Usage

```
OMinit(
  name = NULL,
  ...,
  files = c("xlsx", "rmd"),
  dir = NULL,
  overwrite = FALSE
)
```

### Arguments

| | |
|---|---|
| `name` | The name of the Excel and source.rmd file to be created in the working directory (character). Use 'example' for a populated example OM XL and documentation file. |
| `...` | Optional MSEtool objects to use as templates: OM, Stock, Fleet, Obs, or Imp objects |
| `files` | What files should be created: 'xlsx', 'rmd', or c('xlsx', 'rmd') (default: both) to use as templates for the Operating Model. |
| `dir` | Optional file path to create the xlsx and rmd files. Default is `getwd()` |
| `overwrite` | Logical. Should files be overwritten if they already exist? |

### Value

name.xlsx and name.rmd files are created in the working directory.

### Author(s)

A. Hordyk

### Examples

```
## Not run:
# Create an Excel OM template and rmd file called 'myOM.xlsx' and 'myOM.rmd':
OMinit('myOM')

# Create an Excel OM template and text file called 'myOM.rmd' and 'myOM.rmd', using
# another OM as a template:
OMinit('myOM', myOM)

# Create an Excel OM template and text file called 'myOM.rmd' and 'myOM.rmd', using
# the Stock object 'Herring' as a template:
OMinit('myOM', Herring)

# Create an Excel OM template and text file called 'myOM.rmd' and 'myOM.rmd', using
# the Stock object 'Herring', and Obs object 'Generic_obs' as templates:
OMinit('myOM', Herring, Generic_obs)

## End(Not run)
```

---

## `OverageImp` *Overage Implementation Error*

---

### Description

An imp object representing a fleet that consistently realises more catch and effort than advised – e.g. under-reporting, unrecorded discards landed elsewhere, or effort creep beyond the recommended level.

### Usage

```
OverageImp
```

### Format

An imp object. Populated slots:

- `TAC` (impslot): `Mean` in `[1.05, 1.25]`, `SD` in `[0.05, 0.15]` – realised catch is on average 5-25% above the advised TAC, with additional simulation-to-simulation and year-to-year stochastic variation.
- `Effort` (impslot): same `Mean`/`SD` ranges, applied to effort-based advice.

Empty slots: `Size` (this example isolates TAC/effort overage; see PartialSizeComplianceImp for a size-regulation-focused example).

### See Also

imp, `Imp()`, `ImpSlot()`, FullComplianceImp, UnderageImp, PartialSizeComplianceImp

Other imp: `FullComplianceImp`, `Imp()`, `ImpSlot()`, `PartialSizeComplianceImp`, `UnderageImp`, `imp-class`, `impslot-class`

### Examples

```
OverageImp
slot(OverageImp, "TAC")
```

---

## `Overages` *Imp class objects*

---

### Description

Example objects of class Imp

### Usage

```
Overages

Perfect_Imp
```

### Format

An object of class `Imp` of length 1.

An object of class `Imp` of length 1.

### Examples

```
avail("Imp")
```

`PartialSizeComplianceImp`
*Partial Size-Regulation Compliance Implementation Error*

## Description

An imp object isolating partial fleet adoption of a size-based regulation (a change to `Advice@Retention` and/or `Advice@Selectivity`), while TAC and effort advice are still implemented perfectly.

## Usage

```
PartialSizeComplianceImp
```

## Format

An imp object. Populated slots:

- `TAC` (impslot): `Mean = 1`, `SD = 0` – perfect TAC implementation, as in FullComplianceImp.
- `Effort` (impslot): `Mean = 1`, `SD = 0` – perfect effort implementation, as in FullComplianceImp.
- `Size` (impslot): `Compliance = 0.6` – only 60% of the fleet adopts a newly-advised size regulation in the year it changes; the remaining 40% continues fishing under the previous Retention/Selectivity curve. Selectivity and retention in that year are the compliance-weighted average of the new and previous curves.

## See Also

imp, `Imp()`, `ImpSlot()`, FullComplianceImp, OverageImp, UnderageImp

Other imp: `FullComplianceImp`, `Imp()`, `ImpSlot()`, `OverageImp`, `UnderageImp`, `imp-class`, `impslot-class`

## Examples

```
PartialSizeComplianceImp
slot(PartialSizeComplianceImp, "Size")
```

`PerformanceMetric` *Performance Metrics Methods*

## Description

Performance metric (PMs) methods for your management strategy evaluation.

## Usage

```
P10(MSEobj = NULL, Ref = 0.1, Yrs = NULL)

P50(MSEobj = NULL, Ref = 0.5, Yrs = NULL)

P100(MSEobj = NULL, Ref = 1, Yrs = NULL)

PNOF(MSEobj = NULL, Ref = 1, Yrs = NULL)

LTY(MSEobj = NULL, Ref = 0.5, Yrs = -10)

STY(MSEobj = NULL, Ref = 0.5, Yrs = 10)

Yield(MSEobj = NULL, Ref = 1, Yrs = NULL)

PGK(MSEobj = NULL, Ref = 1, Yrs = NULL)

AAVY(MSEobj = NULL, Ref = 0.2, Yrs = NULL)

AAVE(MSEobj = NULL, Ref = 0.2, Yrs = NULL)
```

## Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `Ref` | Reference point for calculating the performance metric. See details. |
| `Yrs` | Numeric vector of length 2 with year indices to summarize performance. If NULL, the performance is summarized over all projection years. |

## Details

Performance Metric definitions:

| | |
|---|---|
| `P10` | Probability B > 0.1 BMSY |
| `P50` | Probability B > 0.5 BMSY |
| `P100` | Probability B > BMSY |
| `PNOF` | Probability F < FMSY |
| `LTY` | Probability Long-Term Yield > 0.5 Relative Yield |
| `STY` | Probability Short-Term Yield > 0.5 Relative Yield |
| `AAVY` | Probability AAVY < 0.2 (Average Annual Variability in Yield) |

| | |
|---|---|
| AAVE | Probability AAVE < 0.2 (Average Annual Variability in Effort) |
| Yield | Average Yield (relative to Reference Yield) |

Argument Ref provides the ratio relative to the reference point for calculating the performance metric. For biomass-based PMs (P10, P50, P100), this is the fraction of BMSY. For PNOF, the fraction of FMSY. For Yield (and LTY/STY), the fraction of the Reference Yield. For AAVY is it the maximum acceptable variability in yield (i.e, default for AAVY is Ref=0.2)

The Yrs argument defines the number of years to calculate the performance statistic over. A value of NULL, the default for AAVY, AAVE, P10, P50, P100, and PNOF, means that the performance metric is calculated over all projection years. A numeric vector of length two is used to specify the first and last year, e.g, if Yrs=c(1,10) the performance statistic is calculated over the first 10 projection years. A numeric vector of length one with positive or negative value respectively can be used to specify the first $x$ or last $x$ years, e.g, Yrs=10 is first 10 years, and Yrs=-10 is the last 10 years. See ChkYrs for more details.

By default Long-Term Yield (LTY) is the Yield in the last ten years of the projection period in the MSE, and Short-Term Yield (STY) is that in the first 10 years of the projection period.

## Value

An object of class PMobj

## Examples

```
## Not run:
myMSE <- runMSE()
P10(myMSE)
P50(myMSE)
P100(myMSE)
PNOF(myMSE)
LTY(myMSE)
STY(myMSE)
AAVY(myMSE)
AAVE(myMSE)
Yield(myMSE)

## End(Not run)
```

---

# perrecruit-class *The* perrecruit *S4 Class*

---

## Description

An S4 class storing per-recruit quantities evaluated across a vector of apical fishing mortality values. Produced internally by CalcPerRecruit() during reference point and equilibrium calculations.

## Details

### Spawning numbers-per-recruit:

When `SpawnTimeFrac > 0` for any stock, `NPRF_SP` differs from `NPRF`: survival to the spawn date is computed as

$$N \cdot \exp(-Z \cdot \texttt{SpawnTimeFrac})$$

so that `SPR`, `SBiomass`, and `SProduction` reflect abundance at the time of spawning rather than at the start of the year. When `SpawnTimeFrac = 0` for all stocks, `NPRF_SP` is not populated.

### SPFrom:

When stocks share a spawning component (e.g. a complex), the `SPFrom` slot of each stock's `SRR()` determines which stock's (or weighted blend of stocks') `SPRF/SPR0` is used when computing relative `SPR`. This is handled upstream in `CalcPerRecruit()` before the `SPRF` and `SPR0` slots are populated.

### Relationships between slots:

$$\texttt{SPR} = \texttt{SPRF}/\texttt{SPR0}$$

$$\texttt{SBiomass} = \sum_a \texttt{NPRF\_SP}_a \cdot W_a \cdot m_a$$

$$\texttt{SProduction} = \sum_a \texttt{NPRF\_SP}_a \cdot f_a$$

where $W_a$, $m_a$, and $f_a$ are weight, maturity, and fecundity at age respectively.

## Slots

`apicalF` Numeric vector of apical fishing mortality values at which per-recruit quantities were evaluated. Length `nF`.

`NPR0` Numbers-per-recruit at unfished conditions. Array (`Sim × Age × Year`) or a list by stock.

`NPR0_SP` Spawning numbers-per-recruit at unfished conditions, accounting for `SpawnTimeFrac`. Array (`Sim × Age × Year`) or a list by stock. Equal to `NPR0` when `SpawnTimeFrac` is zero for all stocks.

`NPRF` Numbers-per-recruit at fishing mortality. Array (`Sim × Age × Year × F`) or a list by stock, where the `F` dimension has length `nF`.

`NPRF_SP` Spawning numbers-per-recruit at fishing mortality, accounting for `SpawnTimeFrac`. Array (`Sim × Age × Year × F`) or a list by stock. Populated only when `SpawnTimeFrac > 0` for any stock; otherwise `NULL`.

`SPR0` Spawning-per-recruit at unfished conditions. Array (`Sim × Stock × Year`).

`SPRF` Spawning-per-recruit at fishing mortality. Array (`Sim × Stock × Year × F`).

`SPR` Spawning-per-recruit relative to unfished, i.e. `SPRF / SPR0`. Array (`Sim × Stock × Year × F`).

`Biomass` Total biomass-per-recruit. Array (`Sim × Stock × Year × F`).

`SBiomass` Spawning biomass-per-recruit. Array (`Sim × Stock × Year × F`).

`SProduction` Spawning production-per-recruit, computed from fecundity-at-age rather than weight × maturity. Array (`Sim × Stock × Year × F`).

`Removals` Total removals-per-recruit (landings + dead discards), summed over fleets. Array (`Sim × Stock × Year × F`).

`Landings` Landings-per-recruit, summed over fleets. Array (`Sim × Stock × Year × F`).

`Misc` List reserved for internal use.

### See Also

`CalcPerRecruit()`

---

`plot.Data` *Plot Data object*

---

### Description

Creates plots of the Data object in the R console. Wrapper for `summary(Data)`

### Usage

```
## S3 method for class 'Data'
plot(
  x,
  wait = TRUE,
  i = 1,
  plots = "all",
  rmd = FALSE,
  head = "##",
  tplot = 25,
  ...
)
```

### Arguments

| | |
|---|---|
| `x` | An object of class Data |
| `wait` | Logical. Wait for key press before next plot? |
| `i` | iteration number for the Data object. |
| `plots` | Character. What plots to show? `all`, `TS`, `CAA`, `CAL`, `PD` for all plots, time-series, catch-at-age, catch-at-length, and probability distributions respectively |
| `rmd` | Logical. Used in a rmd file? |
| `head` | Character. Heading for rmd file. Default is '##' (second level heading) |
| `tplot` | Integer. Number of plots per page. Default 25 |
| `...` | Not used |

`plot.MMSE` *Standard plot for an object of class MMSE (multi MSE)*

## Description

Plot the projected biomass, fishing, mortality rate and yield for all stocks and MPs

## Usage

```
## S3 method for class 'MMSE'
plot(
  x = NULL,
  maxcol = 6,
  qcol = rgb(0.4, 0.8, 0.95),
  lcol = "dodgerblue4",
  quants = c(0.05, 0.25, 0.75, 0.95),
  curyr = 2018,
  addline = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | Object of class `MMSE`. A Multi-OM object created by `multiMSE(MOM, ...)` |
| `maxcol` | Integer. The maximum number of columns (MPs) to be plotted in each plot |
| `qcol` | Character, color. The color of the inner percentile range |
| `lcol` | Character, color. The color of the outer percentile range. |
| `quants` | Numeric vector. The percentiles that are plotted (LB2,LB1,UB1,UB2). LB2 and UB2 are the outer percentiles, LB1 and UB1 are the inner percentiles. |
| `curyr` | Integer. The current year from which projections start. |
| `addline` | Logical. Should two individual simulations be added to the percentile plots? |
| `...` | Not used |

## Author(s)

T.Carruthers

---

`plot.MOM` *Standard plot for an object of class MOM*

---

### Description

Plot the stocks, fleets, catch fractions and relationships in multi operating model object

### Usage

```
## S4 method for signature 'MOM,missing'
plot(x, silent = TRUE, maxsims = 6)
```

### Arguments

| | |
|---|---|
| `x` | Object of class `MOM`. A Multi-OM object created by `new('MOM', ...)` |
| `silent` | Logical. Do you wish to see print outs / warnings? |
| `maxsims` | Integer. What are the maximum number of individual simulations you wish to plot? |

### Author(s)

T.Carruthers

---

`plot.MSE` *Plot MSE object*

---

### Description

Plot MSE object

### Usage

```
## S3 method for class 'MSE'
plot(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | object of class MSE |
| `...` | other parameters passed to plot (currently ignored) |

`plot.pars` *Plot Operating Model Object*

## Description

Generate HTML reports with plots of operating model components ("Stock", "Fleet", "Obs", and "Imp"), the historical simulations ("Hist"), or the complete OM ("OM").

The individual component plots of objects of class `Stock` and `Fleet` can also be generated by using the generic `plot.pars` function. See Examples below.

## Usage

```
## S3 method for class 'pars'
plot(
  x,
  Object,
  Stock = NULL,
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
  plotPars = NULL,
  html = FALSE,
  open = TRUE,
  dev = FALSE,
  ...
)

## S3 method for class 'Stock'
plot(
  x,
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
```

```
  plotPars = NULL,
  open = TRUE,
  dev = FALSE,
  ...
)

## S3 method for class 'Fleet'
plot(
  x,
  Stock = NULL,
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
  plotPars = NULL,
  open = TRUE,
  dev = FALSE,
  ...
)

## S3 method for class 'Obs'
plot(
  x,
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
  plotPars = NULL,
  open = TRUE,
  dev = FALSE,
  ...
)

## S3 method for class 'Imp'
plot(
  x,
```

```
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
  plotPars = NULL,
  open = TRUE,
  dev = FALSE,
  ...
)

## S3 method for class 'Hist'
plot(
  x,
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
  plotPars = NULL,
  open = TRUE,
  dev = FALSE,
  ...
)

## S3 method for class 'OM'
plot(
  x,
  nsamp = 3,
  nsim = 200,
  nyears = 50,
  proyears = 28,
  output_file = NULL,
  output_dir = getwd(),
  quiet = TRUE,
  tabs = TRUE,
  title = NULL,
  date = NULL,
```

```
  plotPars = NULL,
  open = TRUE,
  dev = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| x | An object of class Stock, Fleet, Obs, Imp, Hist, or OM, OR one of the following character strings for Object of class Stock: "M", "Growth", "Maturity", "Recruitment", "Spatial", or "Depletion" and for Object of class Fleet: "Effort", "Catchability", "MPA", and "Selectivity". |
| Object | An object of class Stock or Fleet |
| Stock | An object of class Stock required for Fleet parameters |
| nsamp | The number of random samples to show in the plot |
| nsim | The number of simulations (only used for objects not of class OM) |
| nyears | The number of historical years (only used for objects not of class OM) |
| proyears | The number of projection years (only used for objects not of class OM) |
| output_file | Name of the output html file (without file extension) |
| output_dir | Output directory. Defaults to getwd() |
| quiet | An option to suppress printing of the pandoc command line |
| tabs | Include tabs in the HTML file? |
| title | Optional title for the markdown report |
| date | Optional date for the markdown report |
| plotPars | A named list with options for plots:<br>• breaks - numeric. Number of breaks in histograms.<br>• col - character. Color of histograms.<br>• axes - logical. Include axes in histogram?<br>• cex.main - numeric. Size of main title in plots.<br>• lwd - numeric. Line width for time-series plots. |
| html | Logical. Compile to a HTML report (TRUE) or print plots in R console (FALSE) |
| open | Logical. Open the html file? |
| dev | Logical. For development use only. |
| ... | Not used |

### Examples

```
## Not run:
# Plot Stock Object:
Stock <- MSEtool::Albacore
plot(Stock)
```

```
# Individual plots:
plot("M", Stock)
plot("Growth", Stock)
plot("Maturity", Stock)
plot("Recruitment", Stock)
plot("Spatial", Stock)
plot("Depletion", Stock)

# Plot Fleet Object
Fleet <- MSEtool::Generic_DecE
plot(Fleet, Stock)

# Individual plots:
plot("Effort", Fleet, Stock)
plot("Catchability", Fleet, Stock)
plot("MPA", Fleet, Stock)
plot("Selectivity", Fleet, Stock)


# Plot Obs Object
Obs <- MSEtool::Imprecise_Unbiased
plot(Obs)

# Plot Imp Object
Imp <- MSEtool::Overages
plot(Imp)


# Plot Hist Object
OM <- MSEtool::testOM
Hist <- Simulate(OM)
plot(Hist)

# Plot OM Object
plot(OM)

## End(Not run)
```

---

`plot_comp` *Plot Age and Size Composition Data*

---

## Description

Bubble plots of catch-at-age or catch-at-size composition: `Year` on the x-axis, `Age`/size `Class` on the y-axis, and bubble area proportional to the proportion of that year's catch falling in that age/class. `PlotLandingsAtAge()`, `PlotDiscardsAtAge()`, `PlotLandingsAtSize()`, and `PlotDiscardsAtSize()` plot the `Landings()`/`Discards()` `byAge`/`bySize` composition data for a hist, mse, or data object.

### Usage

```
PlotLandingsAtAge(
  object,
  byStock = NULL,
  byFleet = NULL,
  Sim = NULL,
  Years = NULL,
  IncHist = TRUE,
  Stocks = NULL
)

PlotDiscardsAtAge(
  object,
  byStock = NULL,
  byFleet = NULL,
  Sim = NULL,
  Years = NULL,
  IncHist = TRUE,
  Stocks = NULL
)

PlotLandingsAtSize(
  object,
  byStock = NULL,
  byFleet = NULL,
  Sim = NULL,
  Years = NULL,
  IncHist = TRUE,
  Stocks = NULL
)

PlotDiscardsAtSize(
  object,
  byStock = NULL,
  byFleet = NULL,
  Sim = NULL,
  Years = NULL,
  IncHist = TRUE,
  Stocks = NULL
)
```

### Arguments

| | |
|---|---|
| `object` | A hist, mse, or data object. `Sim`, `Years`, `IncHist`, and `Stocks` have no effect for data objects, which hold a single (stock-complex, simulation) realization. |
| `byStock` | One of `TRUE`, `FALSE`, or `NULL` (default). `TRUE` facets by stock. `FALSE` sums counts across stocks before computing proportions; for `*AtSize()`, this |

raises an error if stocks don't all share the exact same size-class grid (see compdata). `NULL` facets automatically when `object` has more than one (selected) stock. No effect for data objects (single stock complex).

`byFleet` — One of `TRUE`, `FALSE`, or `NULL` (default). `TRUE` facets by fleet. `FALSE` sums counts across fleets before computing proportions, via `Landings()`/`Discards()`; for `*AtSize()`, this raises an error if fleets don't all share the exact same size-class grid, since fleets are not required to (see compdata). `NULL` facets automatically when `object` has more than one fleet.

`Sim` — Integer or `NULL` (default). For hist/mse objects, which simulation replicate to plot. `NULL` takes the median proportion across all simulations (computed per simulation, then summarised), rather than the median of raw counts.

`Years` — Optional numeric vector. Subset the time series to these years before plotting.

`IncHist` — Logical. For mse objects, include the historical period? Default `TRUE`. No effect for hist objects.

`Stocks` — Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks).

## Details

Proportions are computed within each `Year` (and `Stock`/`Fleet`/`MP` panel), i.e. each year's bubbles sum to 1 within a panel, making years with very different total catch comparable and cohorts easier to track visually. For hist/mse objects with `Sim = NULL`, proportions are computed per simulation and then summarised with the median across simulations, cell by cell – not the proportion of median counts.

Panels are faceted by whichever of `Stock`, `Fleet`, and `MP` have more than one level after filtering (`MP` always facets when mse objects have more than one MP – there is no color channel free for it, since bubble size already encodes proportion).

If the relevant `*AtAge`/`*AtSize` data is not populated for a data object, the function prints an informational message and returns `NULL` invisibly rather than plotting an empty panel.

## Value

A `ggplot` object, or `NULL` invisibly if the relevant data slot isn't populated.

## See Also

`Landings()`, `Discards()`, compdata

## Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
PlotLandingsAtAge(Hist)
PlotLandingsAtSize(Hist)
PlotLandingsAtSize(Hist, Sim = 3)
```

```
MSE <- Project(Hist, ExampleMPs())
PlotDiscardsAtSize(MSE, byMP = TRUE)

dat <- Hist@Data[[1]][[1]]
PlotLandingsAtSize(dat)

## End(Not run)
```

---

plot_data *Plot Observed Fishery Data*

---

## Description

Plotting functions for data objects: the observed (or simulated, with observation error) fishery timeseries, as opposed to the underlying simulated dynamics plotted by plot_hist. A data object holds a single simulation/stock realization (no `Sim` dimension), so these functions draw a single line per fleet or index rather than a median/ribbon summary.

## Usage

```
PlotCPUE(object, byFleet = NULL, Season = NULL, units = TRUE)

PlotSurvey(object, byFleet = NULL, Season = NULL, units = TRUE)

PlotData(
  object,
  byFleet = NULL,
  AggregateYear = FALSE,
  Season = NULL,
  units = TRUE,
  showDiscards = FALSE
)

## S4 method for signature 'data,ANY'
plot(x, y, ...)
```

## Arguments

| | |
|---|---|
| `object` | A data object. |
| `byFleet` | Logical. Controls how multiple fleets are displayed. Default (`NULL`) draws every fleet as its own colored line in a single panel. `TRUE` facets by fleet instead (one panel per fleet). `FALSE` sums across fleets and draws a single total line. No effect when the slot has only one fleet. |
| `Season` | Integer. For seasonal data (`object@Seasons > 1`), restrict `PlotCPUE()`/`PlotSurvey()` to a single season (1 = the first timestep of each year), giving one snapshot per year instead of the full sub-annual series. Default `NULL` (no filtering). Ignored for non-seasonal data and by `PlotLandings()`/`PlotDiscards()`/`PlotEffort()` (see `AggregateYear`). |

`units` Logical or a character unit string. `TRUE` (default) labels the y-axis with the slot's own `Units` (e.g. `object@CPUE@Units`, `"kg/trip"`) when it's set. `FALSE` suppresses the unit label. For `PlotLandings()`/`PlotDiscards()` only, `object@Landings@Units`/ `object@Discards@Units` is a mass unit, so a character string (`"kg"`, `"t"`, `"lb"`, etc.) both relabels the axis *and* rescales the plotted values into that unit; requesting a character unit for `PlotCPUE()`/ `PlotSurvey()` is an error, since those `Units` (e.g. `"kg/trip"`) aren't in a recognized conversion table. No effect on `PlotEffort()`, which has no unit concept for a single `data-class` object.

`AggregateYear` Logical. For seasonal data (`object@Seasons > 1`), sum `PlotLandings()`/`PlotDiscards()`/`PlotE` sub-annual values into whole-year totals, since these are flow quantities. Default `FALSE`. Ignored for non-seasonal data and by `PlotCPUE()`/ `PlotSurvey()` (see `Season`).

`showDiscards` Logical. `PlotData()` only. By default (`FALSE`), the `Discards` panel is omitted when it's populated but every value (across sims/years/fleets) is zero. Set `TRUE` to always show it.

## Details

`PlotCPUE()` and `PlotSurvey()` plot the `CPUE` and `Survey` slots respectively. `PlotLandings()`, `PlotDiscards()`, `PlotRemovals()`, and `PlotEffort()` (see plot_hist) also accept a data object and are documented here for that case.

If the relevant slot is not populated (`NULL`, e.g. no survey configured for this OM), the function prints an informational message and returns `NULL` invisibly rather than plotting an empty panel. `PlotRemovals()` plots `Landings + Discards`; it's omitted (with the same message) only when *both* are unpopulated, and silently treats either as `0` when just one is populated.

`PlotData()` arranges whichever of `Landings`, `Discards`, `Effort`, `CPUE`, and `Survey` are populated into a single figure with `patchwork::wrap_plots()`, silently omitting panels for slots that aren't; unlike the individual `Plot*()` functions it does not print a message for missing slots. Fleet color legends are collected into a single shared legend (`patchwork::plot_layout(guides = 'collect')`) only when every panel with a color legend maps the same set of fleets; otherwise each panel keeps its own legend, since a shared one would misrepresent panels covering a different fleet set (e.g. `Landings` and `Survey` are often reported by different fleets). If none of the five slots are populated, it prints a message and returns `NULL` invisibly instead of an empty figure. The `Discards` panel is also omitted whenever it's populated but every value (across sims/years/ fleets) is zero, unless `showDiscards = TRUE`.

## Value

A `ggplot` object (a `patchwork` object for `PlotData()`), or `NULL` invisibly if the relevant slot isn't populated.

## See Also

plot_hist, `Landings()`, `Discards()`, `CPUE()`, `Survey()`

### Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
dat  <- Hist@Data[[1]][[1]]

PlotLandings(dat)
PlotDiscards(dat)
PlotEffort(dat)
PlotCPUE(dat)
PlotSurvey(dat)
PlotData(dat)

PlotLandings(dat, byFleet = TRUE)
PlotLandings(dat, byFleet = FALSE)

## End(Not run)
```

---

## plot_hist *Plot Historical and Projected Dynamics*

---

### Description

Plotting functions for hist and mse objects, summarizing simulated numbers, biomass, spawning biomass, spawning production, fishing mortality, vulnerable biomass, landings, discards, interactions, removals, and effort over time.

### Usage

```
PlotNumber(
  object,
  byStock = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  Season = NULL,
  units = TRUE,
  Stocks = NULL
)

PlotBiomass(
  object,
  byStock = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
```

```
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  relative = c("none", "B0", "BMSY"),
  type = c("Equilibrium", "Dynamic"),
  Season = NULL,
  units = TRUE,
  Stocks = NULL
)

PlotSBiomass(
  object,
  byStock = NULL,
  byFemale = TRUE,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  relative = c("none", "B0", "BMSY"),
  type = c("Equilibrium", "Dynamic"),
  Season = NULL,
  units = TRUE,
  Stocks = NULL
)

PlotSProduction(
  object,
  byStock = NULL,
  byFemale = TRUE,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  relative = c("none", "B0", "BMSY"),
  type = c("Equilibrium", "Dynamic"),
  Season = NULL,
  units = TRUE,
  Stocks = NULL
)

PlotF(
  object,
  what = c("FDead", "FInteract", "FRetain"),
```

```
  byStock = NULL,
  byFleet = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  relative = c("none", "FMSY"),
  Season = NULL,
  Stocks = NULL
)

PlotVBiomass(
  object,
  byStock = NULL,
  byFleet = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  VBType = c("Removals", "Landings"),
  Season = NULL,
  units = TRUE,
  Stocks = NULL
)

PlotLandings(
  object,
  byStock = NULL,
  byFleet = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  AggregateYear = FALSE,
  units = TRUE,
  Stocks = NULL
)

PlotDiscards(
  object,
  byStock = NULL,
  byFleet = NULL,
```

```
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  AggregateYear = FALSE,
  units = TRUE,
  Stocks = NULL
)

PlotInteractions(
  object,
  byStock = NULL,
  byFleet = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  AggregateYear = FALSE,
  Stocks = NULL
)

PlotRemovals(
  object,
  byStock = NULL,
  byFleet = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  AggregateYear = FALSE,
  units = TRUE,
  Stocks = NULL
)

PlotDynamics(
  object,
  byStock = NULL,
  byFleet = FALSE,
  byFemale = TRUE,
  byMP = FALSE,
  probs = c(0.05, 0.95),
  nsim = 0,
```

```
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  relative = c("none", "B0", "BMSY"),
  type = c("Equilibrium", "Dynamic"),
  Season = NULL,
  AggregateYear = FALSE,
  units = TRUE,
  Stocks = NULL
)
```

## Arguments

| | |
|---|---|
| `object` | A hist or mse object. `PlotLandings()`, `PlotDiscards()`, `PlotRemovals()`, and `PlotEffort()` also accept a data object (e.g. `Hist@Data[[1]][[1]]`), in which case a single observed timeseries (no ribbon) is plotted, with `byFleet` and `AggregateYear` applied as described in plot_data; other arguments (`byStock`, `probs`, `nsim`, `Years`, `free_y`, `IncHist`, `byMP`) have no effect in that case. For `PlotRemovals()`, the `data`-class series is `Landings + Discards` summed together. |
| `byStock` | One of `TRUE`, `FALSE`, `'sum'`, or `NULL` (default). `TRUE` facets by stock. `FALSE` colors each stock as its own line on a single panel, without summing. `'sum'` sums across stocks into a single line; for relative time series (`relative != 'none'`, see `relative`) the numerator and denominator are summed across stocks separately before dividing, giving an aggregate ratio (e.g. overall depletion) rather than a sum of per-stock ratios. `NULL` (default) resolves to `TRUE` when `object` has more than one (selected) stock, otherwise `FALSE`. When `FALSE` and `object` has more than one MP, MP is faceted instead of colored, since stock now occupies the color channel. |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the shaded ribbon across simulations. Default `c(0.05, 0.95)`. |
| `nsim` | Integer. Number of individual simulation replicates to overlay as thin lines, in addition to the median/ribbon. Default `0` (none). |
| `Years` | Optional numeric vector. Subset the time series to these years before plotting. |
| `free_y` | Logical. When faceting by stock and/or fleet, let each panel's y-axis scale independently. Default (`NULL`) is `TRUE`, except for `PlotBiomass()`/`PlotSBiomass()`/`PlotSProdu` with `relative != "none"`, where it defaults to `FALSE` (ratios are already on a comparable scale across stocks). |
| `IncHist` | Logical. For mse objects, include the historical period? `FALSE` is a shortcut for subsetting `Years` to the projection period only, while still starting each MP's line from the last historical value. Default `TRUE`. No effect for hist objects. |
| `byMP` | Logical. For mse objects with more than one MP, facet by MP instead of coloring by MP. Only takes effect when `byStock` and `byFleet` are not *both* already faceting (i.e. there is a free facet dimension); otherwise MP falls back to color. Default `FALSE`. |

Season — Integer. For seasonal models (OM@Seasons > 1), restrict PlotNumber()/PlotBiomass()/PlotSBi PlotF()/PlotVBiomass() to a single season (1 = the first timestep of each year), giving one snapshot per year instead of the full sub-annual sawtooth. These are point-in-time quantities, so no season should be summed/averaged to get an annual value the way Landings()/Discards() can. Default NULL (no filtering). Ignored for non-seasonal models.

units — For PlotEffort(): character or NULL. One of "Effort" (the raw effort index), "Trips" (Effort x TripsScalar), or "Anglers" (Effort x TripsScalar x AnglerPerT TripsScalar/ AnglerPerTrip are read from the first stock's Fleet object for each fleet (see Effort()). NULL (default) auto-selects, per fleet, the most specific unit with the required data. When byFleet = TRUE, each fleet's facet uses its own resolved unit (labelled in the facet strip if they differ); otherwise the weakest unit available across all plotted fleets is used for all of them, since a raw index and real trip counts can't share one axis or line.

For PlotNumber(), PlotBiomass(), PlotSBiomass(), PlotSProduction(), PlotVBiomass(), PlotLandings(), PlotDiscards(), and PlotRemovals(): logical or a character unit string. TRUE (default) labels the y-axis with the unit implied by the stock's own Units slots – e.g. a Weight@Units of "kg" combined with an SRR@Units of 1000 (R0 in thousands of fish) labels Biomass in "t", since the raw values are already Number() x WeightAtAge. PlotNumber() instead shows the SRR@Units scaling alone (e.g. "Number (thousands)"). The label is only added when the relevant Units are set and agree across every plotted stock; otherwise it's silently omitted. FALSE suppresses the unit label entirely (e.g. when a model's Units slots aren't set correctly). A character string (mass unit for all of these except PlotNumber(), e.g. "kg", "t", "lb") both relabels the axis *and* rescales the plotted values into that unit; requesting a unit the stored units can't be converted to (e.g. a target when Weight@Units is unset, or an unrecognized string) is an error. Ignored (relative ratios are already unitless) when PlotBiomass()/PlotSBiomass()/ PlotSProduction() are called with relative != "none".

Stocks — Character or numeric vector. Restrict the plot to specific stocks, either by name (matching StockNames()) or by index. Default NULL (all stocks). byStock and, for PlotSBiomass()/ PlotSProduction(), the automatic byFemale stock selection (see Details) are evaluated against this selection rather than the full set of stocks in object – e.g. Stocks = "Female" with byStock = FALSE plots that one stock's series under its own name, without needing byFemale to guess at it; providing Stocks explicitly always disables the automatic byFemale complex-matching logic (byStock = 'sum' then sums exactly the stocks selected, instead of guessing the female one).

relative — Character. For PlotBiomass(), PlotSBiomass(), and PlotSProduction(): plot absolute values ("none", default), or relative to unfished ("B0", via B_B0()/SB_SB0()/SP_SP0()) or to the MSY reference point ("BMSY", via B_BMSY()/SB_SBMSY()/SP_SPMSY()). For PlotF(): "none" (default, apical what fishing mortality) or "FMSY" (via F_FMSY()). See byStock above for how byStock = 'sum' aggregates a relative time series across stocks.

| | |
|---|---|
| type | Character. One of "Equilibrium" or "Dynamic". Which unfished baseline to use when relative = "B0"; see B_B0(). Ignored otherwise. |
| byFemale | Logical. See Details. Only applies to PlotSBiomass()/PlotSProduction(). |
| what | Character. For PlotF(), which fishing mortality series to plot: "FDead" (default, mortality of fish killed), "FInteract" (mortality of all fish encountered), or "FRetain" (mortality of fish retained). See FDead()/FInteract()/FRetain(). Apical (max-over-age) fishing mortality is plotted. Ignored when relative = 'FMSY', which is always based on FDead. |
| byFleet | Logical. Facet by fleet? Default (NULL) facets automatically when object has more than one fleet, except in PlotDynamics()/plot() where it defaults to FALSE. Only applies to PlotF()/PlotLandings()/PlotDiscards()/PlotInteractions PlotRemovals()/PlotEffort(). Ignored by PlotF() when relative = 'FMSY' (always summed over fleets). When object is a data object, byFleet instead follows the three-state NULL/TRUE/FALSE behavior described in plot_data (default NULL colors by fleet rather than auto-faceting). |
| VBType | Character. For PlotVBiomass(), "Removals" (default, weights biomass by fish that die from the encounter, landed or discarded) or "Landings" (weights by landed fish only); see VBiomass(). |
| AggregateYear | Logical. For seasonal models, sum PlotLandings()/PlotDiscards()/PlotInteractions()/ PlotRemovals()/PlotEffort() sub-annual values into whole-year totals, unlike Season above (these are flows, so summing across a year is valid, unlike the point-in-time state variables). Default FALSE. Ignored for non-seasonal models. |

## Details

Each function extracts the relevant time series with the corresponding accessor (Number(), Biomass(), SBiomass(), SProduction(), FDead()/FInteract()/FRetain(), Landings(), Discards(), Interactions(), Effort()), summarizes across simulations as a median line with a shaded quantile ribbon, and returns a ggplot object. Set nsim to overlay that many individual simulation replicates as thin lines (the background grid is dropped in this case so the replicates stand out).

PlotF() plots apical (max-over-age) fishing mortality; see what above for which of FDead()/FInteract()/FRetain is used, and relative for plotting F/FMSY instead.

PlotInteractions() plots Interactions() (total fish encountered, before discard mortality is applied), following the same byStock/ byFleet/AggregateYear behavior as PlotLandings()/PlotDiscards Unlike those two, it has no data counterpart – Interactions is not an observed quantity.

PlotVBiomass() plots VBiomass() – biomass weighted by the fraction of it actually exposed to each fleet's gear; see VBType above (it has no relative argument of its own). For the projection period of mse objects, this reflects any change to a fleet's selectivity, retention, or discard mortality made by the MP (see Advice()), not just the OM baseline.

For mse objects, each MP is drawn in its own color on the same panel by default (historical years are shown in a fixed neutral grey); set byMP = TRUE to facet by MP instead where there's a free facet dimension (see byMP above). The last historical year is duplicated into each MP's projection series so the line continues without a visual gap at the historical/projection boundary.

`PlotRemovals()` draws `Landings()` and `Discards()` together on the same panel(s) (see `Removals()` for the summed total): for hist objects they are distinguished by color; for mse objects color is used for MP and linetype distinguishes Landings (solid) from Discards (dashed). If `Discards()` is all zero (or essentially so) across the plotted selection, the Discards series is dropped and only Landings is shown. For data objects (see `object` above) they're summed into a single `Landings + Discards` series instead, since there's no color channel to spare.

See `PlotEffort()` for fishing effort, documented on its own page. See `units` above for plotting effort as real trip (or angler) counts instead of the raw index, where `TripsScalar`/`AnglerPerTrip` are available.

`byStock` and (for `PlotLandings()`/`PlotDiscards()`/`PlotRemovals()`) `byFleet` control faceting: when both are `TRUE` panels are arranged in a `Stock x Fleet` grid, matching the behavior of `Landings()`/`Discards()`. `byFleet = FALSE` sums fleets before plotting. `byStock` has a third state that `byFleet` doesn't: `byStock = FALSE` colors each stock as its own line instead of summing (see `byStock` above); `byStock = 'sum'` is what sums stocks before plotting. When faceting is used, `free_y` (default `TRUE`) lets each panel's y-axis scale independently, since stocks or fleets of very different magnitude (e.g. a large and a small stock) are otherwise hard to compare on a shared scale.

For `PlotSBiomass()` and `PlotSProduction()` with `byStock = 'sum'` and `byFemale = TRUE`, stocks within the same `OM@Complexes` group are not naively summed (which would double-count male and female components of the same population). Instead, within each complex, a stock whose name matches `"female"` (case-insensitive) is taken as that complex's contribution to the total. If a complex has more than one stock and no single stock can be unambiguously identified as female, faceting by stock is used instead and a message is printed. `byFemale` has no effect when `byStock = FALSE` (every stock, male and female, is shown colored).

`PlotBiomass()`, `PlotSBiomass()`, `PlotSProduction()`, and `PlotF()` can plot values relative to unfished or MSY reference points instead of absolute values; see `relative` and `type` above. For `PlotBiomass()`, `PlotSBiomass()`, and `PlotSProduction()`, `byStock = 'sum'` with `relative != 'none'` sums the numerator (e.g. `Biomass`) and denominator (e.g. `B0`/`BMSY`) across stocks separately before dividing, giving a true aggregate ratio (e.g. overall depletion) rather than summing the per-stock ratios themselves (which would not be meaningful).

`PlotDynamics()` arranges `PlotNumber()`, `PlotBiomass()`, `PlotSBiomass()`, and `PlotRemovals()` into a single, terse figure with `patchwork::wrap_plots()` – one panel per row/column of a `2 x 2` grid. `PlotSProduction()` and the separate `PlotLandings()`/`PlotDiscards()` panels are deliberately left out of this default overview (call them directly for that level of detail). Its own `byFleet` argument (and `plot()`'s, which dispatches to it) defaults to `FALSE`, unlike the individual `Plot*()` functions; each argument is only forwarded to the panels that accept it. `byStock` is forwarded as-is, including the `'sum'` option described above.

## Value

A `ggplot` object (a `patchwork` object for `PlotDynamics()`).

### See Also

Number(), Biomass(), SBiomass(), SProduction(), FDead(), FInteract(), FRetain(), VBiomass(), Landings(), Discards(), Interactions(), Removals(), Effort(), B_B0(), B_BMSY(), F_FMSY()

### Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
PlotBiomass(Hist)
PlotF(Hist, relative = 'FMSY')
PlotDynamics(Hist)

MSE <- Project(Hist, ExampleMPs())
PlotBiomass(MSE, byMP = TRUE)
PlotF(MSE, relative = 'FMSY')
PlotDynamics(MSE)

MHist <- Simulate(MultiStockOM)
PlotSBiomass(MHist, byStock = TRUE)
PlotLandings(MHist, byStock = TRUE, byFleet = TRUE)

# restrict to specific stocks, by name or index
PlotSBiomass(MHist, Stocks = 'Female')

## End(Not run)
```

---

## plot_schedule *Plot Biological and Gear Schedules*

---

### Description

Line plots of an age-based schedule (selectivity, retention, discard mortality, length, weight, maturity, natural mortality, fecundity) against age, or (via `x = "Length"`) against length instead. By default, each independent curve (the historical schedule, and each MP's projection schedule for mse objects) is drawn for its own last available year only; pass `Years` to see more.

### Arguments

| | |
|---|---|
| `object` | An om, hist, or mse object (`om` objects are populated via `PopulateOM()` if not already), or – for the `*Curve()` functions only – a stock/fleet object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate. |

`byStock`, `byFleet`
: One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock/fleet). These plots never sum or average a curve across stocks/fleets – that isn't a meaningful operation the way summing catch is – so every stock/fleet is always distinguishable somehow. `TRUE`/`NULL` facets by `Stock`/`Fleet`. `FALSE` instead colors by that variable;

`Years`
: Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed.

`units`
: Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.).

`Stocks`
: Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks).

`x`
: `"Age"` (default) or `"Length"`. `"Length"` projects the curve through the stock's `[Length()]` age-length key (`@ALK`) onto a length axis instead – e.g. `PlotWeightCurve(x = "Length")` gives the weight-length relationship, `PlotMaturityCurve(x = "Length")` gives maturity-at-length. Not meaningful for `PlotLengthCurve()` itself (see `PlotLength()`; length-at-length is a trivial identity, and errors if requested). The length axis is always labelled in the stock's native `Length@Units`, independent of `units`.

## Details

`PlotSelectivityCurve()`, `PlotRetentionCurve()`, and `PlotDiscardMortalityCurve()` plot a single fleet-level gear curve (`[Fleet()]`'s `Selectivity`/`Retention`/`DiscardMortality` sub-objects) on a chosen `x` axis. For the projection period of mse objects, these reflect any change made by the MP via `Advice()` (see `VBiomass()`, which resolves the same effective curves). Each has a combining counterpart – `PlotSelectivity()`, `PlotRetention()`, `PlotDiscardMortality()` – that arranges every plot available for that schedule into one figure.

`PlotLengthCurve()`, `PlotWeightCurve()`, `PlotMaturityCurve()`, `PlotNaturalMortalityCurve()`, and `PlotFecundityCurve()` plot a single stock-level biological curve on a chosen `x` axis. Each has a combining counterpart – `PlotLength()`, `PlotWeight()`, `PlotMaturity()`, `PlotNaturalMortality()`, `PlotFecundity()` – that arranges every plot available for that schedule (the age-based curve, the length-based curve when populated, and an age-size key where one exists) into one figure.

## Value

A `ggplot` object.

### See Also

Fleet(), Stock(), VBiomass(), PlotSelectivity(), PlotRetention(), PlotDiscardMortality(), PlotLength(), PlotWeight(), PlotMaturity(), PlotNaturalMortality(), PlotFecundity()

### Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
PlotSelectivityCurve(Hist)
PlotMaturityCurve(Hist)

MSE <- Project(Hist, ExampleMPs())
PlotRetentionCurve(MSE)
PlotWeightCurve(MSE, Sim = 3)

## End(Not run)
```

---

## PlotALK *Plot the Age-Length Key*

---

### Description

Heatmap of a stock's age-length key (`Length@ALK`; see `ALK()`); the probability of belonging to each length class given age.

### Usage

```
PlotALK(
  object,
  Sim = NULL,
  byStock = NULL,
  Stocks = NULL,
  Years = NULL,
  units = TRUE
)
```

### Arguments

| | |
|---|---|
| `object` | A stock/om/hist/mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` averages across all simulations. |
| `byStock` | Unused (present for signature consistency with the other `Plot*()` functions). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |

| | |
|---|---|
| `Years` | Optional numeric vector, or `"all"`. Default `NULL`: the last available calendar year only. `"all"` facets by every available year. |
| `units` | Logical. `TRUE` (default) labels the age/length axes with the stock's `Ages@Units`/`Length@Units` when set and agrees across every plotted stock. `FALSE` suppresses unit labeling. |

### Details

Only one calendar year is shown by default (the last available); pass `Years` for more. When `object` has more than one simulation, the probability shown at each age/length cell is the mean across simulations before that age's probabilities are rescaled (see below).

When `object` has more than one stock, each gets its own panel (`patchwork`-combined).

Returns `NULL` invisibly, with an informational message, when a stock's `Length@CVatAge` isn't set – `ALK` is only populated when it is (see `Length()`).

### Value

A `ggplot` object, or `NULL` invisibly if no stock has a populated `ALK`.

### See Also

`PlotLength()`, `ALK()`, `CalcAgeSizeKey()`, `Length()`

---

## `PlotAWK` *Plot the Age-Weight Key*

---

### Description

Heatmap of a stock's age-weight key (`Weight@AWK`; see `AWK()`); the probability of belonging to each weight class given age.

### Usage

```
PlotAWK(
  object,
  Sim = NULL,
  byStock = NULL,
  Stocks = NULL,
  Years = NULL,
  units = TRUE
)
```

### Arguments

| | |
|---|---|
| `object` | A stock/om/hist/mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` averages across all simulations. |
| `byStock` | Unused (present for signature consistency with the other `Plot*()` functions). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL`: the last available calendar year only. `"all"` facets by every available year. |
| `units` | Logical. `TRUE` (default) labels the age/weight axes with the stock's `Ages@Units`/`Weight@Units` when set and agrees across every plotted stock. `FALSE` suppresses unit labeling. |

### Details

Only one calendar year is shown by default (the last available); pass `Years` for more. When `object` has more than one simulation, the probability shown at each age/weight cell is the mean across simulations before that age's probabilities are rescaled (see below).

Color is rescaled within each age class (its own max -> darkest color, `0` -> white); no color legend is shown.

When `object` has more than one stock, each gets its own panel (`patchwork`-combined).

Returns `NULL` invisibly when a stock's `Weight@CVatAge` isn't set – `AWK` is only populated when it is (see `Weight()`).

### Value

A `ggplot` object, or `NULL` invisibly if no stock has a populated `AWK`.

### See Also

`PlotWeight()`, `AWK()`, `CalcAgeSizeKey()`, `Weight()`

---

## `PlotCatchability` *Plot Catchability*

---

### Description

Plots each fleet's catchability trend (`Fleet@Catchability@Efficiency`) over time. For a hist/mse `object` this already reflects any depletion-driven optimization applied to the historical period (see `Simulate()`) – there's no separate specified-vs-realized distinction the way there is for `Effort()`.

### Usage

```
PlotCatchability(
  object,
  Stock = NULL,
  byStock = NULL,
  byFleet = NULL,
  Stocks = NULL,
  Years = NULL,
  probs = c(0.05, 0.95),
  nsim = 0
)
```

### Arguments

| | |
|---|---|
| `object` | A fleet object, or an om, hist, or mse object. |
| `Stock` | A stock object supplying the biology needed to populate a bare fleet `object`. Ignored otherwise. Default `NULL` uses an example stock, with a message noting this. |
| `byStock`, `byFleet` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock/fleet). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` plots every available year. |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon. Default `c(0.05, 0.95)`. |
| `nsim` | Integer. Number of individual simulation trajectories to overlay as thin lines, in addition to the median/ribbon. Default `0`. |

### Value

A `ggplot` object.

### See Also

`Catchability()`, `PlotEffort()`, `Fleet()`

---

## `PlotClosure` *Plot Spatial Closures*

---

### Description

Heatmap of each fleet's spatio-temporal closure schedule (`Fleet@Closure`);the proportion of simulations in which an area is closed to fishing, by year. Areas that are open in every year for every simulation contribute nothing to look at, so a stock/fleet with no closures specified is silently omitted; if none has any, an informational message is printed and `NULL` is returned invisibly.

### Usage

```
PlotClosure(object, Stock = NULL, byStock = NULL, Stocks = NULL, Years = NULL)
```

### Arguments

| | |
|---|---|
| `object` | A fleet object, or an om, hist, or mse object. |
| `Stock` | A stock object supplying the biology needed to populate a bare fleet `object`. Ignored otherwise. Default `NULL` uses an example stock, with a message noting this. |
| `byStock` | Logical. Facet by stock? Default `NULL` facets automatically when `object` has more than one stock. |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` plots every available year. |

### Value

A `ggplot` object, or `NULL` invisibly if no selected stock/fleet has any closures specified.

### See Also

`Fleet()`

---

## `PlotCompShift` *Validate* `Shift` *for Composition Data*

---

### Description

Diagnostic plot comparing three catch-at-age or catch-at-size proportions-at-bin curves: the true (OM-predicted) composition, the composition expected once `CompObs()`'s `Shift` slot is applied to the true composition, and the simulated observed composition actually drawn by the Dirichlet-Multinomial sampler. Useful for confirming that a configured `Shift` tilts the sampled composition in the expected direction (e.g. towards larger sizes or older ages) before running a full MSE.

### Usage

```
PlotCompShift(
  Hist,
  sim = 1,
  i = 1,
  type = c("LandingsAtSize", "DiscardsAtSize", "LandingsAtAge", "DiscardsAtAge"),
  fleets = NULL,
  years = NULL,
  nYears = 4
)
```

### Arguments

| | |
|---|---|
| `Hist` | A hist object, i.e. the output of `Simulate()`. |
| `sim` | Integer. Simulation replicate to plot. |
| `i` | Integer or character. Stock complex index or name (as in `names(Hist@OM@Complexes)`). Default `1`. |
| `type` | Character. One of `"LandingsAtSize"`, `"DiscardsAtSize"`, `"LandingsAtAge"`, or `"DiscardsAtAge"`. |
| `fleets` | Character vector of fleet names to include, or `NULL` (default) to include all fleets with non-empty `Shift`. |
| `years` | Numeric vector of calendar years to include, or `NULL` (default) to use the last `nYears` historical years. |
| `nYears` | Integer. Number of most recent historical years to plot when `years = NULL`. Default `4`. Ignored if `years` is supplied. |

### Details

The true composition q is `Hist@LandingsAtSize`/`@LandingsAtAge` (or the `Discards` equivalent), summed over stocks and areas within complex `i`, and normalised to proportions within each fleet and year.

The expected shifted composition is computed directly from `Shift` (`Hist@OM@Obs[[i]][[fleet]]@<type>@Shift`) as $q'_b = q_b \exp(\text{Shift}_b) / \sum_b q_b \exp(\text{Shift}_b)$. This is the mean of the Dirichlet concentration vector used to generate the observed draw (`ESS` and `Theta` scale the concentration but cancel out of its mean, so they don't affect this curve).

The observed composition is the corresponding slot of `Hist@Data[[sim]][[i]]`, i.e. the actual noisy Dirichlet-Multinomial draw, also normalised to proportions. All three series are drawn from the same replicate `sim`.

### Value

A `ggplot` object (invisibly plotted), faceted by `Fleet` and `Year`, with the true, expected-shifted, and observed proportions overlaid as lines.

---

## `PlotDepletion` *Plot Initial/Final Depletion*

---

### Description

Histogram of each stock's sampled depletion values (relative to `Depletion@Reference`, e.g. `"B0"` or `"SB0"`) across simulation replicates; the `Initial`/`Final` slots of `[Depletion()]`. Both are shown, side by side, when a stock has both populated.

### Usage

```
PlotDepletion(object, byStock = NULL, Stocks = NULL, bins = 15, tol = 0.1)
```

### Arguments

| | |
|---|---|
| `object` | stock, om, hist, or mse object. |
| `byStock` | Logical. Facet by stock? Default `NULL` facets automatically when `object` has more than one stock. |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `bins` | Integer. Number of histogram bins. Default `15`. |
| `tol` | Numeric. Relative tolerance for flagging a simulation as having missed its `Depletion@Initial/Final` target – a simulation is flagged when `abs(log(achieved / target)) > log(1 + tol)`. Default `0.1` (10%). Ignored for stock/om `object` (nothing yet simulated to check against). |

### Details

For a hist or mse `object`, a second panel is added below: the actual historical relative-biomass trajectory (`[B_B0()]` or `[SB_SB0()]`, matching `Depletion@Reference`) for every simulation, so the sampled targets above can be checked against what was actually achieved. Where `Depletion@Initial/Final` are specified, the target value for each simulation is overlaid as a point on its own trajectory at the first/last historical year; a simulation whose achieved value misses its target by more than `tol` (relative) is colored red rather than grey, and listed in a warning (via `cli::cli_warn()`). This can legitimately happen – e.g. a low target depletion combined with selectivity maturing well before maturity, or a `Final` target above the unfished trajectory (`Unfished@Dynamic`) after a run of poor recruitment, which no catchability can reach; the warning lists these separately. `Simulate()` performs the same check itself, recording any misses to `Hist@Log` (see `Log()`) rather than printing them.

### Value

A `ggplot` object, or (for hist/mse `object`) a `patchwork` object combining the histogram and trajectory panels.

### See Also

`Depletion()`, `B_B0()`, `SB_SB0()`, `PlotSRR()`, `Stock()`

---

## `PlotDiscardMortality` *Plot the Discard Mortality Schedule*

---

### Description

`PlotDiscardMortality()` arranges every plot related to a fleet's discardmortality object – the age-based curve (`PlotDiscardMortalityCurve()`) and the length-based curve (when `DiscardMortality@MeanAtLength` is populated) – into a single figure with `patchwork::wrap_plots()`.

### Usage

```
PlotDiscardMortality(
  object,
  Sim = NULL,
  byStock = NULL,
  byFleet = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  Stock = NULL,
  probs = c(0.05, 0.95)
)

PlotDiscardMortalityCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  byFleet = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  Stock = NULL,
  x = c("Age", "Length"),
  probs = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `object` | A fleet object, or an om, hist, or mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `byStock`, `byFleet` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock/fleet). These plots never sum or average a curve across stocks/fleets – that isn't a meaningful operation the way summing catch is – so every stock/fleet is always distinguishable somehow. `TRUE`/`NULL` facets by `Stock`/`Fleet`. `FALSE` instead colors by that variable; |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed. |
| `units` | Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string |

(e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.).

`Stocks` Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks).

`Stock` A stock object supplying the biology needed to populate a bare fleet `object`. Ignored otherwise. Default `NULL` uses an example stock, with a message noting this.

`probs` Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate.

`x` `"Age"` (default) or `"Length"`. `"Length"` projects the curve through the stock's `[Length()]` age-length key (`@ALK`) onto a length axis instead – e.g. `PlotWeightCurve(x = "Length")` gives the weight-length relationship, `PlotMaturityCurve(x = "Length")` gives maturity-at-length. Not meaningful for `PlotLengthCurve()` itself (see `PlotLength()`; length-at-length is a trivial identity, and errors if requested). The length axis is always labelled in the stock's native `Length@Units`, independent of `units`.

## Value

`PlotDiscardMortalityCurve()` returns a `ggplot` object; `PlotDiscardMortality()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

## See Also

plot_schedule

---

# PlotEffort *Plot Fishing Effort*

---

## Description

For a bare fleet or om `object` (nothing simulated yet), `PlotEffort()` plots the specified input effort trend (`PlotEffortCurve()`, `Fleet@Effort@Effort`, historical years only). For a hist/mse `object`, it plots the realized historical/projection effort trajectory.

## Usage

```
PlotEffort(
  object,
  Stock = NULL,
  units = NULL,
  byStock = NULL,
  byFleet = NULL,
  Stocks = NULL,
```

```
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL,
  IncHist = TRUE,
  byMP = FALSE,
  AggregateYear = FALSE
)

PlotEffortCurve(
  object,
  Stock = NULL,
  units = NULL,
  byStock = NULL,
  byFleet = NULL,
  Stocks = NULL,
  probs = c(0.05, 0.95),
  nsim = 0,
  Years = NULL,
  free_y = NULL
)
```

## Arguments

| | |
|---|---|
| `object` | A fleet object, an om, hist, or mse object, or a data object (plots `Effort(object)` directly). |
| `Stock` | A stock object supplying the biology needed to populate a bare fleet `object`. Ignored otherwise. Default `NULL` uses an example stock, with a message noting this. |
| `units` | Character or `NULL`. `NULL` (default) uses the coarsest unit common to every plotted fleet (`"Effort"`, `"Trips"`, or `"Anglers"`, depending on what each fleet's `Effort()` object supplies). Set explicitly to force one; requesting a unit some fleet lacks the data for is an error. |
| `byStock`, `byFleet` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock/fleet). `FALSE` sums effort across fleets instead of faceting/coloring by fleet (only when every summed fleet shares the same `units`). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon. Default `c(0.05, 0.95)`. |
| `nsim` | Integer. Number of individual simulation trajectories to overlay as thin lines, in addition to the median/ribbon. Default `0`. |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` plots every available year. |

`free_y` Logical. Free the y-axis scale across facets? Default `NULL` resolves to `TRUE`.

`IncHist`, `byMP`, `AggregateYear`
Used only when `object` is a hist/mse object; see plot_hist.

### Value

`PlotEffortCurve()` returns a `ggplot` object; `PlotEffort()` returns a `ggplot` object of realised effort when `object` is a hist/mse object, otherwise the `PlotEffortCurve()` plot.

### See Also

`Effort()`, `PlotCatchability()`, `Fleet()`

---

## PlotFecundity *Plot the Fecundity-at-Age Schedule*

---

### Description

`PlotFecundity()` arranges every plot related to a stock's fecundity object – the mean fecundity-at-age curve (`PlotFecundityCurve()`) and the fecundity-at-length curve (when `Fecundity@MeanAtLength` is populated) – into a single figure with `patchwork::wrap_plots()`.

### Usage

```
PlotFecundity(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  probs = c(0.05, 0.95)
)

PlotFecundityCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  x = c("Age", "Length"),
  probs = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `object` | An om, hist, mse, or stock object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed. |
| `units` | Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate. |
| `x` | `PlotFecundityCurve()` only; see plot_schedule. |

### Value

`PlotFecundityCurve()` returns a `ggplot` object; `PlotFecundity()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

### See Also

plot_schedule

---

## `PlotFleet` *Plot All Fishery Schedules for a Fleet*

---

### Description

Runs every fleet-related `Plot*()` function against a fleet object: `PlotSelectivity()`, `PlotRetention()`, `PlotDiscardMortality()`, `PlotCatchability()`, and `PlotClosure()` (omitted when no closure is specified).

### Usage

```
PlotFleet(
  Fleet,
  Stock = NULL,
  byStock = NULL,
  byFleet = NULL,
  Stocks = NULL,
  Years = NULL,
  probs = c(0.05, 0.95),
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `Fleet` | A fleet object, or an om, hist, or mse object. |
| `Stock` | A stock object supplying the biology needed to populate a bare fleet `Fleet`. Ignored otherwise. Default `NULL` uses an example stock, with a message noting this. |
| `byStock` | Logical. Facet by stock? Passed to each `Plot*()` function. Default `NULL` facets automatically when `Fleet` has more than one stock. |
| `byFleet` | Logical. Facet by fleet? Passed to each `Plot*()` function. Default `NULL` facets automatically when `Fleet` has more than one fleet. |
| `Stocks` | Character or numeric vector. Restrict the plots to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` plots every available year. |
| `probs` | Numeric vector of length 2. Lower/upper quantiles for simulation envelopes. Default `c(0.05, 0.95)`. |
| `silent` | Logical. If `TRUE`, the returned list doesn't announce what it contains when printed at the console (see `print.PlotFleetList()`). Default `FALSE`. |

### Value

A named list of `ggplot`/`patchwork` objects: `Selectivity`, `Retention`, `DiscardMortality`, `Effort`, `Catchability`, `Closure` (omitted when no closure is specified).

### See Also

`Fleet()`, `PlotSelectivity()`, `PlotRetention()`, `PlotDiscardMortality()`, `PlotEffort()`, `PlotCatchability()`, `PlotClosure()`

---

`plotFun` *Print out plotting functions*

---

## Description

This function prints out the available plotting functions for objects of class MSE or Data

## Usage

```
plotFun(class = c("MSE", "Data"), msg = TRUE)
```

## Arguments

| | |
|---|---|
| `class` | Character string. Prints out the plotting functions for objects of this class. |
| `msg` | Logical. Should the functions be printed to screen? |

## Note

Basically the function looks for any functions in the MSEtool that have the word `plot` in them. There is a chance that some plotting functions are missed. Let us know if you find any and we will add them.

## Author(s)

A. Hordyk

---

`PlotIndexFit` *Plot Observed/Simulated Index Against the Underlying Population Trend*

---

## Description

Overlays the `CPUE`/`Survey` index, on its own natural scale, against the underlying population trend scaled to match it. Diagnostic for how well an index tracks the simulated dynamics (hyperstability, error, autocorrelation). See `IndexFitTable()` for the matching parameter summary.

## Usage

```
PlotIndexFit(
  object,
  type = c("CPUE", "Survey"),
  Fleets = NULL,
  Stocks = NULL,
  probs = c(0.05, 0.95),
```

```
  nsim = 1,
  ribbon = FALSE,
  Years = NULL,
  IncHist = TRUE,
  free_y = TRUE
)
```

### Arguments

| | |
|---|---|
| `object` | A hist or mse object. |
| `type` | Character. One of `"CPUE"` or `"Survey"`. |
| `Fleets` | Character vector. Restrict to these fleet names. Default `NULL` (all fleets with a populated `type` index). |
| `Stocks` | Character vector. Restrict to these stock/complex names (matching `names(object@OM@Complexe` or `StockNames()` when `Complexes` is unset). Default `NULL` (all). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the shaded ribbon across simulations. Only used when `ribbon = TRUE`. Default `c(0.05, 0.95)`. |
| `nsim` | Integer. Number of individual simulation replicates to plot. Default `1`. Sims are chosen to span the range of overall trajectory outcomes (see Details), not just the first `nsim` sim IDs. When `nsim > 1`, panels are additionally faceted by `Sim` (see Details) – keep this small. Set to `0` to show only the median/ribbon summary (forces `ribbon = TRUE`). |
| `ribbon` | Logical. Show the across-simulation median and `probs`-quantile ribbon instead of (or, if `nsim > 0`, in addition to) individual simulation lines. Default `FALSE`. See Details for why individual lines are the default. |
| `Years` | Optional numeric vector. Subset the time series to these years before plotting. |
| `IncHist` | Logical. For mse objects, include the historical period? Default `TRUE`. No effect for hist objects. |
| `free_y` | Logical. Let each facet panel's y-axis scale independently. Default `TRUE`. |

### Details

The `Index` series is plotted on its natural scale. The `True` (OM) series is put on that same scale, per simulation, via `Efficiency * NomIndex^Beta`.

Panels are faceted by `Fleet` (and `Stock` when more than one stock/complex is plotted). For a single displayed simulation (`nsim <= 1`) and an mse object with more than one MP, an `MP` facet column is added, with the historical period repeated in every MP's panel so each line is unbroken across the historical/projection boundary.

### Value

A `ggplot` object, or `NULL` invisibly if no populated `type` index is found.

### See Also

`IndexFitTable()`, `PlotCPUE()`, `PlotSurvey()`, `Biomass()`, `Number()`

---

`PlotLength` *Plot the Length-at-Age Schedule*

---

### Description

`PlotLength()` arranges every plot related to a stock's length object – the mean length-at-age curve (`PlotLengthCurve()`) and the age-length key (`PlotALK()`) – into a single figure with `patchwork::wrap_plots()`. The `ALK` panel is omitted when the stock's `Length@CVatAge` isn't set (so `ALK` was not populated).

### Usage

```
PlotLength(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  probs = c(0.05, 0.95)
)

PlotLengthCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  x = c("Age", "Length"),
  probs = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `object` | An om, hist, mse, or stock object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an |

identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed.

`units` Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.).

`Stocks` Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks).

`probs` Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate.

`x` `PlotLengthCurve()` only. Always `"Age"`; `"Length"` is not meaningful (length-at-length is a trivial identity) and errors if requested.

### Value

`PlotLengthCurve()` returns a `ggplot` object; `PlotLength()` returns a `patchwork` object.

### See Also

`PlotALK()`, plot_schedule

---

## PlotMaturity *Plot the Maturity-at-Age Schedule*

---

### Description

`PlotMaturity()` arranges every plot related to a stock's maturity object – the mean maturity-at-age curve (`PlotMaturityCurve()`), the maturity-at-length curve (when `Maturity@MeanAtLength` is populated), and the maturity-at-weight curve (when `Maturity@MeanAtWeight` is populated) – into a single figure with `patchwork::wrap_plots()`.

### Usage

```
PlotMaturity(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  probs = c(0.05, 0.95)
)
```

```
PlotMaturityCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  x = c("Age", "Length", "Weight"),
  probs = c(0.05, 0.95)
)
```

## Arguments

| | |
|---|---|
| `object` | An om, hist, mse, or stock object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed. |
| `units` | Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate. |
| `x` | `PlotMaturityCurve()` only. `"Age"` (default), `"Length"`, or `"Weight"` – `"Weight"` projects the curve through the stock's `[Weight()]` age-weight key (`@AWK`) onto a weight axis, and requires `Weight@CVatAge` to be set. |

## Value

`PlotMaturityCurve()` returns a `ggplot` object; `PlotMaturity()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

## See Also

plot_schedule

## plotmulti *A basic SSB plot for debugging runMSE output*

### Description

A basic SSB plot for debugging runMSE output

### Usage

```
plotmulti(MSEmulti, maxsim = 8)
```

### Arguments

| | |
|---|---|
| MSEmulti | An object of class MMSE arising from a run of multiMSE(MOM, ...) |
| maxsim | Integer. The number of simulations to plot |

### Author(s)

T.Carruthers

## PlotNaturalMortality *Plot the Natural Mortality-at-Age Schedule*

### Description

`PlotNaturalMortality()` arranges every plot related to a stock's naturalmortality object – the mean natural-mortality-at-age curve (`PlotNaturalMortalityCurve()`) and the natural-mortality-at-length curve (when `NaturalMortality@MeanAtLength` is populated) – into a single figure with `patchwork::wrap_plots()`.

### Usage

```
PlotNaturalMortality(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  probs = c(0.05, 0.95)
)

PlotNaturalMortalityCurve(
  object,
  Sim = NULL,
  byStock = NULL,
```

```
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  x = c("Age", "Length"),
  probs = c(0.05, 0.95)
)
```

## Arguments

| | |
|---|---|
| `object` | An om, hist, mse, or stock object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed. |
| `units` | Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate. |
| `x` | `PlotNaturalMortalityCurve()` only; see plot_schedule. |

## Value

`PlotNaturalMortalityCurve()` returns a `ggplot` object; `PlotNaturalMortality()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

## See Also

plot_schedule

## plotOFL *A generic OFL plot for NOAA use*

### Description

As title.

### Usage

```
plotOFL(Data, xlims = NA, perc = 0.5)
```

### Arguments

| | |
|---|---|
| Data | An object of class Data that has been run though TAC() |
| xlims | x axis limits |
| perc | The percentile of the OFL distribution to be plotted |

### Value

A table of performance metrics.

### Author(s)

T. Carruthers

## plotquant *A fairly tidy time-series quantile plot*

### Description

A fairly tidy time-series quantile plot

### Usage

```
plotquant(
  x,
  p = c(0.05, 0.25, 0.75, 0.95),
  yrs,
  qcol,
  lcol,
  addline = T,
  ablines = NA
)
```

### Arguments

| | |
|---|---|
| `x` | Matrix. A time series quantity `[simulation, year]` |
| `p` | Numeric vector. The percentiles that are plotted (LB2,LB1,UB1,UB2). LB2 and UB2 are the outer percentiles, LB1 and UB1 are the inner percentiles. |
| `yrs` | Numeric vector. The years corresponding to the indexing of x |
| `qcol` | Character, color. The color of the inner percentile range |
| `lcol` | Character, color. The color of the outer percentile range. |
| `addline` | Logical. Should two individual simulations be added to the percentile plots? |
| `ablines` | Numeric vector. Horizontal lines to be added to the plot. |

### Author(s)

T.Carruthers

---

## `PlotRecDevs` *Plot Recruitment Deviations*

---

### Description

Plots a stock's recruitment deviations (`RecDevInit`/`RecDevHist`/`RecDevProj`; see srr)

### Usage

```
PlotRecDevs(
  object,
  Sim = NULL,
  byStock = NULL,
  Stocks = NULL,
  probs = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `object` | A stock/om/hist/mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations and adds a `probs` quantile ribbon (unless `nSim == 1`). |
| `byStock` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. |

### Details

`PlotRecDevs()` plots the full log-recruitment-deviation timeseries as one continuous series per stock. When `object` has more than one simulation, the median is drawn with a `probs` quantile ribbon. A dashed vertical line marks the historical/projection boundary (the first projection year); a dotted vertical line marks the initial-age-structure/historical boundary, when initial-age deviations are present.

### Value

A `ggplot` object.

### See Also

`PlotSRR()`, `PlotSRRCurve()`, `GenRecDevs()`, srr

---

## plotRel *Plot a relationship between stocks*

---

### Description

Plot a relationship between stocks

### Usage

```
plotRel(Stocks, Rel, Relno, Snams, leg = F, extras = 0)
```

### Arguments

| | |
|---|---|
| `Stocks` | A list of stock objects (MOM@Stocks) |
| `Rel` | A list of inter-stock MICE relationships (MOM@Rel) |
| `Relno` | Integer. The relationship you wish to plot |
| `Snams` | A vector of stock names |
| `leg` | Logical. Do you want to plot a legend? |
| `extras` | Integer. The number of blank plots to create at the end. |

### Author(s)

T.Carruthers

`PlotRetention` *Plot the Retention Schedule*

## Description

`PlotRetention()` arranges every plot related to a fleet's retention object – the age-based curve (`PlotRetentionCurve()`), the length-based curve (when `Retention@MeanAtLength` is populated), and the weight-based curve (when `Retention@MeanAtWeight` is populated) – into a single figure with `patchwork::wrap_plots()`.

## Usage

```
PlotRetention(
  object,
  Sim = NULL,
  byStock = NULL,
  byFleet = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  Stock = NULL,
  probs = c(0.05, 0.95)
)

PlotRetentionCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  byFleet = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  Stock = NULL,
  x = c("Age", "Length", "Weight"),
  probs = c(0.05, 0.95)
)
```

## Arguments

| | |
|---|---|
| `object` | A fleet object, or an om, hist, or mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `byStock`, `byFleet` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock/fleet). These plots never sum or average a curve |

across stocks/fleets – that isn't a meaningful operation the way summing catch is – so every stock/fleet is always distinguishable somehow. `TRUE`/`NULL` facets by `Stock`/`Fleet`. `FALSE` instead colors by that variable;

`Years` Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed.

`units` Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.).

`Stocks` Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks).

`Stock` A stock object supplying the biology needed to populate a bare fleet `object`. Ignored otherwise. Default `NULL` uses an example stock, with a message noting this.

`probs` Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate.

`x` `PlotRetentionCurve()` only; see `PlotSelectivityCurve()`.

## Value

`PlotRetentionCurve()` returns a `ggplot` object; `PlotRetention()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

## See Also

plot_schedule

---

`PlotSelectivity` *Plot the Selectivity Schedule*

---

## Description

`PlotSelectivity()` arranges every plot related to a fleet's selectivity object – the age-based curve (`PlotSelectivityCurve()`), the length-based curve (when `Selectivity@MeanAtLength` is populated), and the weight-based curve (when `Selectivity@MeanAtWeight` is populated) – into a single figure with `patchwork::wrap_plots()`.

### Usage

```
PlotSelectivity(
  object,
  Sim = NULL,
  byStock = NULL,
  byFleet = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  Stock = NULL,
  probs = c(0.05, 0.95)
)

PlotSelectivityCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  byFleet = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  Stock = NULL,
  x = c("Age", "Length", "Weight"),
  probs = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `object` | A fleet object, or an om, hist, or mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `byStock`, `byFleet` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock/fleet). These plots never sum or average a curve across stocks/fleets – that isn't a meaningful operation the way summing catch is – so every stock/fleet is always distinguishable somehow. `TRUE`/`NULL` facets by `Stock`/`Fleet`. `FALSE` instead colors by that variable; |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed. |
| `units` | Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string |

(e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.).

`Stocks` Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks).

`Stock` A stock object supplying the biology needed to populate a bare fleet `object` (`Length`/`Weight` for projecting onto those axes, etc.). Ignored otherwise. Default `NULL` uses an example stock, with a message noting this.

`probs` Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate.

`x` `PlotSelectivityCurve()` only; see plot_schedule. Also accepts `"Weight"`, projecting the curve through the stock's `[Weight()]` age-weight key (`@AWK`) onto a weight axis; requires `Weight@CVatAge` to be set.

### Value

`PlotSelectivityCurve()` returns a `ggplot` object; `PlotSelectivity()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

### See Also

plot_schedule

---

## `PlotSpatial` *Plot Spatial Structure*

---

### Description

Visualizes the spatial structure of one or more stocks: the area-to-area movement probability matrix (as a heatmap) and the equilibrium (unfished) distribution of biomass across areas (as a bar chart), side by side.

### Usage

```
PlotSpatial(object, Stocks = NULL, byStock = NULL, Age = NULL, Year = NULL)
```

### Arguments

`object` A stock, om, hist, or mse object.

`Stocks` Character or numeric vector restricting the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` includes every stock in `object`.

`byStock` Logical. Facet the panels by stock? Default `NULL` facets automatically when more than one selected stock has more than one area.

| | |
|---|---|
| `Age` | The age class at which to evaluate `Movement`/`UnfishedDist`, matched against the age dimension of those arrays. Default `NULL` uses each stock's first age class. |
| `Year` | The calendar year at which to evaluate `Movement`/ `UnfishedDist`, matched against the year dimension of those arrays. Default `NULL` uses the last historical year. |

### Details

Both panels are evaluated at a single age class and calendar year, since the underlying `Movement` and `UnfishedDist` arrays can vary by age and year. Simulation replicates are averaged. Stocks with only one area have no spatial structure to show and are skipped with a message.

### Value

A `patchwork` object combining the movement-matrix heatmap and the equilibrium area-distribution bar chart. Invisibly returns `NULL` (with a message) if no selected stock has more than one area.

### See Also

`Movement()`, `UnfishedDist()`, `Populate()`

### Examples

```
## Not run:
Hist <- Simulate(SeasonalSpatialOM)
PlotSpatial(Hist)
PlotSpatial(Hist, Stocks = 1, Age = 5)

## End(Not run)
```

---

## `PlotSRR` *Plot the Stock-Recruit Relationship*

---

### Description

`PlotSRR()` plots stock-recruit curve (`PlotSRRCurve()`) and the recruitment deviations (`PlotRecDevs()`.

### Usage

```
PlotSRR(
  object,
  Sim = NULL,
  byStock = NULL,
```

```
  Stocks = NULL,
  nPoints = 50,
  probs = c(0.05, 0.95)
)

PlotSRRCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  Stocks = NULL,
  nPoints = 50,
  probs = c(0.05, 0.95)
)
```

## Arguments

| | |
|---|---|
| `object` | A stock/om/hist/mse object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations and adds a `probs` quantile ribbon (unless `nSim == 1`). |
| `byStock` | One of `TRUE`, `FALSE`, or `NULL` (default, facets automatically when `object` has more than one stock). Ignored for a bare `Stock` (always a single curve). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). Ignored for a bare `Stock`. |
| `nPoints` | Integer. Number of spawning-production values to evaluate the curve at. Default `50`. |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. |

## Details

`PlotSRRCurve()` plots recruitment against spawning production, evaluating each stock's `SRR` model (`[SRR()]`) at the last available calendar year. When `object` has more than one simulation, the median curve is drawn with a `probs` quantile ribbon.

A bare stock `object` plots relative recruitment (`R/R0`) against relative spawning production using the stock's SRR model function (`SRR@Model`, e.g. `BevertonHolt()`) evaluated at `S0 = R0 = 1`.

An om, hist, or mse `object` instead plots the absolute curve (real recruitment against real spawning production, using `SP0` and `R0`).

## Value

`PlotSRRCurve()` returns a `ggplot` object; `PlotSRR()` returns a `patchwork` object.

### See Also

SRR(), PlotRecDevs, PlotDepletion(), Stock()

---

PlotStock *Plot All Biological Schedules for a Stock*

---

### Description

Runs every stock-related Plot*() function against a bare stock object: PlotLength(), PlotWeight(), PlotMaturity(), PlotNaturalMortality(), PlotFecundity() (each combining the age-based curve with any additional plot available for that schedule – a length-based curve, an age-size key – see their own docs), PlotSpatial() (when the stock has more than one area), PlotDepletion(), and PlotSRR().

### Usage

```
PlotStock(
  Stock,
  Stocks = NULL,
  nYear = NULL,
  pYear = NULL,
  CurrentYear = NULL,
  nSim = 5,
  Seasons = 1,
  ALK = TRUE,
  AWK = TRUE,
  seed = 102,
  force = FALSE,
  CalcAtLength = FALSE,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| Stock | A Stock() object (populated or not), or an om, hist, or mse object – in which case the plots are built from the single stock it resolves to (see Stocks), already populated as part of that object. |
| Stocks | Character or numeric vector. Selects which stock to plot when Stock is an om/hist/mse object with more than one, either by name (matching StockNames()) or by index; must resolve to exactly one stock. Ignored for a bare Stock() object. |
| nYear, pYear, CurrentYear, nSim, Seasons, ALK, AWK, seed, force, CalcAtLength | Passed to PopulateStock(); same arguments, same defaults, except nYear/pYear also accept NULL in which case defaults nYear = 20 and pYear = 0 are used. An already-populated bare Stock is plotted as-is (with whatever nYear/pYear/nSim it already has) unless nYear, pYear, CurrentYear, or force is explicitly set, in which case it's re-populated with these values. |

`silent` Logical. If `TRUE`, the returned list doesn't announce what it contains when printed at the console (see `print.PlotStockList()`). Default `FALSE`.

### Value

A named list of `ggplot`/`patchwork` objects: `Length`, `Weight`, `Maturity`, `NaturalMortality`, `Fecundity` (each a `patchwork` when more than one plot is available for that schedule, otherwise a plain `ggplot`), `Spatial` (omitted for a single-area stock), `Depletion`, `SRR`.

### See Also

`Stock()`, `PopulateStock()`, `PlotSRR()`, `PlotALK()`, `PlotAWK()`, `PlotDepletion()`

---

## PlotWeight *Plot the Weight-at-Age Schedule*

---

### Description

`PlotWeight()` arranges every plot related to a stock's weight object – the mean weight-at-age curve (`PlotWeightCurve()`), the weight-at-length curve (when `Weight@MeanAtLength` is populated), and the age-weight key (`PlotAWK()`, when `Weight@CVatAge` is set) – into a single figure with `patchwork::wrap_plots()`.

### Usage

```
PlotWeight(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  probs = c(0.05, 0.95)
)

PlotWeightCurve(
  object,
  Sim = NULL,
  byStock = NULL,
  Years = NULL,
  units = TRUE,
  Stocks = NULL,
  x = c("Age", "Length"),
  probs = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `object` | An om, hist, mse, or stock object. |
| `Sim` | Integer or `NULL` (default). Which simulation replicate to plot. `NULL` takes the median across all simulations, cell by cell, and adds a `probs` quantile ribbon behind it (unless `nSim == 1` or values constant across simulations. |
| `Years` | Optional numeric vector, or `"all"`. Default `NULL` shows each independent curve's own last calendar year, labelled by year in the legend (legend dropped if the curve is constant over that year). `"all"` shows every available year. Both `NULL` and `"all"` collapse consecutive years with an identical curve to just the breakpoint years. A numeric vector shows exactly those years, uncompressed. |
| `units` | Logical or a character unit string. `TRUE` (default) labels axes with the relevant `Units` slot (e.g. `Ages@Units`; also `Length@Units`/`Weight@Units` for `PlotLengthCurve()`/ `PlotWeightCurve()`). `FALSE` suppresses unit labelling. For `PlotLengthCurve()`/`PlotWeightCurve()` only, a unit string (e.g. `"cm"`, `"kg"`) relabels *and rescales* the y-axis into that unit; an unconvertible unit is an error. No effect beyond the x-axis label for the unitless-proportion plots (`PlotMaturityCurve()`, `PlotSelectivityCurve()`, etc.). |
| `Stocks` | Character or numeric vector. Restrict the plot to specific stocks, either by name (matching `StockNames()`) or by index. Default `NULL` (all stocks). |
| `probs` | Numeric vector of length 2. Lower and upper quantiles of the across-simulation ribbon drawn when `Sim = NULL`. Default `c(0.05, 0.95)`. Ignored when `Sim` is a specific replicate. |
| `x` | `PlotWeightCurve()` only; see plot_schedule. |

### Value

`PlotWeightCurve()` returns a `ggplot` object; `PlotWeight()` returns a `patchwork` object (or a plain `ggplot` when only the age-based curve is available).

### See Also

`PlotAWK()`, plot_schedule

---

## PM *Performance Metrics*

---

### Description

`PM_*` functions evaluate a performance metric against an mse object, or a `list` of mse objects (combined via `CombineMSE()` before calculation, so simulations from every list element are pooled and treated as one analysis). See PM-equations for the mathematical definition of each metric.

**Usage**

```
PM_FFMSY(object, Ref = 1, Years = NULL, silent = TRUE)

PM_SBSBMSY(object, Ref = 1, Years = NULL, silent = TRUE)

PM_SPSPMSY(object, Ref = 1, Years = NULL, silent = TRUE)

PM_Kobe(
  object,
  Quadrant = c("green", "red", "yellow", "orange"),
  Definition = c("SBiomass", "SProduction"),
  Years = NULL,
  silent = TRUE
)

PM_Status(
  object,
  Definition = c("SBiomass", "SProduction"),
  Years = NULL,
  silent = TRUE
)

PM_Red(
  object,
  Definition = c("SBiomass", "SProduction"),
  Years = NULL,
  silent = TRUE
)

PM_MinStatus(
  object,
  Reference = c("MSY", "Unfished"),
  Definition = c("SBiomass", "SProduction"),
  Years = NULL,
  silent = TRUE
)

PM_SBSBlim(object, Lim, Years = NULL, silent = TRUE)

PM_SPSPlim(object, Lim, Years = NULL, silent = TRUE)

PM_Safety(
  object,
  Lim,
  Definition = c("SBiomass", "SProduction"),
  Years = NULL,
  silent = TRUE
)
```

```
PM_Rebuild(object, Year, Target = 1, silent = TRUE)

PM_Removals(object, Years = NULL, Stocks = NULL, silent = TRUE)

PM_Landings(object, Years = NULL, Stocks = NULL, silent = TRUE)

PM_Yield(
  object,
  Years = NULL,
  Stocks = NULL,
  Type = c("Removals", "Landings"),
  silent = TRUE
)

PM_LogYield(
  object,
  Years = NULL,
  Stocks = NULL,
  Type = c("Landings", "Removals"),
  Floor = 0.001,
  silent = TRUE
)

PM_RelYield(object, Ref = NULL, Years = NULL, Stocks = NULL, silent = TRUE)

PM_AAVY(
  object,
  Type = c("TAC", "Removals", "Landings"),
  Years = NULL,
  Stocks = NULL,
  IncludeFirst = FALSE,
  silent = TRUE
)

PM_AAVE(object, Years = NULL, Fleets = NULL, silent = TRUE)

PM_Stability(
  object,
  Threshold,
  Type = c("TAC", "Removals", "Landings"),
  Years = NULL,
  Stocks = NULL,
  IncludeFirst = FALSE,
  silent = TRUE
)

PM_TACLimited(
```

```
  object,
  DeltaUp = NULL,
  DeltaDown = NULL,
  IncludeFirst = TRUE,
  Years = NULL,
  Stocks = NULL,
  tol = 1e-06,
  silent = TRUE
)
```

## Arguments

| | |
|---|---|
| `object` | An mse object, or a `list` of mse objects. |
| `Ref` | Numeric, or `NULL`. Reference/threshold value for `PM_FFMSY`/ `PM_SBSBMSY`/`PM_SPSPMSY`'s probability metric (default `1`). Pass `Ref = NULL` instead to get the mean projected ratio itself (`Stat` averaged across simulations, on its natural scale) rather than a probability against a threshold – `Prob` is left `NA` in that case, consistent with other natural-scale metrics such as `PM_Yield()`. |
| `Years` | Numeric vector of projection years to evaluate over. Default `NULL` uses all projection years in which the MP was active, i.e. excluding any "interim" years before `OM@MPStartYear` (see om). |
| `silent` | Logical. Suppress the `CombineMSE()` summary message when `object` is a `list`. Default `TRUE`. |
| `Quadrant` | Character. The Kobe quadrant of `PM_Kobe`: `"green"`, `"red"`, `"yellow"`, or `"orange"`. See Details. |
| `Definition` | Character. Which stock-status metric to use in `PM_Status`, `PM_Kobe`, `PM_Red`, `PM_MinStatus`, and `PM_Safety`: `"SBiomass"` (spawning biomass, default) or `"SProduction"` (spawning production). |
| `Reference` | Character. The reference point of `PM_MinStatus`: `"MSY"` (`SBMSY`, default) or `"Unfished"` (equilibrium unfished spawning biomass, `SB0`). |
| `Lim` | Numeric, or a named numeric vector keyed by stock name. Limit reference point, expressed as a fraction of `SBMSY` (`PM_SBSBlim`) or `SPMSY` (`PM_SPSPlim`), or of `SBMSY`/`SPMSY` per `Definition` (`PM_Safety`) – e.g. `Lim = 0.5` tests against half of `SBMSY`. |
| `Year` | Numeric. The single projection year in which to evaluate rebuilding, used by `PM_Rebuild`. |
| `Target` | Numeric. Rebuilding target, expressed as a multiple of `SBMSY`, used by `PM_Rebuild`. Default `1`. |
| `Stocks` | Character vector of stock names to include. Default `NULL` uses the automatic complex/spawning-stock grouping described above. |
| `Type` | Character. For `PM_Yield`, which catch metric to report: `"Removals"` (landings + discards, default) or `"Landings"`. For `PM_AAVY`/`PM_Stability`, which series to assess interval-to-interval variability in: `"TAC"` (the MP's recommendation, default), `"Removals"`, or `"Landings"`. |

`Floor` Positive number. In `PM_LogYield`, catches are floored at `Floor` times the simulation's mean historical catch before taking logs, so a zero catch has a finite log. Default `1e-3`.

`IncludeFirst` Logical. In `PM_AAVY`, `PM_Stability`, and `PM_TACLimited`, include the change from the value in effect before the first management cycle in `Years` to the first value: for the first cycle in which the MPs are active, the TAC in the last time step before `OM@MPStartYear` (e.g. an interim TAC), or, if there is none, the removals (or landings) in the calendar year before. Default `FALSE` (`TRUE` for `PM_TACLimited`).

`Fleets` Character vector of fleet names to include in `PM_AAVE`. Default `NULL` uses all fleets.

`Threshold` Numeric. Maximum acceptable interval-to-interval change in the series selected by `Type`, used by `PM_Stability`.

`DeltaUp`, `DeltaDown`
`NULL` (default), a number, or a named numeric vector (one value per MP). The maximum proportional TAC increase and decrease in `PM_TACLimited`, overriding the `DeltaUp[2]`/`DeltaDown[2]` of the MPs.

`tol` Numeric. Tolerance of `PM_TACLimited`: a change within `tol` of the limit is at the limit. Default `1e-6`.

## Details

`PM_Removals`, `PM_Landings` (and `PM_Yield`, which is a thin wrapper around the two), `PM_LogYield`, `PM_RelYield`, `PM_AAVY`, and `PM_Stability` sum their series (TAC, removals, or landings) over the stocks in each `OM@Complexes` group by default; pass `Stocks` to override with an explicit set of stock names.

`PM_FFMSY` and `PM_Status` are evaluated at the *complex* level (`OM@Complexes`), not per stock, because `FMSY` is a single value optimised jointly across a complex's member stocks (see `F_FMSY()`).

`PM_Status` sums the status metric and its MSY reference point across each complex's spawning stock(s) before dividing, so both sides of the joint status test are assessed at the same aggregation level rather than pairing a per-stock ratio with an identical, complex-wide F flag.

`PM_Kobe` is the probability of being in one quadrant of the Kobe plot, evaluated at the complex level in the same way as `PM_Status`. The four quadrants partition every year: `"green"` (`SB > SBMSY` and `F < FMSY`), `"red"` (`SB < SBMSY` and `F > FMSY`), `"yellow"` (`SB <= SBMSY` and `F <= FMSY`), and `"orange"` (all other years: `SB >= SBMSY` and `F >= FMSY`, excluding the green and red quadrants). `PM_Status` is the green quadrant and `PM_Red` the red quadrant.

`PM_MinStatus` is the lowest annual stock status in each simulation over `Years`, relative to `SBMSY` (`Reference = "MSY"`) or to the equilibrium unfished spawning biomass `SB0` (`Reference = "Unfished"`), at the complex level. It is a statistic, not a probability (`Prob` is `NA`).

`PM_TACLimited` is the proportion of management cycles in each simulation in which the change in the TAC is at the maximum allowed by the MP: increases are compared with `DeltaUp[2]` and decreases with `DeltaDown[2]` (see `ConstrainTAC()`). The limits are the defaults of the `DeltaUp` and `DeltaDown` arguments of each MP (e.g. as set with

SetMPArgs()) unless given in DeltaUp/DeltaDown. MPs with no limits, or no TAC, are NA.

PM_Status, PM_Kobe, PM_MinStatus, and PM_Safety accept a Definition argument selecting whether stock status is assessed on spawning biomass (SBiomass(), the default) or spawning production (SProduction()). PM_SBSBMSY/PM_SBSBlim and PM_SPSPMSY/PM_SPSPlim are the corresponding single-metric, non-switchable counterparts for spawning biomass and spawning production respectively.

In seasonal models (Seasons > 1), the metrics based on MSY reference points (PM_FFMSY, PM_SBSBMSY, PM_SPSPMSY, PM_Status, PM_Kobe, PM_MinStatus, PM_SBSBlim, PM_SPSPlim, PM_Safety, and PM_Rebuild) are evaluated on annual status per calendar year, as returned by F_FMSY(), SB_SBMSY(), and SP_SPMSY(), and Years/Year refer to calendar years. PM_MinStatus with Reference = "Unfished" uses the ratio to unfished spawning biomass in the reference season of each year (see RefSeason()), the season used for SB/SBMSY. The catch-based metrics (PM_Yield, PM_Removals, PM_Landings, PM_LogYield, PM_RelYield, and PM_AAVY/PM_Stability with Type = "Removals" or "Landings") use annual catch, summed over the seasons of each complete calendar year. In all metrics, Years refers to calendar years; see PMYears() for the years in which the MPs are active, or a window of them.

### Value

A pm object.

### See Also

PM-equations for the mathematical definition of each metric.

---

## pm-class pm *Class*

---

### Description

Stores the result of evaluating a performance metric (PM) function against an mse object (or a list of mse objects, combined via CombineMSE()). Replaces the legacy PMobj class.

### Slots

Name Character. Short name of the performance metric.

Caption Character. Description of the performance metric, used for plotting/reporting.

Stat Array with dimensions Sim x Stock x MP. The underlying statistic (e.g. SB/SBMSY), averaged over the Years window, on its natural scale.

Ref Numeric. Reference/threshold value the statistic is compared against.

Prob Array with dimensions Sim x Stock x MP. Probability (proportion of years in Years meeting the objective) per simulation. Always scaled so that higher values are better outcomes.

`Mean` Array with dimensions `Stock x MP`. `Prob` averaged over `Sim`.

`MPs` Character vector. Names of the MPs evaluated.

`Years` Numeric vector. Projection years over which the PM was calculated.

### See Also

PM for the full list of performance metric functions, PM-equations for their mathematical definitions, and `CombineMSE()`.

---

`PM-equations` *Performance Metric Equations*

---

### Description

Mathematical definitions of the performance metrics computed by the PM functions. Notation: `s` indexes simulation replicates, `y` indexes the projection years in the evaluation window, and each metric is computed per management procedure (MP).

### Status

`PM_FFMSY`: fishing mortality relative to $F_{MSY}$,

$$F_{s,y}/F_{MSY,s}$$

with the probability metric $P(F/F_{MSY} < Ref)$.

`PM_SBSBMSY` / `PM_SPSPMSY`: spawning biomass or spawning production relative to its value at MSY,

$$SB_{s,y}/SB_{MSY,s} \quad \text{or} \quad SP_{s,y}/SP_{MSY,s}$$

with the probability metric $P(SB/SB_{MSY} > Ref)$ (or the SP equivalent).

`PM_Status`: the joint probability that a complex is neither overfished nor experiencing overfishing,

$$P\left(\frac{SB_y}{SB_{MSY}} > 1 \text{ and } \frac{F_y}{F_{MSY}} < 1\right)$$

evaluated at the complex level, where `SB` is spawning biomass or spawning production according to `Definition`.

`PM_Kobe`: the probability of being in a quadrant of the Kobe plot, e.g. for the red quadrant,

$$P\left(\frac{SB_y}{SB_{MSY}} < 1 \text{ and } \frac{F_y}{F_{MSY}} > 1\right)$$

(see PM for the definition of each quadrant).

`PM_MinStatus`: the lowest annual stock status over the evaluation window,

$$\min_{y \in Y} \frac{SB_{s,y}}{SB_{ref,s}}$$

where $SB_{ref}$ is $SB_{MSY}$ or $SB_0$ according to `Reference`.

**Safety**

`PM_SBSBlim` / `PM_SPSPlim`: spawning biomass or spawning production relative to MSY, relative in turn to a limit fraction `Lim` of `SBMSY`/ `SPMSY`,

$$\frac{SB_{s,y}/SB_{MSY,s}}{Lim_s} \quad \text{or} \quad \frac{SP_{s,y}/SP_{MSY,s}}{Lim_s}$$

with the probability metric $P(\frac{SB/SB_{MSY}}{Lim} > 1)$ (or the SP equivalent) – i.e. the probability that `SB/SBMSY` exceeds `Lim`.

`PM_Safety`: the probability that the stock-status metric, relative to MSY, never falls below the limit fraction `Lim` at any point during the projection,

$$P\left(\min_{y\in Y} \frac{SB_{s,y}}{SB_{MSY,s}} > Lim_s\right)$$

where `SB` is spawning biomass or spawning production according to `Definition`.

**Rebuild**

`PM_Rebuild`: for stocks overfished ($SB/SB_{MSY} < 1$) at the end of the historical period, the probability that the stock has rebuilt to a target multiple of $SB_{MSY}$ by a target year,

$$P\left(\frac{SB_{s,\text{Year}}}{SB_{MSY,s}} > \text{Target}\right)$$

**Yield**

`PM_Yield` / `PM_Removals` / `PM_Landings`: mean catch over the evaluation window,

$$\frac{1}{|Y|}\sum_{y\in Y} C_{s,y}$$

where `C` is annual removals (landings + discards), or landings only, summed over seasons in seasonal models.

`PM_LogYield`: mean log catch over the evaluation window, with catch floored at a fraction `Floor` of the simulation's mean historical catch $\bar{C}_s^{hist}$ so that a zero catch has a finite log,

$$\frac{1}{|Y|}\sum_{y\in Y} \log\left(\max\left(C_{s,y},\ \text{Floor}\cdot\bar{C}_s^{hist}\right)\right)$$

where `C` is landings (default) or removals, per `Type`.

`PM_RelYield`: mean catch over the evaluation window, expressed relative to MSY yield,

$$\frac{1}{|Y|}\sum_{y\in Y} C_{s,y}\Big/MSY_s$$

with the probability metric $P(\text{RelYield} > Ref)$ when `Ref` is supplied.

### Stability

`PM_AAVY` / `PM_AAVE`: average annual variability in TAC, removals, landings (per `Type`), or effort, across management intervals,

$$\frac{1}{|Y|-1}\sum_{y\in Y\setminus\{y_1\}}\frac{|C_{s,y}-C_{s,y-1}|}{C_{s,y-1}}$$

where `Y` is the set of management years in the evaluation window and `y-1` denotes the preceding management year.

`PM_Stability`: the probability that the series selected by `Type` changes by no more than a threshold amount between one management interval and the next,

$$P\left(\frac{|C_{s,y}-C_{s,y-1}|}{C_{s,y-1}}\leq \text{Threshold}\right)$$

evaluated per interval `y` and averaged across the evaluation window and simulations.

`PM_TACLimited`: the probability that the change in the TAC between one management interval and the next is at the limit of the MP,

$$P\left(\frac{TAC_{s,y}-TAC_{s,y-1}}{TAC_{s,y-1}}\geq \Delta^{up} \text{ or } \frac{TAC_{s,y-1}-TAC_{s,y}}{TAC_{s,y-1}}\geq \Delta^{down}\right)$$

where $\Delta^{up}$ and $\Delta^{down}$ are the maximum proportional increase and decrease (`DeltaUp[2]` and `DeltaDown[2]` of the MP).

With `IncludeFirst = TRUE`, `y-1` of the first management year in the evaluation window is the value in effect before it (see PM).

### Stat, Prob, and Mean

Every `PM_*` function returns a pm object with three related summaries: `Stat` is the raw metric above, computed per simulation (and averaged over the evaluation window for the Yield-family metrics); `Prob` is the per-simulation probability/indicator of meeting the stated objective (`NA` for metrics with no such objective, e.g. `PM_Yield`); `Mean` is `Prob` averaged across simulations, or — for metrics with no `Prob` — `Stat` averaged across simulations instead.

### See Also

PM for the full list of performance metric functions and their arguments; `NewPM()` to create a custom metric.

PMLimit *Create a table of Performance Limits and Performance Objectives*

## Description

Create a table of Performance Limits and Performance Objectives

## Usage

```
PMLimit(
  MSE,
  ...,
  Prob = NULL,
  Labels = NULL,
  FeaseMPs = NULL,
  out.file = NULL,
  output_format = "html_document",
  openFile = TRUE,
  quiet = TRUE,
  dir = NULL,
  RMDfile = NULL,
  font_size = 14,
  auto_width = FALSE,
  enableSearch = TRUE,
  PMlist = NULL,
  build = TRUE
)

PMObj(
  MSE,
  ...,
  Labels = NULL,
  out.file = NULL,
  output_format = "html_document",
  openFile = TRUE,
  quiet = TRUE,
  dir = NULL,
  RMDfile = NULL,
  font_size = 14,
  use.colors = TRUE,
  cols = NULL,
  show.legend = TRUE,
  auto_width = FALSE,
  enableSearch = TRUE,
  PMlist = NULL,
  build = TRUE,
```

```
  cex.tex = 0.75,
  inc.title = TRUE,
  title = "Legend"
)
```

### Arguments

| | |
|---|---|
| `MSE` | An object of class 'MSE' |
| `...` | PM objects to be used as performance limits. Characters (i.e names of PM objects) |
| `Prob` | Minimum probability threshold |
| `Labels` | Optional named list specifying new labels for MPs. For example: `Labels = list(AvC="Average Catch", CC1="Constant Catch")` |
| `FeaseMPs` | Optional. Character vector of MP names that are considered feasible. e.g. the output from `Fease()` |
| `out.file` | Name of the output file. If none provided, output file will be named 'PerfLimTable' |
| `output_format` | Output file format. Currently only 'html_document' is supported |
| `openFile` | Logical. Should the file be opened in browser? |
| `quiet` | Logical. An option to suppress printing of the pandoc command line. |
| `dir` | Optional. Directory for output file. Default is working directory. |
| `RMDfile` | Optional. RMD template file |
| `font_size` | Numeric. Font size for text in the table |
| `auto_width` | Logical. Should table be width be automatic? |
| `enableSearch` | Currently disabled. Logical. Should search be enabled in the html table? |
| `PMlist` | Optional. List of PM names. |
| `build` | Logical. Build the html table? |
| `use.colors` | Logical. Color scale the probability text? |
| `cols` | Optional character vector of colors for probability text |
| `show.legend` | Logical. Show the legend?? |
| `cex.tex` | Size of legend text |
| `inc.title` | Logical. Include title for legend? |
| `title` | Title for the legend |

### Value

`PMLimit` invisibly returns names of MPs that pass all performance limits

### Functions

- `PMLimit()`: Create a table of Performance Limits
- `PMObj()`: Create a table of Performance Objectives.

### Author(s)

A. Hordyk

### Examples

```
## Not run:
MSE <- runMSE()
PMLimit(MSE, "P50", "PNOF", Prob=0.9)
PMObj(MSE, "P100", "LTY")

## End(Not run)
```

---

## PMobj-class *An object for storing data for analysis using data-limited methods*

---

### Description

Used internally

### Slots

`Name` Name of the Performance Metric. Character

`Caption` A caption to be used in plots. Character, call, or function.

`Stat` Statistic of interest for the PM. Dimensions: nsim, nMP, yrs. Array

`Ref` Reference value to calculate probability for statistic. Numeric.

`Prob` Probability (mean over years) Dimensions: nsim by MP. Matrix, numeric or data.frame

`Mean` Mean probability (mean over years and simulations). Numeric. Length nMPs

`MPs` Name of MPs. Single value. Character string

### Objects from the Class

Objects can be created by calls of the form `new('PMobj')`

### Author(s)

A. Hordyk

`PMYears` *Years in Which the Management Procedures Are Active*

## Description

The calendar years of the projection period in which the management procedures (MPs) are active (from `OM@MPStartYear`), or a window of them, for the `Years` argument of the PM functions.

## Usage

```
PMYears(object, Window = c("all", "first", "last", "middle"), n = 10, Skip = 0)
```

## Arguments

| | |
|---|---|
| `object` | An om, hist, or mse object, or a `list` of mse objects (the first is used). |
| `Window` | Character. The window of years: `"all"` (default), `"first"`, `"last"`, or `"middle"`. See Details. |
| `n` | Integer. The number of years of the `"first"`, `"last"`, and `"middle"` windows. Default `10`. |
| `Skip` | Integer. The number of years to drop from the start before applying `Window`. Default `0`. |

## Details

The years are the complete calendar years of the projection period from `OM@MPStartYear` (all projection years if `MPStartYear` is `NULL`). The first `Skip` years are dropped, and `Window` selects:

- `"all"`: all the years.
- `"first"`: the first `n` years.
- `"last"`: the last `n` years.
- `"middle"`: the years between the first `n` and the last `n` years.

## Value

A numeric vector of calendar years.

## See Also

PM

### Examples

```
## Not run:
PMYears(MSE)                      # all years the MPs are active
PMYears(MSE, 'first', n = 10)     # the first 10
PMYears(MSE, 'middle', n = 10)    # between the first 10 and the last 10
PMYears(MSE, 'last', n = 1)       # the terminal year
PMYears(MSE, Skip = 5)            # all but the first 5
PM_Status(MSE, Years = PMYears(MSE, 'last', n = 15))

## End(Not run)
```

---

## popdynamics-class popdynamics *Class*

---

### Description

Stores population time-series for a single stock across simulations, ages, years, and areas.

### Slots

`Number` Number-at-age. List of length `nStock`, each element an array with dimensions `Sim`, `Age`, `Year`, `Area`.

`Biomass` Total biomass. Array with dimensions `Sim`, `Stock`, `Year`.

`SBiomass` Spawning biomass. Array with dimensions `Sim`, `Stock`, `Year`.

`SProduction` Spawning production. Array with dimensions `Sim`, `Stock`, `Year`.

`Misc` Miscellaneous list for additional outputs.

---

## Populate *Populate Operating Model Components*

---

### Description

Generic dispatcher function to populate operating model objects and their components.

### Usage

```
Populate(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | An object to populate. The class of `object` determines which class-specific `Populate*()` function is called. |
| `...` | Named arguments passed to the appropriate `Populate*()` function. Only arguments relevant to the class of `object` are used. |

### Details

Generally, users will not need to use the `Populate*` functions as they are called internally.

Depending on the class of `object`, the function calls a class-specific `Populate*()` function that generates stochastic values, fills derived slots, and checks object structure and contents.

`Populate()` is a wrapper around class-specific population functions. The following methods may be called depending on the class of `object`:

- `PopulateOM()`
- `PopulateStock()`
- `PopulateLength()`
- `PopulateWeight()`
- `PopulateNaturalMortality()`
- `PopulateMaturity()`
- `PopulateFecundity()`
- `PopulateSRR()`
- `PopulateSpatial()`
- `PopulateDepletion()`
- `PopulateFleet()`
- `PopulateEffort()`
- `PopulateCatchability()`
- `PopulateSelectivity()`
- `PopulateRetention()`
- `PopulateDiscardMortality()`
- `PopulateObs()`
- `PopulateEffortObs()`
- `PopulateCatchObs()`
- `PopulateIndexObs()`
- `PopulateCompObs()`
- `PopulateImp()`

Each `Populate*()` function has its own documentation detailing which named arguments it accepts. Users should consult the specific `Populate*()` help page for information about required slots, default behavior, and argument descriptions.

### Value

The populated object, of the same class as `object`.

### Examples

```
## Not run:
# Populate a full operating model
OM <- Populate(OM)

# Populate a stock with specific arguments
Stock <- Populate(Stock, nYear = 40, pYear = 20, nSim = 100)

# Populate a fleet using an already-populated stock
Fleet <- Populate(Fleet, Stock = Stock, nSim = 100)

## End(Not run)
```

---

## PopulateCatchability *Populate a Catchability Object*

---

### Description

Populate a `Catchability` object by setting fleet/gear efficiency values for historical and projected years, and applying optional stochastic or increasing catchability parameters.

### Usage

```
PopulateCatchability(
  Catchability,
  nSim = 5,
  HistYears = NULL,
  ProjYears = NULL,
  seed = NULL,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| `Catchability` | A `Catchability()` object to populate. |
| `nSim` | Integer. Number of simulation replicates. |
| `HistYears` | Numeric vector of historical years. |
| `ProjYears` | Numeric vector of projected years. |
| `seed` | Integer. Random number seed used for stochastic generation. |
| `silent` | Logical; if `TRUE`, suppresses messages. |

### Details

`PopulateCatchability()` performs the following steps:

- Initializes `Efficiency` to 1 if not provided.

- Checks that `Efficiency` has correct dimensions (rows = 1 or `nSim`).
- Adds dimension names if missing.
- Applies `qInc` (annual increase in catchability) if provided.
- Applies `qCV` (coefficient of variation) to introduce stochasticity.

After population, `qInc` and `qCV` hold the per-simulation values used, and the multipliers applied to the projection years of `Efficiency` are stored in `Misc$qAdjust`. Re-populating leaves `Efficiency` unchanged while `qInc` and `qCV` are unchanged. If either is modified, its previous multiplier is divided out of `Efficiency` before the new one is applied.

### Value

A populated `Catchability()` object.

### Examples

```
## Not run:
Catch <- Catchability()
Catch_pop <- PopulateCatchability(
  Catchability = Catch,
  nSim = 50,
  HistYears = 2000:2020,
  ProjYears = 2021:2025,
  seed = 123
)

## End(Not run)
```

---

## `PopulateCatchObs` *Populate Catch Observation Error*

---

### Description

Populate a catchobs object by expanding CV, generating stochastic error arrays, applying multiplicative bias, and generating a stochastic reference level, all across simulation replicates and years.

### Usage

```
PopulateCatchObs(Catch, nSim, HistYears, ProjYears)
```

### Arguments

| | |
|---|---|
| `Catch` | A catchobs object. Used for both landed catch (`Obs@Landings`) and discarded catch (`Obs@Discards`). |
| `nSim` | Integer. Number of simulation replicates. |
| `HistYears` | Integer vector. Calendar years of the historical period. |
| `ProjYears` | Integer vector. Calendar years of the projection period. |

### Details

If EmptyObject() returns TRUE for Catch, the object is returned unchanged and no population is performed.

### Value

A populated catchobs object.

### See Also

PopulateObs(), catchobs

---

## PopulateClosure *Populate a Closure Object*

---

### Description

Populates a Closure array for a fleet, ensuring proper dimensions across simulations, years, and spatial areas.

### Usage

```
PopulateClosure(Closure, nArea, nSim = 5, Years, silent = FALSE)
```

### Arguments

| | |
|---|---|
| Closure | An array representing fleet-specific area closures, or NULL. |
| nArea | Integer. Number of spatial areas. |
| nSim | Integer. Number of simulation replicates. |
| Years | Numeric vector of model years. |
| silent | Logical; if TRUE, suppresses informational messages. |

### Details

PopulateClosure() ensures the Closure array has dimensions: sim × year × area. If the array is empty, it defaults to 1 (open) for all areas.

Existing arrays are checked for compatibility with nArea. Dimension names for Sim, Year, and Area are automatically added if missing.

### Value

An array representing fleet area closures with proper dimensions and dimnames.

### Examples

```
## Not run:
Closure <- NULL
Closure_pop <- PopulateClosure(Closure, nArea = 5, nSim = 3, Years = 2000:2025)

## End(Not run)
```

---

`PopulateCompObs` *Populate Composition Observation Error*

---

### Description

Populate a compobs object by expanding `SampleSize`, `ESS`, `Theta`, and `Shift` across simulation replicates, years, and composition bins, and applying default values where slots are unspecified.

### Usage

```
PopulateCompObs(
  Comp,
  nSim,
  HistYears,
  ProjYears,
  Bins = NULL,
  BinName = "Class"
)
```

### Arguments

| | |
|---|---|
| `Comp` | A compobs object. Used for landed catch-at-age (`Obs@LandingsAtAge`), discarded catch-at-age (`Obs@DiscardsAtAge`), landed catch-at-size (`Obs@LandingsAtSize`), and discarded catch-at-size (`Obs@DiscardsAtSize`). |
| `nSim` | Integer. Number of simulation replicates. |
| `HistYears` | Integer vector. Calendar years of the historical period. |
| `ProjYears` | Integer vector. Calendar years of the projection period. |
| `Bins` | Numeric vector. Age classes for age composition slots, or length class midpoints for size composition slots. Used to set the length of the `Bin` dimension and to name it in the populated `Shift` array. `NULL` is accepted only when `EmptyObject()` returns `TRUE` for `Comp`. |
| `BinName` | Character scalar. Name used for the bin dimension in populated arrays, typically `"Age"` or `"Class"`. |

### Details

If `EmptyObject()` returns `TRUE` for `Comp`, or both `SampleSize` and `ESS` are `NULL`, the object is returned without composition generation settings and no composition data are generated.

Otherwise the following slots are populated:

`SampleSize`: expanded to a named `[nSim x nYear]` array covering all historical and projection years. A length-2 input `c(lower, upper)` is interpreted as bounds of a Uniform distribution from which `nSim` values are drawn. If `NULL`, defaults silently to the populated `ESS` array.

`ESS`: expanded in the same way as `SampleSize`. If `NULL`, defaults silently to the populated `SampleSize` array. Typically `ESS <= SampleSize`; values of `ESS < SampleSize` produce overdispersion relative to a pure multinomial with `SampleSize` draws.

`Theta`: expanded to a named `[nSim x nYear]` array of values in `(0, 1]`. If `NULL`, defaults silently to 1 (standard multinomial, no additional overdispersion). A length-2 input is treated as Uniform bounds.

`Shift`: expanded to a named `[nSim x nYear x nBin]` array where `nBin = length(Bins)`. Accepted input forms:

- `NULL` (default): no shift applied; slot remains `NULL`.
- Scalar: constant offset across all simulations, years, and bins.
- Vector of length `nBin`: bin-specific offset, constant across simulations and years.
- Named matrix with any subset of `Sim`, `Year`, and `Bin` dimensions with change-point years.
- Full `[nSim x nYear x nBin]` array.

In conditioning mode, `ESS`, `Theta`, and `Shift` are populated internally.

**Composition generation model:**

After population, the fully-expanded slots are used by the simulation engine to generate observed compositions as follows. Let $\mathbf{q}$ be the OM-predicted composition vector for a given simulation, year, and fleet:

1. Base concentration: $\alpha = \text{ESS} \times \mathbf{q}$
2. Apply shift (if non-`NULL`): $\alpha'_b = \alpha_b \times \exp(\text{Shift}[\text{sim}, \text{year}, b])$
3. Dirichlet draw: $\mathbf{p}^* \sim \text{Dirichlet}(\alpha'/\text{Theta})$
4. Multinomial draw: $\text{obs} \sim \text{Multinomial}(\text{SampleSize}, \mathbf{p}^*)$

### Value

A populated compobs object.

### See Also

`CompObs()`, compobs, `PopulateObs()`,

`PopulateDepletion` *Populate a Depletion Object*

## Description

Populate a `Depletion` object by generating initial and final depletion values across simulation replicates, and validating the reference biomass.

## Usage

```
PopulateDepletion(Depletion, nSim = 5, seed = NULL, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `Depletion` | A `Depletion()` object to populate. |
| `nSim` | Integer. Number of simulation replicates. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |

## Details

`PopulateDepletion()` handles population of initial and final depletion values. Steps include:

- Generating stochastic values if required for `Initial` and `Final` slots.
- Validating that the `Reference` slot is one of the accepted types: `"B0"`, `"BMSY"`, `"SB0"`, or `"SBMSY"`.
- Returning a fully populated `Depletion` object.

## Value

A populated `Depletion()` object.

## Examples

```
## Not run:
# Assuming a Depletion object `D` exists
D_pop <- PopulateDepletion(D, nSim = 10, seed = 123)

## End(Not run)
```

`PopulateDiscardMortality`
*Populate a DiscardMortality Object*

## Description

Populates a `DiscardMortality` object with mean-at-age and mean-at-length values.

## Usage

```
PopulateDiscardMortality(
  DiscardMortality,
  Ages = NULL,
  Length = NULL,
  Selectivity = NULL,
  Retention = NULL,
  nSim = 5,
  Years = NULL,
  nArea = 1,
  CalcAtLength = TRUE,
  seed = NULL,
  silent = FALSE,
  force = FALSE,
  replace = FALSE,
  ASKOverride = NULL
)
```

## Arguments

| | |
|---|---|
| `DiscardMortality` | A `DiscardMortality()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. |
| `Length` | A `Length()` object. Required if the model depends on length-at-age. |
| `Selectivity` | A populated `Selectivity()` object. Optional, but recommended whenever discard mortality is defined at length. Ignored if either `Selectivity` or `Retention` is unavailable, or either's `isAtLength` is `FALSE`. |
| `Retention` | A populated `Retention()` object. See `Selectivity`. |
| `nSim` | Integer. Number of simulation replicates. |
| `Years` | Numeric vector of model years. |
| `nArea` | Integer. Number of spatial areas. |
| `CalcAtLength` | Logical; if `TRUE`, calculates mean-at-length from age-based arrays. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical; if `TRUE`, suppresses informational messages. |

| | |
|---|---|
| `force` | Logical; if `TRUE`, forces re-population even if digest indicates object is current. |
| `replace` | Used internally. |
| `ASKOverride` | Used internally. |

### Details

`PopulateDiscardMortality()` handles population of discard mortality at age and optionally at length. Steps include:

- Converting mean-at-length to mean-at-age, weighted by `Selectivity x (1 - Retention)` when both are available and length-native (see `Selectivity`), otherwise via the plain (unweighted) age-length key
- Calculating mean-at-length from age-based arrays if `CalcAtLength = TRUE`
- Adding an 'Area' dimension and setting dimension names
- Handling empty objects by setting discard mortality to zero

### Value

A populated `DiscardMortality()` object.

### Examples

```
## Not run:
DM <- DiscardMortality()
DM_pop <- PopulateDiscardMortality(
  DM, Ages = Ages, Length = Length, Years = 2000:2025, nSim = 5
)

## End(Not run)
```

---

## `PopulateEffort` *Populate an Effort Object*

---

### Description

Populate an `Effort` object by generating historical effort, and populating distribution and targeting across areas and simulation replicates.

### Usage

```
PopulateEffort(
  Effort,
  HistYears,
  ProjYears = NULL,
  nArea = 1,
  nSim = 5,
  seed = NULL
)
```

### Arguments

| | |
|---|---|
| `Effort` | An effort object to populate. |
| `HistYears` | Numeric vector of historical years to populate effort for. |
| `ProjYears` | Numeric vector of projected years. Used to extend `TripsScalar`/`AnglerPerTrip`/`Theta` (if supplied) to cover the full model time series; not used for `Effort@Effort` itself. |
| `nArea` | Integer. Number of spatial areas. |
| `nSim` | Integer. Number of simulation replicates. |
| `seed` | Integer. Random number seed used for stochastic generation. |

### Details

`PopulateEffort()` performs the following steps:

- Generates historical effort if the `Effort` slot is a data frame.
- Ensures the effort matrix has correct dimensions and names.
- Validates fleet effort distribution and targeting across areas.
- If supplied, extends `TripsScalar`/`AnglerPerTrip`/`Theta` to cover historical and projected years (a bare scalar, length-2 uniform bounds, or length-`nSim` vector for `Theta` is first resolved to a single-year `Sim x Year` array, then replicated across all years).

### Value

A populated `Effort()` object.

### Examples

```
## Not run:
Eff <- Effort()
Eff_pop <- PopulateEffort(
  Effort = Eff,
  HistYears = 2000:2025,
  nArea = 3,
  nSim = 50,
  seed = 123
)

## End(Not run)
```

`PopulateEffortObs` *Populate Effort Observation Error*

## Description

Populate an effortobs object by expanding CV, generating stochastic error arrays, and generating multiplicative bias across simulations.

## Usage

```
PopulateEffortObs(Effort, nSim, HistYears, ProjYears)
```

## Arguments

| | |
|---|---|
| `Effort` | An effortobs object. |
| `nSim` | Integer. Number of simulation replicates. |
| `HistYears` | Integer vector. Calendar years of the historical period. |
| `ProjYears` | Integer vector. Calendar years of the projection period. |

## Details

If `EmptyObject()` returns `TRUE` for `Effort`, the object is returned unchanged and no population is performed.

Otherwise the following slots are populated in order:

- `CV`: expanded to a named `[nSim]` array.
- `Error`: generated as a lognormal draw from `CV` across all historical and projection years, producing a named `[nSim x nYear]` array. If `Error` is already a fully-formed array it is validated and dimension names are applied.
- `Bias`: expanded to a named `[nSim]` multiplicative bias array. Defaults to 1 (no bias) if unspecified.

## Value

A populated effortobs object.

## See Also

`PopulateObs()`, effortobs

`PopulateFecundity` *Populate a Fecundity Object*

## Description

Populate a `Fecundity` object by generating mean-at-age values, adding stochastic deviations, and optionally calculating fecundity-at-length. Handles models that depend on age, length, or weight.

## Usage

```
PopulateFecundity(
  Fecundity,
  Ages = NULL,
  Length = NULL,
  Weight = NULL,
  Maturity = NULL,
  Years = NULL,
  nSim = 5,
  seed = NULL,
  silent = FALSE,
  CalcAtLength = FALSE,
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| `Fecundity` | A `Fecundity()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. |
| `Length` | A `Length()` object. Required if the fecundity model depends on length-at-age. |
| `Weight` | A `Weight()` object. Required if the fecundity model depends on weight-at-age. |
| `Maturity` | A `Maturity()` object. Required to calculate fecundity as egg production of mature individuals. |
| `Years` | Numeric vector of model years. |
| `nSim` | Integer. Number of simulation replicates. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `CalcAtLength` | Logical. If `TRUE`, calculate fecundity-at-length values from age-based arrays. |
| `force` | Logical. If `TRUE`, force re-population even if the object digest is unchanged. |

### Details

`PopulateFecundity()` handles population of mean fecundity-at-age for a stock. Steps include:

- Structuring model parameters across `nSim` and `Years`
- Resolving the fecundity model class
- Generating mean fecundity at age, or at length if applicable
- Converting between mean-at-length and mean-at-age
- Multiplying by maturity to represent egg production of mature individuals
- Adding stochastic variation

### Value

A populated `Fecundity()` object.

### See Also

`Populate()`, `PopulateStock()`, `PopulateLength()`, `PopulateWeight()`, `PopulateMaturity()`

### Examples

```
## Not run:
# Assuming `Ages`, `Length`, `Weight`, and `Maturity` objects exist
F <- Fecundity()
F_pop <- PopulateFecundity(
  F,
  Ages = Ages,
  Length = Length,
  Weight = Weight,
  Maturity = Maturity,
  Years = 2000:2025,
  nSim = 5,
  seed = 123,
  CalcAtLength = TRUE
)

## End(Not run)
```

---

## `PopulateFleet` *Populate a Fleet Object*

---

### Description

Populate a `Fleet` object by generating effort, catchability, selectivity, retention, discard mortality, and spatial closures for a fleet.

### Usage

```
PopulateFleet(
  Fleet,
  Stock,
  seed = 103,
  silent = FALSE,
  force = FALSE,
  WeightFallback = NULL,
  ExtendedLength = NULL,
  ExtendedWeight = NULL
)
```

### Arguments

| | |
|---|---|
| `Fleet` | A `Fleet()` object to populate. |
| `Stock` | A populated `Stock()` object. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `force` | Logical. If `TRUE`, force re-population even if digest indicates the object is current. |
| `WeightFallback` | Used internally. |
| `ExtendedLength` | Used internally. |
| `ExtendedWeight` | Used internally. |

### Details

`PopulateFleet()` performs the following steps:

- Populates fleet effort for historical years.
- Populates fleet catchability for historical and projected years.
- Populates selectivity and retention objects.
- Populates discard mortality and spatial closures.
- Initializes `WeightFleetRetained` and `WeightFleetSelected` if not already defined, by weighting the stock's weight-at-length by selectivity-at-length (`WeightFleetSelected`) or by selectivity-at-length times retention-at-length (`WeightFleetRetained`) over the age-length distribution.

### Value

A populated `Fleet()` object.

### See Also

`Populate()`, `PopulateStock()`

### Examples

```
## Not run:
F <- Fleet()
F_pop <- PopulateFleet(Fleet = F, Stock = Stock, seed = 123, nSim = 50)

## End(Not run)
```

---

`PopulateImp` *Populate an Imp Object*

---

### Description

Populates the `TAC`, `Effort`, and `Size` impslot sub-objects of an imp object across simulation replicates and years.

### Usage

```
PopulateImp(Imp, nSim, HistYears, ProjYears)
```

### Arguments

| | |
|---|---|
| `Imp` | An imp object. |
| `nSim` | Integer. Number of simulation replicates. |
| `HistYears` | Integer vector. Calendar years of the historical period. |
| `ProjYears` | Integer vector. Calendar years of the projection period. |

### Value

A populated imp object.

### See Also

`PopulateImpSlot()`, imp, impslot

---

`PopulateImpSlot` *Populate an ImpSlot Object*

---

### Description

Expands `Mean` and `SD` to named `[Sim]` arrays, `Compliance` to a named `[Sim x Year]` array, and generates a stochastic lognormal `Error` array from `Mean`/`SD` if not supplied directly. Mirrors `PopulateCatchObs()`'s `CV`/`Bias`/`Error` handling, applied to `Mean`/`SD` instead. Left untouched (all slots `NULL`) if the input object is empty – see `EmptyObject()` – so that the (common) case of an unconfigured implementation model incurs no extra computation.

### Usage

```
PopulateImpSlot(ImpSlot, nSim, Years)
```

### Arguments

| | |
|---|---|
| ImpSlot | An impslot object. |
| nSim | Integer. Number of simulation replicates. |
| Years | Integer vector. Calendar years (historical and projection). |

### Details

Compliance is expanded to a [Sim x Year] array, using the same conventions as every other structured OM parameter: a scalar is a fixed value for every simulation and year; a length-2 vector is c(lower, upper) bounds sampled once per simulation (held constant across years); a full array is respected as-is, with named Year dimnames treated as breakpoints and forward-filled to cover all of Years (see Extend()/ExtendYears()).

### Value

A populated impslot object.

### See Also

PopulateImp(), impslot, ImpSlot()

---

## PopulateIndexObs *Populate Index Observation Error*

---

### Description

Populate an indicesobs object by expanding CV, generating stochastic error arrays, applying lag-1 autocorrelation, applying multiplicative bias, and populating index years, across simulation replicates and years.

### Usage

```
PopulateIndexObs(Index, nSim, HistYears, ProjYears)
```

### Arguments

| | |
|---|---|
| Index | An indicesobs object. Used for both CPUE indices (Obs@CPUE) and fishery-independent survey indices (Obs@Survey). |
| nSim | Integer. Number of simulation replicates. |
| HistYears | Integer vector. Calendar years of the historical period. |
| ProjYears | Integer vector. Calendar years of the projection period. |

### Details

If `EmptyObject()` returns `TRUE` for `Index`, the object is returned unchanged and no population is performed.

Otherwise the following slots are populated in order:

- `CV` — expanded to a named `[nSim]` array.
- `Error` — generated as a lognormal draw from `CV` across all historical and projection years, producing a named `[nSim x nYear]` array. If `Error` is already a fully-formed array it is validated and dimension names are applied. If `AC` is non-zero and `Error` was freshly generated (i.e. not already supplied by the user), the error series is subsequently transformed to an AR(1) process (see below). A user-supplied `Error` array is never transformed by `AC`, since it may already encode its own autocorrelation structure.
- `AC` — expanded to a named `[nSim]` array. Accepted inputs: scalar applied to all simulations, length-2 vector `c(lower, upper)` for a per-simulation Uniform draw, or a vector of length `nSim` used directly. Defaults to 0 (no autocorrelation).
- `Ref` — generated as a lognormal draw from a CV, producing a named `[nSim]` reference level error array.
- `qInc` — expanded to a named `[nSim]` array (scalar, length-2 uniform range, or length-`nSim` vector); `NULL` if all values are 0.
- `Years` — if empty, defaults to all historical years (`HistYears`).

The following slots are used internally during the simulation and are not populated here: `Beta`, `Selectivity`, `Type`, `Efficiency`.

**Autocorrelated index error:**

When `AC` is non-zero and `Error` was freshly generated, the independently-drawn lognormal error series is transformed into a variance-preserving AR(1) process on the log scale. Let $\varepsilon_t = \log(\text{Error}_t)$ be the raw log-errors drawn from $N(\mu, \sigma^2)$ where $\mu$ and $\sigma$ are derived from `CV`. The AR(1) series is constructed as:

$$\varepsilon_t^* = \rho\, \varepsilon_{t-1}^* + \sqrt{1-\rho^2}\, \varepsilon_t$$

where $\rho$ is the per-simulation lag-1 autocorrelation coefficient (`AC`). This formulation preserves the marginal variance $\text{Var}(\varepsilon_t^*) = \sigma^2$ for all values of $\rho$. The AR(1) series is then exponentiated to return to the natural scale: $\text{Error}_t^* = \exp(\varepsilon_t^*)$.

### Value

A populated indicesobs object.

### See Also

`PopulateObs()`, indicesobs

PopulateLength *Populate a Length Object*

## Description

Populate a `Length` object by generating stochastic length-at-age values, expanding parameters across simulation replicates and years, and optionally constructing age–length keys.

## Usage

```
PopulateLength(
  Length,
  Ages = NULL,
  Years = NULL,
  nSim = 5,
  ALK = TRUE,
  seed = NULL,
  silent = FALSE,
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| `Length` | A `Length()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. If `NULL`, defaults ages 0:10 are used. |
| `Years` | Numeric vector of model years. If `NULL`, defaults years `seq(1950, CurrentYear + 5)` are used. |
| `nSim` | Integer. Number of simulation replicates. |
| `ALK` | Logical. If `TRUE`, populate age–length keys. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `force` | Logical. If `TRUE`, force re-population even if the object digest is unchanged. |

## Details

`PopulateLength()` expands length-at-age parameters and generates stochastic realizations across simulations and years. Population involves:

- Structuring model parameters across `nSim` and `Years`
- Resolving the length-at-age model
- Generating mean and random length-at-age values
- Expanding coefficients of variation
- Optionally constructing age–length keys

To avoid unnecessary recomputation, a digest of the length object and key arguments is checked before population. If the digest is unchanged and `force = FALSE`, the input object is returned unchanged.

### Value

A populated `Length()` object.

### See Also

`Populate()`, `PopulateStock()`

### Examples

```
## Not run:
Length <- PopulateLength(
  Length,
  Ages = Ages,
  Years = Years,
  nSim = 100,
  ALK = TRUE,
  seed = 123
)

## End(Not run)
```

---

## `PopulateMaturity` *Populate a Maturity Object*

---

### Description

Populate a `Maturity` object by generating maturity-at-age or maturity-at-weight values for a stock, expanding parameters across simulation replicates and years, and optionally calculating values at length.

### Usage

```
PopulateMaturity(
  Maturity,
  Ages = NULL,
  Length = NULL,
  Weight = NULL,
  Years = NULL,
  nSim = 5,
  seed = NULL,
  silent = FALSE,
  CalcAtLength = FALSE,
  force = FALSE
)
```

### Arguments

| | |
|---|---|
| Maturity | A Maturity() object to populate. |
| Ages | An Ages() object defining age classes. |
| Length | A Length() object. Required if the maturity model depends on length-at-age. |
| Weight | A Weight() object. Required if the maturity model depends on weight-at-age. |
| Years | Numeric vector of model years. |
| nSim | Integer. Number of simulation replicates. |
| seed | Integer. Random seed used for stochastic generation. |
| silent | Logical. If TRUE, suppress informational messages. |
| CalcAtLength | Logical. If TRUE, calculate maturity-at-length values from age-based arrays. |
| force | Logical. If TRUE, force re-population even if the object digest is unchanged. |

### Details

PopulateMaturity() handles population of maturity-at-age or maturity-at-weight for a stock. Steps include:

- Structuring model parameters across nSim and Years
- Resolving the maturity model class
- Generating mean maturity at age, at length, or at weight as appropriate
- Converting between mean-at-length/weight and mean-at-age
- Adding stochastic variation
- Setting semelparous array if applicable

### Value

A populated Maturity() object.

### See Also

Populate(), PopulateStock(), PopulateLength(), PopulateWeight()

### Examples

```
## Not run:
M <- PopulateMaturity(
  Maturity,
  Ages = Ages,
  Length = Length,
  Weight = Weight,
  Years = Years,
  nSim = 100,
  CalcAtLength = TRUE,
```

```
  seed = 123
)

## End(Not run)
```

---

`PopulateNaturalMortality` *Populate a Natural Mortality Object*

---

## Description

Populate a `NaturalMortality` object by generating stochastic natural mortality-at-age values, expanding parameters across simulation replicates and years, and optionally calculating values at length.

## Usage

```
PopulateNaturalMortality(
  NaturalMortality,
  Ages = NULL,
  Length = NULL,
  Years = NULL,
  nSim = 5,
  CalcAtLength = FALSE,
  seed = NULL,
  silent = FALSE,
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| `NaturalMortality` | A `NaturalMortality()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. |
| `Length` | A `Length()` object. Required if the mortality model depends on length-at-age. |
| `Years` | Numeric vector of model years. |
| `nSim` | Integer. Number of simulation replicates. |
| `CalcAtLength` | Logical. If `TRUE`, calculate mortality-at-length values from age-based arrays. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `force` | Logical. If `TRUE`, force re-population even if the object digest is unchanged. |

### Details

`PopulateNaturalMortality()` handles natural mortality-at-age population for a stock, including stochastic generation and propagation across simulations and years. Steps include:

- Structuring model parameters across `nSim` and `Years`
- Resolving the mortality model class
- Generating mean natural mortality at age or at length
- Converting between mean-at-length and mean-at-age if necessary
- Adding stochastic variation

### Value

A populated `NaturalMortality()` object.

### See Also

`Populate()`, `PopulateStock()`, `PopulateLength()`

### Examples

```
## Not run:
NM <- PopulateNaturalMortality(
  NaturalMortality,
  Ages = Ages,
  Length = Length,
  Years = Years,
  nSim = 100,
  CalcAtLength = TRUE,
  seed = 123
)

## End(Not run)
```

---

## PopulateObs *Populate an Obs Object*

---

### Description

Populate an obs object by expanding observation error parameters across simulation replicates and years, generating stochastic error arrays, and applying default values where slots are unspecified.

### Usage

```
PopulateObs(Obs, HistYears, ProjYears, nSim, AgeBins = NULL, SizeBins = NULL)
```

### Arguments

| | |
|---|---|
| `Obs` | An obs object to populate. |
| `HistYears` | Integer vector. Calendar years of the historical period. |
| `ProjYears` | Integer vector. Calendar years of the projection period. |
| `nSim` | Integer. Number of simulation replicates. |
| `AgeBins` | Numeric vector. Age class values used to name the `Bin` dimension of populated age-composition arrays. Required when `Obs@LandingsAtAge` or `Obs@DiscardsAtAge` is non-empty. |
| `SizeBins` | Numeric vector. Length class midpoints, used to name the `Bin` dimension of populated size-composition arrays. Required when `Obs@LandingsAtSize` or `Obs@DiscardsAtSize` is non-empty. |

### Details

`PopulateObs()` coordinates population of all observation error components of an obs object, calling a dedicated sub-function for each slot:

- Effort observation error via `PopulateEffortObs()`
- Landed catch observation error via `PopulateCatchObs()`
- Discarded catch observation error via `PopulateCatchObs()`
- CPUE index observation error via `PopulateIndexObs()`
- Survey index observation error via `PopulateIndexObs()`
- Landed catch-at-age composition error via `PopulateCompObs()`
- Discarded catch-at-age composition error via `PopulateCompObs()`
- Landed catch-at-size composition error via `PopulateCompObs()`
- Discarded catch-at-size composition error via `PopulateCompObs()`

Slots for `LifeHistory` and `Exploitation` observation error are reserved for future implementation and are not populated.

If a slot contains an empty object (as determined by `EmptyObject()`), it is returned unchanged. See the individual sub-function documentation for details on expansion rules, default values, and accepted input forms.

### Value

A populated obs object.

### See Also

`Populate()`, `PopulateEffortObs()`, `PopulateCatchObs()`, `PopulateIndexObs()`, `PopulateCompObs()`

PopulateOM *Populate an Operating Model*

## Description

Populate a complete operating model (om object) by populating all component objects (stocks, fleets, observation models, and derived structures), performing internal consistency checks, and initializing simulation-specific values.

## Usage

```
PopulateOM(
  OM,
  silent = FALSE,
  force = FALSE,
  standardize_effort = TRUE,
  adjust_fecundity = TRUE
)
```

## Arguments

| | |
|---|---|
| OM | An OM() object to populate. |
| silent | Logical. If TRUE, suppress informational messages and warnings during population. |
| force | Logical. If TRUE, force re-population even if the internal object digest indicates no changes since the last call. |
| standardize_effort | Logical. Used internally. Apply StandardizeEffort()? |
| adjust_fecundity | Logical. Used internally. Apply AdjustSeasonalFecundity()? |

## Details

This function is typically called internally, but may also be called directly.

PopulateOM() is the top-level population routine for operating models. It orchestrates population of all model components, including:

- Stocks via PopulateStock()
- Fleets via PopulateFleet()
- Implementation models (imp)
- Observation models (obs)

For multi-stock OMs, projection recruitment deviations are additionally correlated across stocks based on historical covariance (see GenMultiStockRecDevs()), unless disabled via OM@Control$CorrelatedRecDevs <- FALSE.

To avoid unnecessary recomputation, a digest of the operating model is checked before population. If the digest is unchanged and force = FALSE, the input object is returned unchanged.

### Value

An `OM()` object populated with simulation- and time-specific values.

### See Also

`Populate()`, `PopulateStock()`, `PopulateFleet()`

### Examples

```
## Not run:
OM <- PopulateOM(OM)


## End(Not run)
```

---

`PopulateRetention` *Populate a Retention Object*

---

### Description

Populates a `Retention` object with mean-at-age, mean-at-length, or mean-at-weight values.

### Usage

```
PopulateRetention(
  Retention,
  Ages = NULL,
  Length = NULL,
  Weight = NULL,
  Maturity = NULL,
  Selectivity = NULL,
  nSim = 5,
  Years = NULL,
  nArea = 1,
  CalcAtLength = TRUE,
  seed = NULL,
  silent = FALSE,
  force = FALSE,
  replace = FALSE,
  ASKOverride = NULL
)
```

### Arguments

| | |
|---|---|
| `Retention` | A `Retention()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. |

| | |
|---|---|
| `Length` | A `Length()` object. Required if the retention model depends on length-at-age. |
| `Weight` | A `Weight()` object. Required if the retention model depends on weight-at-age. |
| `Maturity` | A `Maturity()` object. Required if the retention model uses relative selectivity. |
| `Selectivity` | A populated `Selectivity()` object. Optional, but recommended whenever the retention model is defined at length or weight: used to weight the age-length key when collapsing retention-at-length (or -weight) to retention-at-age, so that the result reflects the length distribution of fish the fleet actually catches at each age rather than the unconditional population length distribution at age. Ignored for retention models defined directly at age. |
| `nSim` | Integer. Number of simulation replicates. |
| `Years` | Numeric vector of model years. |
| `nArea` | Integer. Number of spatial areas. |
| `CalcAtLength` | Logical; if `TRUE`, calculates mean-at-length from age-based arrays. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical; if `TRUE`, suppresses informational messages. |
| `force` | Logical; if `TRUE`, forces re-population even if digest indicates object is current. |
| `replace` | Used internally. |
| `ASKOverride` | Used internally. |

## Details

`PopulateRetention()` handles population of retention-at-age, retention- at-length, or retention-at-weight. Steps include:

- Resolving the retention model class
- Generating mean retention at age, length, or weight as appropriate
- Converting between mean-at-length/weight and mean-at-age
- Adding stochastic variation
- Adding an 'Area' dimension and setting dimension names

## Value

A populated `Retention()` object.

## Examples

```
## Not run:
Ret <- Retention()
Ret_pop <- PopulateRetention(
  Ret, Ages = Ages, Length = Length, Weight = Weight,
```

```
  Maturity = Maturity, Years = 2000:2025, nSim = 5
)

## End(Not run)
```

---

`PopulateSelectivity` *Populate a Selectivity Object*

---

## Description

Populate a `Selectivity` object by generating selectivity-at-age, selectivity-at-length, or selectivity-at-weight values for a fleet.

## Usage

```
PopulateSelectivity(
  Selectivity,
  Ages = NULL,
  Length = NULL,
  Weight = NULL,
  Maturity = NULL,
  nSim = 5,
  Years = NULL,
  nArea = 1,
  CalcAtLength = TRUE,
  seed = NULL,
  silent = FALSE,
  CheckMaxValue = TRUE,
  replace = FALSE,
  ASKOverride = NULL
)
```

## Arguments

| | |
|---|---|
| `Selectivity` | A `Selectivity()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. |
| `Length` | A `Length()` object. Required if the selectivity model depends on length-at-age. |
| `Weight` | A `Weight()` object. Required if the selectivity model depends on weight-at-age. |
| `Maturity` | A `Maturity()` object. Required if `isRel = TRUE` to scale selectivity relative to maturity `L50`. |
| `nSim` | Integer. Number of simulation replicates. |
| `Years` | Numeric vector of model years. |
| `nArea` | Integer. Number of spatial areas. |

| | |
|---|---|
| `CalcAtLength` | Logical. If `TRUE`, calculate selectivity-at-length values from age-based arrays. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `CheckMaxValue` | Logical. If `TRUE`, ensure selectivity has a maximum value of 1 across age classes. |
| `replace` | Used interally. |
| `ASKOverride` | Used internally. |

## Details

`PopulateSelectivity()` performs the following steps:

- Structures selectivity parameters across simulation replicates, years, and areas.
- Resolves the selectivity model class and applies relative scaling to maturity if `isRel = TRUE`.
- Generates mean selectivity-at-age, selectivity-at-length, or selectivity-at-weight depending on the model.
- Converts between mean-at-length/weight and mean-at-age.
- Ensures maximum selectivity does not exceed 1 if `CheckMaxValue = TRUE`.
- Adds an area dimension to all arrays and sets dimension names.

## Value

A populated `Selectivity()` object.

## Examples

```
## Not run:
S <- Selectivity()
S_pop <- PopulateSelectivity(
  Selectivity = S,
  Ages = Ages,
  Length = Length,
  Weight = Weight,
  Maturity = Maturity,
  Years = 2000:2025,
  nSim = 50,
  nArea = 5
)

## End(Not run)
```

`PopulateSpatial` *Populate a Spatial Object*

## Description

Populate a `Spatial` object by validating or generating spatial structure for a stock, including movement matrices, unfished distributions, and relative area size.

## Usage

```
PopulateSpatial(
  Spatial,
  Ages = NULL,
  Years = NULL,
  nSim = 5,
  seed = NULL,
  silent = FALSE,
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| `Spatial` | A `Spatial()` object to populate. |
| `Ages` | An `Ages()` object defining age classes. |
| `Years` | Numeric vector of model years. |
| `nSim` | Integer. Number of simulation replicates. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `force` | Logical. If `TRUE`, force re-population even if the object digest is unchanged. |

## Details

`PopulateSpatial()` handles population of spatial structure for a stock. Steps include:

- Creating a default non-spatial structure if the object is empty
- Validating supplied movement matrices and their dimensions
- Calculating unfished spatial distributions
- Generating movement probabilities and movement matrices when required
- Expanding spatial objects across simulations and years

If a movement matrix is supplied, it is used to calculate unfished distribution.

## Value

A populated `Spatial()` object.

### See Also

Populate(), PopulateStock(), PopulateFleet()

### Examples

```
## Not run:
Spatial <- PopulateSpatial(
  Spatial,
  Ages = Ages,
  Years = Years,
  nSim = 100,
  seed = 123
)

## End(Not run)
```

---

## PopulateSRR *Populate an SRR Object*

---

### Description

Populate a SRR object by generating recruitment deviations, and structuring values across simulation replicates and years.

### Usage

```
PopulateSRR(
  SRR,
  Ages = NULL,
  CurrentYear = NULL,
  Years = NULL,
  nSim = 5,
  seed = NULL,
  silent = FALSE,
  force = FALSE
)
```

### Arguments

| | |
|---|---|
| SRR | A SRR() object to populate. |
| Ages | An Ages() object defining age classes. |
| CurrentYear | Numeric. Last historical year of the operating model. |
| Years | Numeric vector of model years. |
| nSim | Integer. Number of simulation replicates. |
| seed | Integer. Random seed used for stochastic generation. |
| silent | Logical. If TRUE, suppress informational messages. |
| force | Logical. If TRUE, force re-population even if the object digest is unchanged. |

### Details

`PopulateSRR()` handles population of stock-recruitment parameters. Steps include:

- Structuring model parameters across `nSim` and `Years`
- Resolving the SRR model class and relative recruitment function
- Initializing `R0`, `SD`, and `AC` parameters
- Generating historical and projection recruitment deviations

### Value

A populated `SRR()` object.

### See Also

`Populate()`, `PopulateStock()`

### Examples

```
## Not run:
# Assuming `Ages` object exists
SRR_obj <- SRR()
SRR_pop <- PopulateSRR(
  SRR_obj,
  Ages = Ages,
  CurrentYear = 2025,
  Years = 2000:2030,
  nSim = 10,
  seed = 123
)

## End(Not run)
```

---

## `PopulateStock` *Populate a Stock Object*

---

### Description

Populate a `Stock` object by generating stochastic biological quantities, expanding values across simulation replicates and years, and filling derived age- and length-based components.

### Usage

```
PopulateStock(
  Stock,
  nYear,
  pYear,
  CurrentYear = NULL,
```

```
  nSim = 5,
  Seasons = 1,
  ALK = TRUE,
  AWK = TRUE,
  seed = 102,
  silent = FALSE,
  force = FALSE,
  CalcAtLength = TRUE
)
```

## Arguments

| | |
|---|---|
| `Stock` | A `Stock()` object to populate. |
| `nYear` | Integer. Number of historical years. |
| `pYear` | Integer. Number of projection years. |
| `CurrentYear` | Numeric or character. Calendar year of the final historical year. Defaults to the current system year if `NULL`. |
| `nSim` | Integer. Number of simulation replicates. |
| `Seasons` | Integer. Number of seasons per year. |
| `ALK` | Logical. If `TRUE`, populate age–length keys. |
| `AWK` | Logical. If `TRUE`, populate age–weight keys. |
| `seed` | Integer. Random seed used for stochastic generation. |
| `silent` | Logical. If `TRUE`, suppress informational messages. |
| `force` | Logical. If `TRUE`, force re-population even if the object digest is unchanged. |
| `CalcAtLength` | Logical. If `TRUE`, calculate at-length quantities from at-age arrays where applicable |

## Details

`PopulateStock()` coordinates population of all biological components associated with a stock, including:

- Length-at-age via `PopulateLength()`
- Weight-at-age via `PopulateWeight()`
- Natural mortality via `PopulateNaturalMortality()`
- Maturity via `PopulateMaturity()`
- Fecundity via `PopulateFecundity()`
- Stock–recruitment via `PopulateSRR()`
- Spatial structure via `PopulateSpatial()`
- Initial depletion via `PopulateDepletion()`

To avoid unnecessary recomputation, a digest of the stock object and key arguments is checked before population. If the digest is unchanged and `force = FALSE`, the input object is returned unchanged.

### Value

A populated `Stock()` object.

### See Also

`Populate()`, `PopulateOM()`

### Examples

```
## Not run:
Stock <- PopulateStock(
  Stock,
  nYear = 40,
  pYear = 20,
  nSim = 100,
  seed = 123
)


## End(Not run)
```

---

## PopulateWeight *Populate a Weight-at-Age Object*

---

### Description

Populate a `Weight` object by generating stochastic weight-at-age values, expanding parameters across simulation replicates and years, and optionally constructing age–weight keys.

### Usage

```
PopulateWeight(
  Weight,
  Ages = NULL,
  Length = NULL,
  Years = NULL,
  nSim = 5,
  AWK = FALSE,
  CalcAtLength = TRUE,
  seed = NULL,
  silent = FALSE,
  force = FALSE
)
```

### Arguments

| | |
|---|---|
| Weight | A Weight() object to populate. |
| Ages | An Ages() object defining age classes. If NULL, defaults are inferred from Weight. |
| Length | A Length() object. Required if the weight model depends on length-at-age. |
| Years | Numeric vector of model years. If NULL, defaults are inferred from Weight. |
| nSim | Integer. Number of simulation replicates. |
| AWK | Logical. If TRUE, populate age–weight keys. |
| CalcAtLength | Logical. If TRUE, calculate weight-at-length values from weight-at-age arrays. |
| seed | Integer. Random seed used for stochastic generation. |
| silent | Logical. If TRUE, suppress informational messages. |
| force | Logical. If TRUE, force re-population even if the object digest is unchanged. |

### Details

PopulateWeight() handles weight-at-age population for a stock, including stochastic generation and propagation across simulations and years. Steps include:

- Structuring model parameters across nSim and Years
- Resolving the weight model class
- Generating mean weight-at-age or weight-at-length values
- Converting between mean-at-length and mean-at-age as needed
- Optionally constructing age–weight keys if AWK = TRUE
- Generating random variation and coefficients of variation

### Value

A populated Weight() object.

### See Also

Populate(), PopulateStock(), PopulateLength()

### Examples

```
## Not run:
Weight <- PopulateWeight(
  Weight,
  Ages = Ages,
  Length = Length,
  Years = Years,
  nSim = 100,
  AWK = TRUE,
  CalcAtLength = TRUE,
```

```
  seed = 123
)

## End(Not run)
```

---

`PPD` *Extract and Reconstruct Posterior Predicted Data (PPD)*

---

## Description

Extracts the projected data objects from an mse object.

## Usage

```
PPD(MSE)
```

## Arguments

| | |
|---|---|
| `MSE` | An object of class mse. |

## Details

To avoid duplication, historical data are stored only for the first MP. `PPD()` extracts the historical data from the first MP, and then prepends those data to the other (projection only) data objects stored in `MSE@PPD`

## Value

A named list of PPD (data objects) for each simulation, MP, and stock/complex.

## See Also

`JoinYear()`, `Years()`, mse

---

`Pplot` *A projection by projection plot of F/FMSY and B/BMSY*

---

## Description

A shorter version of the plot method for MSEs that just shows the projected trends in stock status and over exploitation

## Usage

```
Pplot(MSEobj, nam = NA, maxMP = 10, MPs = NA, maxsims = 20)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `nam` | Title of plot |
| `maxMP` | The maximum number of MPs to plot (defaults to the first 10) |
| `MPs` | A character vector of MPs to plot |
| `maxsims` | Integer, the maximum number of simulations to plot |

### Author(s)

T. Carruthers

---

## `Pplot2` *A projection by projection plot of F/FMSY, B/BMSY, B/B0, and yield*

---

### Description

A projection by projection plot of F/FMSY, B/BMSY, B/B0, and yield

### Usage

```
Pplot2(
  MSEobj,
  YVar = c("F_FMSY", "SSB_SSBMSY"),
  MPs = NA,
  sims = NULL,
  traj = c("all", "quant", "both"),
  quants = c(0.1, 0.9),
  incquant = TRUE,
  quantcol = "lightgray",
  RefYield = c("lto", "curr"),
  LastYr = TRUE,
  ref.lines = c(0.5, 1, 1.5),
  maxMP = 6,
  alpha = 60,
  cex.axis = 1,
  cex.lab = 1,
  YLab = NULL,
  incMP = TRUE,
  MPcex = 1,
  MPcol = "black",
  incLeg = TRUE,
  cex.leg = 1.5,
  legPos = "topleft",
  yline = NULL,
```

```
  xline = NULL,
  parOR = FALSE,
  xaxis = TRUE,
  yaxis = TRUE,
  oneIt = TRUE,
  ...
)
```

**Arguments**

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `YVar` | What to plot on the y-axis? Options are: `c('SSB_SSB0','SSB_SSBMSY', 'F_FMSY', 'Yield')` |
| `MPs` | Optional subset by MP |
| `sims` | Optional subset by simulation |
| `traj` | Plot all projections (`all`), only quantiles (`quant`), or both projections and median (`both`) |
| `quants` | Numeric vector of length 2 specifying the quantiles (e.g., 10th and 90th. Median is always included) |
| `incquant` | Logical. Include the quantiles or only plot median? |
| `quantcol` | Colour of the quantile polygon |
| `RefYield` | Should yield be relative to long-term optimum (`lto`) or last historical year (`curr`) |
| `LastYr` | Logical. Include the last historical year in the yield projections? |
| `ref.lines` | Numeric vector of y-values for horizontal reference lines. Set to NULL to remove lines. |
| `maxMP` | Maximum number of MPs to plot |
| `alpha` | Alpha for transparency of lines |
| `cex.axis` | Size of axis text |
| `cex.lab` | Size of axis label |
| `YLab` | Optional label for y-axis |
| `incMP` | Logical. Include name of MP? |
| `MPcex` | Size of MP label |
| `MPcol` | Optional character vector of colors for MP labels |
| `incLeg` | Logical. Include a legend? |
| `cex.leg` | Size of legend text |
| `legPos` | Legend position |
| `yline` | Optional horizontal lines |
| `xline` | Optional vertical lines |
| `parOR` | Logical to over-ride the par parameters |
| `xaxis` | Logical. Should x-axis labels be displayed? |
| `yaxis` | Logical. Should y-axis labels be displayed? |
| `oneIt` | Logical. Should one iteration be plotted on the quantile plot? |
| `...` | Additional arguments to be passed to plotting functions |

### Author(s)

T. Carruthers & A.Hordyk

---

`print.multiHist` *Print method for* `multiHist` *objects*

---

### Description

Print method for `multiHist` objects

### Usage

```
## S3 method for class 'multiHist'
print(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | For `print.multiHist`, a `multiHist` class object |
| `...` | Additional arguments (not used) |

### Value

Prints a summary of object `x` to the console

---

`print.PlotFleetList` *Print a* `PlotFleetList` *object*

---

### Description

Draws every panel in `x` to the active graphics device.

### Usage

```
## S3 method for class 'PlotFleetList'
print(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | A `PlotFleetList` object. |
| `...` | Additional arguments (currently ignored). |

### Value

Invisibly returns `x`.

---

`print.PlotStockList` *Print a* `PlotStockList` *object*

---

## Description

Draws every panel in `x` to the active graphics device.

## Usage

```
## S3 method for class 'PlotStockList'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `PlotStockList` object. |
| `...` | Additional arguments (currently ignored). |

## Value

Invisibly returns `x`.

---

`print.RepList` *Print a* `RepList` *object*

---

## Description

Print a summary of a `RepList` object, which is a list of outputs from `r4ss::SS_output`.

## Usage

```
## S3 method for class 'RepList'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A `RepList` object. |
| `...` | Additional arguments (currently ignored). |

## Value

Invisibly returns `x`.

`Project` *Project an Operating Model Over the Projection Period*

## Description

Runs the MSE projection loop for one or more management procedures (MPs) and returns a completed MSE-legacy object. Dispatches to the appropriate internal projection engine based on the class of `Hist`: hist objects use the new `om`-based engine; Hist-legacy objects use the legacy engine.

## Usage

```
Project(
  Hist,
  MPs = NULL,
  parallel = FALSE,
  silent = FALSE,
  nSim = NULL,
  Reduce = TRUE,
  extended = FALSE,
  checkMPs = FALSE
)
```

## Arguments

| | |
|---|---|
| `Hist` | A hist or Hist-legacy object containing the conditioned operating model and historical dynamics, as returned by `Simulate()`. |
| `MPs` | MPs to project: a character vector of names of MP functions available in the current environment, or (for hist objects) a named list of MP functions, optionally mixed with MP names, e.g. `list(IR5 = SetMPArgs(IndexRate, CalibYears = 5), 'IndexTarget')`. List names are used as the MP names. If `NULL` (default), projects `c("CurrentEffort", "CurrentCatch")`. |
| `parallel` | Logical or named list controlling parallel execution. For hist objects, `TRUE` splits the simulations into chunks and projects every MP x chunk combination in parallel on the active `SetupParallel()` plan. The number of chunks defaults to the number of workers; set `Hist@OM@Control$ProjectChunks` to change it. Results do not depend on the number of chunks. For Hist-legacy objects, `TRUE` runs all MPs in parallel, and a named list selects individual MPs to run in parallel. Default `FALSE`. |
| `silent` | Logical. Suppress progress messages if `TRUE`. Default `FALSE`. |
| `nSim` | Integer. If provided, reduces the number of simulations to `nSim` before projecting. Only used for hist objects. If `NULL` (default), all simulations in `Hist` are used. |
| `Reduce` | Logical. Reduce object size after simulation for memory efficiency? Default `TRUE`. See `ReduceDims()`. Only for hist objects. |

| | |
|---|---|
| `extended` | Logical. If `TRUE`, stores full age- and area-structured arrays for all years in `MSE@Misc$extended`. Only used for Hist-legacy objects. Substantially increases object size. Default `FALSE`. |
| `checkMPs` | Logical. Validate MP names and availability before projecting. Only used for Hist-legacy objects. Default `FALSE`. |

### Details

For hist objects, `Hist@OM@Control` settings used in projections (e.g. `ProjectChunks`, `EffortOptim`, `DataOM`) are described in OMControl.

### Value

An mse or a MSE-legacy object containing projection results for all MPs.

### See Also

`Simulate()`, `runMSE()`, OMControl

---

## `PWhisker` *Performance Whisker Plot*

---

### Description

A NAFO / ICCAT / SSB style MSE performance whisker plot

### Usage

```
PWhisker(MSEobj)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |

### Value

A box plot of performance

### Author(s)

T. Carruthers

`quantile_plot` *A quantile plot*

## Description

Plots quantiles and simulations for a stochastic time-series variable

## Usage

```
quantile_plot(
  datmat,
  xvals,
  p = c(0.05, 0.25, 0.5, 0.75, 0.95),
  tcol,
  ylim,
  sims = 1:3,
  refline = NA,
  dox = F,
  doy = F
)
```

## Arguments

| | |
|---|---|
| `datmat` | Matrix of real values with dimensions (simulation, year) (e.g. SB/SBMSY) |
| `xvals` | Vector of numerical values of length ncol(datmat). The xaxis labels for datmat. |
| `p` | Vector of quantiles five positions long. Defaults to c(0.05,0.25,0.5,0.75,0.95) so the 90% and 50% intervals with the median plotted in white. |
| `tcol` | Color of shaded regions (transparent) |
| `ylim` | Numerical vector of length 2, lower and upper limits for the yaxis |
| `sims` | Vector of positive integers, the individual simulations to plot |
| `refline` | Positive real number, a reference line to plot (on scale of y axis) |
| `dox` | Logical, should the x axis labels be plotted. |
| `doy` | Logical, should the y axis labels be plotted. |

## Author(s)

T. Carruthers

`rDirichletMultinomial` *Dirichlet-Multinomial Sampler*

## Description

Draw a single composition sample from the Dirichlet-Multinomial distribution. The Dirichlet-Multinomial arises as a multinomial draw whose probability vector is itself drawn from a Dirichlet distribution, producing compositions that are overdispersed relative to a standard multinomial.

## Usage

```
rDirichletMultinomial(n, alpha)
```

## Arguments

| | |
|---|---|
| `n` | Integer. Total count (sample size) for the multinomial draw. |
| `alpha` | Numeric vector of positive Dirichlet concentration parameters, length `nBin`. The relative magnitudes determine the expected composition. |

## Details

### Model:

Given concentration vector $\boldsymbol{\alpha}$ and total count $n$:

$$\mathbf{p} \sim \text{Dirichlet}(\boldsymbol{\alpha})$$

$$\mathbf{y} \sim \text{Multinomial}(n,\, \mathbf{p})$$

The marginal distribution of $\mathbf{y}$ is Dirichlet-Multinomial with mean $n \cdot \boldsymbol{\alpha} / \sum \boldsymbol{\alpha}$ and variance exceeding the multinomial by a factor of $(n + \sum \boldsymbol{\alpha})/(1 + \sum \boldsymbol{\alpha})$.

The Dirichlet draw is obtained via the standard gamma representation: independent $\text{Gamma}(\alpha_b, 1)$ draws normalised to sum to 1.

### Relationship to `ESS` and `Theta`:

In the composition observation model (see `CompObs()`), the concentration vector is constructed as:

$$\alpha_b = \text{ESS} \cdot \Theta \cdot q_b \cdot \exp(\text{Shift}_b)$$

where $\mathbf{q}$ is the vector of OM-predicted proportions. Passing this as `alpha` and `SampleSize` as `n` recovers the full composition observation model.

- `Theta = 1`, **no** `Shift`: recovers a near-multinomial draw (Dirichlet variance shrinks as `ESS` grows).
- `Theta < 1`: shrinks `alpha`, increasing overdispersion.
- `Shift` **non-**`NULL`: tilts the expected composition bin-by-bin on the log-concentration scale before drawing.

### Value

Integer vector of length `nBin` summing to `n`.

### See Also

`CompObs()`

### Examples

```
# Near-multinomial draw (high ESS, Theta = 1, no shift)
q <- c(0.1, 0.3, 0.4, 0.2)
alpha <- 200 * 1 * q
rDirichletMultinomial(n = 500, alpha = alpha)

# Overdispersed draw (low Theta)
alpha_od <- 200 * 0.3 * q
rDirichletMultinomial(n = 500, alpha = alpha_od)

# With shift - tilts expected composition toward older ages
shift <- c(-0.5, -0.2, 0.2, 0.5)
alpha_shifted <- 200 * 1 * q * exp(shift)
rDirichletMultinomial(n = 500, alpha = alpha_shifted)
```

---

## RealFease *MP feasibility diagnostic using real data*

---

### Description

What MPs do not return NAs from the real data

### Usage

```
RealFease(Data = NULL)
```

### Arguments

| | |
|---|---|
| `Data` | An object of class 'Data'. Optional. If Data object is included, the returned MPs are both feasible (in terms of management) and possible (sufficient data to run MP) |

### Value

a vector of MP names that calculate without errors for the specific data.

### Author(s)

T. Carruthers

---

`Rec-class` *Class* `'Rec'`

---

### Description

An object for storing the MP recommendations

### Slots

`TAC` A numeric value with the TAC recommendation

`Effort` A numeric value with the effort recommendation as a fraction of current (nyear) fishing effort

`Spatial` A boolean vector of length 'nareas' specifying if area is open (1) or closed (0) to fishing

`Allocate` A boolean value describing if effort should be re-allocated from close to open areas

`LR5` smallest length at 5 per cent retention - in absolute units - i.e same units as Linf and L50

`LFR` smallest length at full retention - in absolute units - i.e same units as Linf and L50

`HS` upper harvest slot (no retention above this) - in absolute units - i.e same units as Linf and L50

`Rmaxlen` retention of the largest size class - fraction between 0 and 1

`L5` smallest length at 5 per cent selection - in absolute units - i.e same units as Linf and L50

`LFS` smallest length at full selection - in absolute units - i.e same units as Linf and L50

`Vmaxlen` selection of the largest size class - fraction between 0 and 1

`Fdisc` fraction of discarded fish that die - fraction between 0 and 1

`DR` Discard rate - the fraction of caught fish that are discarded

`Misc` An empty list that can be used to store information and pass on to MPs in future

### Objects from the Class

Objects can be created by calls of the form `new('Rec')`

### Author(s)

A. Hordyk

`ReduceDims` *Reduce redundant Sim, Year, and Age dimensions in arrays*

## Description

Reduce the size of array-like objects by collapsing dimensions (`Sim`, `Year`, and/or `Age`) when their values are identical across that dimension. The function works recursively on S4 objects and lists, applying the reduction to all array-valued components.

## Usage

```
ReduceDims(
  array,
  IncSim = TRUE,
  IncAge = FALSE,
  IncYear = FALSE,
  debug = FALSE
)
```

## Arguments

| | |
|---|---|
| `array` | An array, list, or S4 object. For non-array objects, the function is applied recursively to all components or slots. |
| `IncSim` | Logical; reduce the `Sim` dimension if present and all simulations are identical. |
| `IncAge` | Logical; reduce the `Age` dimension if present and all ages are identical. |
| `IncYear` | Logical; reduce the `Year`dimension if present and all years are identical or by keeping only unique years. |
| `debug` | Logical. Print debug messages? |

## Details

Reduction is controlled independently for each supported dimension. If a dimension is excluded or not present, it is left unchanged.

For arrays, the function detects dimensions named `Sim`, `Age`, and `Year`. If enabled, dimensions are reduced as follows:

- `Sim`: kept at length 1 when all simulations are identical.
- `Age`: kept at length 1 when all ages are identical.
- `Year`: kept at length 1 when all years are identical; otherwise, only unique years are retained.

## Value

An object of the same class as `array`, with reduced dimensions where applicable.

### See Also

ReduceNSim()

---

`ReduceNSim` *Reduce the number of simulations*

---

### Description

Subsets an OM() or Hist() object by reducing number of simulations to `nSim`

### Usage

```
ReduceNSim(object, nSim = NULL)
```

### Arguments

| | |
|---|---|
| `object` | An OM() or Hist() object |
| `nSim` | Integer. The maximum number of simulations |

### Value

The OM() or Hist() with reduced `nSim`

### See Also

Reduce()

### Examples

```
OM <- Populate(SingleStockOM)
nSim(OM)
OM |> Stock(1) |> Length() |> MeanAtAge() |> dim()

OM_reduced <- ReduceNSim(OM, 2)
nSim(OM_reduced)
OM_reduced |> Stock(1) |> Length() |> MeanAtAge() |> dim()
```

## reference-class *Reference Points Object*

### Description

The `reference` class stores biological and management reference points derived from per-recruit, equilibrium, or dynamic population analyses. These reference points are typically used for management strategy evaluation, harvest control rules, and performance metrics.

### Slots

`SPR0` List or array of unfished spawning potential ratio (SPR0), by stock and simulation.

`MSY` An object containing MSY-based reference points such as FMSY, BMSY, and MSY.

`Landings` List or array of reference landings values.

`Removals` List or array of reference removals values.

`F01` List or array of fishing mortality corresponding to the 0.1 slope reference point.

`FMax` List or array of fishing mortality at maximum yield-per-recruit.

`FSPR` List or array of fishing mortality at user-specified spawning-potential-ratio targets, with a `Target` dimension.

`FMed` List or array of fishing mortality at the empirical replacement line (median historical recruits-per-spawner).

`FCrash` List or array of fishing mortality leading to population collapse.

`SPRcrash` List or array of spawning potential ratio at collapse.

`MGT` List or array of mean generation time reference points.

`BLow` List or array defining the B-Low biomass reference points.

`Misc` Miscellaneous list for additional reference quantities or diagnostics.

## referencedata-class `referencedata` *Class*

### Description

Stores biological and management reference points by extending refpointsMSY. Used in the `Reference` slot of a data object.

### Usage

```
ReferenceData()
```

### Details

This class inherits all slots from refpointsMSY (e.g., MSY, FMSY, BMSY) and appends a `Misc` slot for any additional reference point data not covered by the parent class.

### Value

`ReferenceData()` returns a `referencedata` object.

### Slots

`Misc` A named list for any additional reference point metadata not covered by the inherited refpointsMSY slots.

`ReferenceData()` creates a new `referencedata` object.

### See Also

data, `Data()`, refpointsMSY

---

## `ReferenceMPs` *Reference Management Procedures*

---

### Description

Diagnostic management procedures that fish at fixed fractions of `FMSY` (effort-based) or `MSY` (TAC-based), using true operating-model reference points.

### Usage

```
refFMSY(Data)

refFMSY75(Data)

refFMSY50(Data)

refMSY(Data)

refMSY75(Data)

refMSY50(Data)

refFCurr(Data)

NoFishing(Data)

ReferenceMPs()
```

### Arguments

| | |
|---|---|
| `Data` | A data object containing historical observations. |

### Details

refFMSY, refFMSY75, refFMSY50 set effort so that realised F equals 100%, 75%, or 50% of FMSY, and default to being applied every year (like any MP, this can be overridden via OM@Interval).

refMSY, refMSY75, refMSY50 set TAC to 100%, 75%, or 50% of MSY (landings plus discards). MSY is an annual quantity, so on a seasonal OM (OM@Seasons > 1) the TAC returned at each call is MSY divided by the number of seasons, since Advice() returned by an MP is reapplied unchanged at every timestep until the MP is next called.

refFCurr fixes F at its last historical value. It is currently a wrapper around CurrentEffort(), since advice-class has no way to specify apical F directly. In spatial models, or where catchability (q) drifts over time, realised F can diverge from the last historical effort. It will be updated to set F directly once advice-class supports it.

NoFishing sets Effort to 0 every year

FMSY/MSY are computed once, during Simulate(), at the terminal historical year (or OM@Control$RefYears) and held fixed for the whole projection; i.e., they are not recalculated per projection year.

If reference points are not available for an OM each reference MP returns empty advice and a single warning is issued the first time it is called.

The refFMSY* and refMSY* MPs read reference points from Data@Misc$DataOM@Reference, requested by their attr(MP, 'DataOM') <- list(Reference = TRUE). A custom MP that calls one of them needs Reference as well, via its own DataOM attribute or OM@Control$DataOM; see OMControl.

In spatial OMs, fleet targeting means the realised F from refFMSY* may differ from the specified fraction of FMSY, because the FMSY calculation does not account for spatial structure.

### Value

An advice object.

### See Also

Advice(), data, advice, CalcMSY(), F_FMSY(), CurrentEffort()

---

## refpoints-accessors *Access Per-Recruit and Replacement-Line Reference Point Slots*

---

### Description

Accessor functions for reference points stored in Hist@Reference, populated by CalcRefPoints().

### Usage

```
F01(Hist)

FMax(Hist)

FSPR(Hist)

FMed(Hist)

FCrash(Hist)

SPRcrash(Hist)

BLow(Hist)
```

### Arguments

| | |
|---|---|
| `Hist` | A hist object. |

### Details

- `F01()`: apical F at 10% of the yield-per-recruit slope at the origin.
- `FMax()`: apical F maximising yield-per-recruit.
- `FSPR()`: apical F at each requested SPR target.
- `FMed()`: apical F at the median historical replacement line.
- `FCrash()`: apical F at which the stock can no longer replace itself.
- `SPRcrash()`: spawning potential ratio at `FCrash()`.
- `BLow()`: spawning biomass rebuilding threshold (see `CalcBLow()`). Only populated when computed explicitly, since it is substantially more expensive than the other reference points here.

### Value

A numeric array with dimensions `Sim x Stock x Year` (`FSPR()` additionally has a `Target` dimension). Returns `NULL` if the slot has not yet been populated (i.e. `CalcRefPoints()` has not been run).

### See Also

`CalcRefPoints()`, `CalcBLow()`, reference

## refpointsMSY-class *The* refpointsMSY *S4 Class*

### Description

An S4 class storing MSY-based biological reference points. Produced internally during reference point calculations and not intended to be modified directly by users.

### Details

**Reference point calculations:**

MSY reference points are derived from the per-recruit quantities in a perrecruit object by identifying the apical fishing mortality that maximises total removals (or optionally landing), accounting for the stock-recruitment relationship.

All slots share the same `Sim × Stock × Year` dimensions, where the `Year` dimension reflects the year(s) for which biological and fishery parameters were evaluated.

### Slots

`FMSY` Fishing mortality rate at MSY (`Sim × Stock × Year`).

`BMSY` Total biomass at MSY (`Sim × Stock × Year`).

`SBMSY` Spawning biomass at MSY (`Sim × Stock × Year`).

`SPMSY` Spawning production at MSY (`Sim × Stock × Year`).

`SPRMSY` Spawning potential ratio at MSY (`Sim × Stock × Year`).

`MSYLandings` Landed catch at MSY (`Sim × Stock × Year`).

`MSYDiscards` Dead discards at MSY (`Sim × Stock × Year`).

`Misc` List reserved for internal use. #'

### See Also

perrecruit

## RefSeason *Reference Point Season Handling for OM Objects*

### Description

Access or replace the `RefSeason` and `RefEffortYears` slots of an om object. Only relevant for seasonal OMs (`Seasons > 1`): both control how equilibrium/MSY reference points (e.g. `BMSY`, `SBMSY`, `B0`, `SB0`, `BLow`) are calculated from a seasonal per-recruit model. Also accepts hist and mse objects, extracting the embedded `OM` slot transparently.

### Usage

```
RefSeason(x)

RefSeason(x) <- value

RefEffortYears(x)

RefEffortYears(x) <- value
```

### Arguments

| | |
|---|---|
| `x` | An om, hist, or mse object. |
| `value` | Replacement value; an integer vector of season indices (`RefSeason`, in `1..Seasons`) or a numeric vector of historical calendar years (`RefEffortYears`). `NULL` restores the default behaviour described in *Details*. |

### Details

`RefSeason`**:**

Reference point biomass quantities are a cross-sectional snapshot within the annual cycle, so a season (or seasons) must be chosen at which to take that snapshot. `NULL` (default) auto-detects, independently per simulation, which season(s) have nonzero spawning contribution, and averages the cross-sectional snapshot across them when more than one is detected. Set `RefSeason` explicitly to override the auto-detection with a fixed season index/indices.

`RefEffortYears`**:**

One or more historical calendar years whose relative seasonal effort/catchability pattern fixes the seasonal *shape* of fishing mortality used during per-recruit and MSY reference point optimization - decoupled from the year used for the biological parameters themselves. `NULL` (default) reuses the same year as the biological parameters. When more than one year is given, the per-season effort is averaged across those years before use.

### Value

- `RefSeason()`/`RefEffortYears()` return the value of the corresponding slot.
- `RefSeason<-()`/`RefEffortYears<-()` return `x` with the corresponding slot updated.

### See Also

- `OM()` for the constructor, om for the class definition.
- `CalcEquilibrium()`, `CalcMSY()` for the reference point calculations that use these slots.
- OM-accessors for the remaining `OM` slot accessors.

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
om <- OM(Seasons = 4)
RefSeason(om) <- 1L
RefSeason(om)
```

---

## relative_ref *Biomass and Fishing Mortality Relative to Reference Points*

---

### Description

Extract time series of biomass, spawning biomass, spawning production, or fishing mortality expressed as a fraction of an unfished or MSY reference level.

### Usage

```
B_B0(
  object,
  type = c("Equilibrium", "Dynamic"),
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

SB_SB0(
  object,
  type = c("Equilibrium", "Dynamic"),
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

SP_SP0(
  object,
  type = c("Equilibrium", "Dynamic"),
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

B_BMSY(
```

```
  object,
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

SB_SBMSY(
  object,
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

SP_SPMSY(
  object,
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)

F_FMSY(
  object,
  df = TRUE,
  Reduce = TRUE,
  IncYear = FALSE,
  Extend = FALSE,
  silent = TRUE
)
```

## Arguments

| | |
|---|---|
| `object` | A hist or mse object. |
| `type` | Character. One of `'Equilibrium'` or `'Dynamic'`. Controls which unfished baseline is used. See `BO()` for details. Only applies to the `*_*0` family. |
| `df` | Logical. If `FALSE` returns the raw ratio array. If `TRUE` (default) returns a tidy `data.frame`. |
| `Reduce` | Logical. If `TRUE` (default) simulation dimensions are reduced using `ReduceDims()` before conversion to a data frame. |
| `IncYear` | Logical. Passed to `ReduceDims()`; controls whether the year dimension is retained during reduction. Default `FALSE`. |

| | |
|---|---|
| `Extend` | Logical. If `TRUE`, rows sharing one value across every simulation (e.g. the historical period, or any deterministic quantity – see `Reduce`/`ReduceDims()`) are broadcast to all `nSim` simulations via `ExtendSims()`, rather than appearing once with `Sim = 1`. Default `FALSE`. Ignored when `df = FALSE`. |
| `silent` | Logical. If `FALSE` and `Extend = FALSE`, emits a message when the returned data frame has rows sharing one value across every simulation, pointing at `Extend = TRUE`. Default `TRUE`. Ignored when `df = FALSE`. |

## Details

`B_B0()`, `SB_SB0()`, and `SP_SP0()` divide the time series by the corresponding unfished level (`type = 'Equilibrium'` or `'Dynamic'`). `B_BMSY()`, `SB_SBMSY()`, and `SP_SPMSY()` divide by the corresponding MSY reference point. `F_FMSY()` divides total apical fishing mortality (summed over fleets) by `FMSY`.

When applied to an mse object the historical and projection periods are row-bound and labelled via the `Period` column; historical rows carry `MP = "Historical"`.

For the projection period of an mse object the denominator is extended to cover projection years by repeating the reference value from the last historical year. In seasonal models, each projection time step uses the value from the same season of the last historical year.

**Seasonal models:**

In seasonal models (`Seasons > 1`) the MSY reference points are annual quantities, so `B_BMSY()`, `SB_SBMSY()`, `SP_SPMSY()`, and `F_FMSY()` return one value per complete calendar year, with `Year` holding the calendar year:

- `F_FMSY()`: apical fishing mortality summed over the seasons of each calendar year, the same annual basis as `FMSY` (see `CalcMSY()`).
- `B_BMSY()`, `SB_SBMSY()`, `SP_SPMSY()`: biomass or spawning production in the reference season(s) of each calendar year (see `RefSeason()`), the same snapshot used for `BMSY`, `SBMSY`, and `SPMSY`.

`B_B0()`, `SB_SB0()`, and `SP_SP0()` are reported per time step. The per-time-step series are available from `Biomass()`, `SBiomass()`, `SProduction()`, and `FDead()`.

`FMSY` is a single "apical fishing mortality" optimised jointly across all stocks in a complex (see `CalcMSY()`) – there is no separate `FMSY` for an individual stock within a multi-stock complex. `F_FMSY()` therefore reports its ratio at the *complex* level (`OM@Complexes`), not the stock level: the numerator is the controlling (maximum) realised apical F across the stocks in each complex – the same quantity `CalcMSY()` itself optimises against – divided by that complex's `FMSY`. For a complex containing a single stock this is identical to the stock's own F/FMSY. The `Stock` column of the returned `data.frame` (and dimension of the `df = FALSE` array) holds complex names, matching the convention already used for `FMSY` in refpointsMSY.

## Value

- `df = FALSE`: a numeric array of ratios with dimensions `Sim × Stock × Year`.
- `df = TRUE`: a tidy `data.frame` with columns `Sim`, `Stock`, `Year`, `Period`, `MP` (MSE only), `Value`, `Variable`.

### See Also

Biomass(), B0(), BMSY(), FDead(), FMSY(), ExtendSims()

### Examples

```
Hist <- Simulate(SingleStockOM)
B_B0(Hist)
SB_SB0(Hist, type = 'Dynamic')
F_FMSY(Hist)

MSE <- Project(Hist, 'CurrentEffort')
B_BMSY(MSE, df = TRUE)
SB_SBMSY(MSE, df = TRUE)
SP_SPMSY(MSE, df = TRUE)
F_FMSY(MSE, df = TRUE)
```

---

## Rename *Rename Stocks and Fleets in an OM*

---

### Description

Renames stocks and/or fleets throughout an operating model (OM) object, updating all names recursively across S4 slots, lists, and array dimension names.

### Usage

```
Rename(object, Stocks = NULL, Fleets = NULL)
```

### Arguments

| | |
|---|---|
| `object` | An object of class `om`. |
| `Stocks` | Optional named character vector or named list mapping new stock names to current names (e.g. `c('new_name' = 'old_name')`). If `NULL`, stock names are unchanged. |
| `Fleets` | Optional named character vector or named list mapping new fleet names to current names. If `NULL`, fleet names are unchanged. |

### Value

The input `OM` object with stock and/or fleet names updated throughout.

Renaming is applied recursively: list element names, S4 slot names, and the `Stock` and `Fleet` dimensions of arrays are all updated wherever a match is found. Names not present in `Stocks` or `Fleets` are left unchanged, so partial lookups are safe.

The lookup arguments use the convention `NewName = OldName`, e.g.: `Fleets = list(NewFleet = 'OldFleet')`.

### See Also

Reorder()

### Examples

```
OM <- SingleStockOM
StockNames(OM)
FleetNames(OM)

OM2 <- Rename(OM,
              Stocks = list('New Stock Name' = 'Example Stock'),
              Fleets = list('New Fleet Name' = 'Example Fleet')
)

StockNames(OM2)
FleetNames(OM2)
```

---

## Reorder *Reorder Stocks and Fleets in an OM*

---

### Description

Reorders stocks and/or fleets throughout an operating model (OM) object, updating all list elements and array dimension orders recursively across S4 slots.

### Usage

```
Reorder(object, Stocks = NULL, Fleets = NULL)
```

### Arguments

| | |
|---|---|
| `object` | An object of class `om`. |
| `Stocks` | Optional character vector of all stock names in the desired order. Must contain exactly the same names as currently in `OM`. If `NULL`, stock order is unchanged. |
| `Fleets` | Optional character vector of all fleet names in the desired order. Must contain exactly the same names as currently in `OM`. If `NULL`, fleet order is unchanged. |

### Details

Reordering is applied recursively: list element names, S4 slot names, and the `Stock` and `Fleet` dimensions of arrays are all reordered wherever a match is found. The full set of names must be provided — partial reordering is not supported.

### Value

The input `om` object with stocks and/or fleets reordered throughout.

### See Also

Rename()

### Examples

```
OM <- AddFleet(SingleStockOM, 'DummyFleet')
FleetNames(OM)

OM2 <- Reorder(OM, Fleets = rev(FleetNames(OM)))
FleetNames(OM2)
```

---

## Replace *Replace an existing Stock, Fleet, Obs, or Imp object*

---

### Description

A function that replaces a Stock, Fleet, Obs, or Imp object from an OM with one from another object.

### Usage

```
Replace(
  OM,
  from,
  Sub = c("Stock", "Fleet", "Obs", "Imp"),
  Name = NULL,
  silent = FALSE
)
```

### Arguments

| | |
|---|---|
| OM | An operating model object (class OM) which will be updated with a sub-model from another OM |
| from | An object of class OM, Stock, Fleet, Obs, or Imp to be replace the values in OM |
| Sub | A character string specifying what object type to replace (only used if from is class OM) "Stock", "Fleet", "Obs", or "Imp" (default is all four which is probably not what you want to do) |
| Name | Character. Name for the new OM object (OM@Name) |
| silent | Should messages be printed? |

### Value

An object of class OM

### Author(s)

A. Hordyk

### Examples

```
# Replace Stock
OM <- MSEtool::testOM
OM2 <- Replace(OM, Blue_shark)

# Replace Fleet
OM <- MSEtool::testOM
OM2 <- Replace(OM, Generic_DecE)

# Replace Fleet from another OM
# OM1 <- new("OM", Albacore, Generic_DecE, Perfect_Info, Overages)
# OM2 <- new("OM", Blue_shark, Generic_IncE, Generic_Obs, Perfect_Imp)
# OM1a <- Replace(OM1, OM2, "Fleet")
```

---

## replic8 — *Enlarge (replicate) a DLM data object to create an additional dimension for simulation / sensitivity testing*

---

### Description

Replicates position 1 data to multiple positions for sensitivity testing etc

### Usage

```
replic8(Data, nrep)
```

### Arguments

| | |
|---|---|
| Data | A data-limited methods data object |
| nrep | The number of positions to expand the DLM object to |

### Author(s)

T. Carruthers

`Report` *Generate a Data Report*

## Description

A HTML Data Report is generated and opened in a web browser

## Usage

```
Report(
  Data = NULL,
  md = NULL,
  name = "Data-Report",
  title = "Data Documentation",
  author = "Author Name",
  date = Sys.Date(),
  output_format = c("html_document", "pdf_document"),
  open = TRUE,
  quiet = TRUE,
  dir = NULL,
  overwrite = FALSE
)
```

## Arguments

| | |
|---|---|
| `Data` | Either an object of class `Data` or the file path to a valid file to be imported with `XL2Data` |
| `md` | Full file path to a valid text file documenting the Data |
| `name` | Optional. Name of the output file |
| `title` | Title for the Report. Title in the markdown file will override this value |
| `author` | Author of the Report. Author in the markdown file will override this value |
| `date` | Date of the Report. Date in the markdown file will override this value |
| `output_format` | Output file format: `html_document` or `pdf_document` |
| `open` | Logical. Open the compiled report? |
| `quiet` | Logical.An option to suppress printing of the pandoc command line. |
| `dir` | Optional. Directory to save the file. Defaults to `getwd()` |
| `overwrite` | Logical. Overwrite an existing file with the same name? |

## Value

Nothing. A Data Report is generated and saved in `dir`

### Author(s)

A. Hordyk

### Examples

```
## Not run:
DataInit('Example') # generate example Data Input and Documentation files
Report('Example', 'Example.md')

## End(Not run)
```

---

`ReqData` *ReqData*

---

### Description

Dataframe with required data slots for built-in MPs

### Usage

```
ReqData
```

### Format

An object of class `data.frame` with 123 rows and 2 columns.

---

`Required` *What management procedures need what data*

---

### Description

A function that finds all the MPs and searches the function text for slots in the Data object

### Usage

```
Required(funcs = NA, noCV = FALSE)
```

### Arguments

| | |
|---|---|
| `funcs` | A character vector of management procedures |
| `noCV` | Logical. Should the CV slots be left out? |

### Value

A matrix of MPs and their required data in terms of `slotnames('Data')`, and broad Data classes for each MP

### Author(s)

T. Carruthers

### See Also

Can Cant Needed MPtype Data-legacy

---

`Retention` *Retention*

---

### Description

Construct and manipulate a retention object defining retention-at-age, retention-at-length, or retention-at-weight for a `Fleet()` object. Retention is optional; if not specified, all age and length classes are assumed fully retained.

### Usage

```
Retention(
  Pars = list(),
  Model = NULL,
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  MeanAtWeight = NULL,
  Classes = NULL,
  isRel = FALSE,
  isAtLength = TRUE,
  Misc = list()
)

Retention(x) <- value
```

### Arguments

| | |
|---|---|
| `Pars` | Named list of retention parameters passed to the retention model function. Parameter names must match the arguments of the chosen model (see `RetentionModels()`). Each element may be a scalar, vector, or array; see Specifying Biological and Fleet Schedules for accepted formats. Default `list()` (empty).<br>If `Pars` and `Model` are both provided, `MeanAtAge` is computed from the model and any values already in `MeanAtAge` are overwritten. To preserve a user-specified `MeanAtAge`, leave `Pars = list()` (the default).<br>If `Pars` is a fleet object, the `Retention` slot of that fleet is returned. |
| `Model` | Character or function or `NULL`. Retention model identifier. If `NULL` (default), the model is inferred automatically from the names in `Pars`. May also be supplied as a custom R function with arguments matching those in `Pars`. See `RetentionModels()` for built-in options. |

`MeanAtAge` Numeric array or `NULL`. Mean retention-at-age with dimensions `Sim x Age x Year` (area dimension added during population). Used directly when `Pars` is empty and `MeanAtLength` is not supplied. See Specifying Biological and Fleet Schedules for accepted array formats. **Note:** if `Pars` and `Model` are both provided, any values supplied here will be overwritten during population.

`MeanAtLength` Numeric array or `NULL`. Mean retention-at-length with dimensions `Sim x Class x Year`. If provided and `Pars` is empty, takes precedence over `MeanAtAge`; `MeanAtAge` is derived from it via the age-length key. A numeric vector is also accepted and converted to a `1 x nClass x 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`.

`MeanAtWeight` Numeric array or `NULL`. Mean retention-at-weight with dimensions `Sim x Class x Year`. A numeric vector is also accepted and converted the same way as `MeanAtLength`, using the weight object's `AWK`. Default `NULL`.

`Classes` Numeric vector or `NULL`. Length or weight class midpoints corresponding to the second dimension of `MeanAtLength` or `MeanAtWeight`. Default `NULL`.

`isRel` Logical. If `TRUE`, length-based parameters (e.g., `LR5`, `LFR`) are interpreted as multiples of the length-at-50%-maturity (`L50`) of the paired stock rather than absolute length values. A `Maturity()` object must be available to `PopulateRetention()` for scaling to occur. Default `FALSE`.

`isAtLength` Logical. Whether `MeanAtLength`/`MeanAtWeight` (or an at-length/at-weight `Model`) reflect a genuinely length- or weight- retention schedule. Set `FALSE` when retention is only meaningfully defined at age (e.g. imported from an age-structured assessment) so that fleet weight-at-age calculations skip weighting by this object's at-length schedule and use the stock's plain weight-at-age instead. Default `TRUE`.

`Misc` List. Miscellaneous additional inputs. Default `list()`.

`x` A retention object, or a fleet object for `Retention<-`.

`value` For `Retention<-`: a retention object.

## Details

Retention is optional for all fleet objects. It defines the probability that a selected fish is retained, on a scale from 0 to 1. Fish that are selected but not retained are treated as discards, and their fate is determined by `DiscardMortality()`.

When no retention object is supplied, `PopulateRetention()` sets full retention (1) for all age and length classes.

### Specifying Retention:

There are two ways to specify retention:

**Model-based** (recommended): supply `Pars` as a named list whose element names match the arguments of a built-in or custom model function. The model is resolved automatically unless `Model` is specified explicitly. See `RetentionModels()` for available models and their

required parameters, and Specifying Biological and Fleet Schedules for how parameter values are structured across simulations and years.

Available model families are: logistic (at-age, at-length, at-weight, with optional `MaxRet` asymptote), knife-edge (at-age, at-length), and double-normal (at-length, at-weight). At-length and at-weight schedules are converted to at-age internally using the age-length or age-weight key.

**Direct array**: leave `Pars = list()` and supply `MeanAtAge`, `MeanAtLength`, or `MeanAtWeight` directly. If `MeanAtLength` is provided, it takes precedence and `MeanAtAge` is derived from it. If only `MeanAtAge` is provided, `MeanAtLength` is calculated from it unless already populated.

These two approaches are mutually exclusive: if `Pars` is non-empty and `Model` can be resolved, any values in `MeanAtAge` will be overwritten.

**Relative Parameters:**

When `isRel = TRUE`, length-based parameters (e.g., `LR5`, `LFR`) are scaled by the maturity `L50` of the paired stock before the retention curve is computed.

**Attaching to a Fleet:**

A retention object can be attached to a `Fleet()` with `Retention(Fleet) <- MyRetention` and retrieved with `Retention(Fleet)`.

Individual slots may be accessed or modified using `Pars()`, `Model()`, `MeanAtAge()`, `MeanAtLength()`, `MeanAtWeight()`, and `Classes()`.

## Value

- `Retention()` returns a retention object. If `Pars` is a fleet object, the `Retention` slot of that fleet is returned.
- `Retention<-` returns `x` with the `Retention` slot replaced by `value`.

## See Also

- retention for the class definition and slot-level documentation.
- `RetentionModels()` for available model functions and their parameters.
- `Fleet()` for the enclosing fleet constructor.
- `Selectivity()`, `DiscardMortality()` for related fleet components.
- Specifying Biological and Fleet Schedules for how `Pars`, `Model`, and `MeanAt*` arrays are structured.
- `PopulateRetention()` for population details.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## Examples

```
## ---- List available retention models ----
RetentionModels()
```

```
## ---- Default: full retention ----
# An empty Retention object causes PopulateRetention() to set retention = 1
# for all age and length classes (all fish retained).
r_full <- Retention()
r_full

## ---- Double-normal retention-at-length (asymptotic) ----
# LR5: length at 5% retention; LFR: length at full retention;
# Rmaxlen = 1 produces asymptotic (full) retention above LFR.
r_asymptotic <- Retention(
  Pars = list(LR5 = 15, LFR = 28, Rmaxlen = 1)
)

## ---- Double-normal retention-at-length (dome-shaped) ----
# Rmaxlen < 1 allows retention to decline at larger lengths, representing
# a slot limit or size-based discard regulation.
r_dome <- Retention(
  Pars = list(LR5 = 15, LFR = 28, Rmaxlen = 0.4)
)

## ---- Logistic retention-at-length ----
# RL50: length at 50% retention; RL50_95: increment to 95% retention.
# MaxRet (default 1) caps the asymptote, allowing partial retention.
r_logistic <- Retention(
  Pars = list(RL50 = 25, RL50_95 = 8)
)

## ---- Logistic retention with partial maximum retention ----
# MaxRet < 1 models a fishery where large fish are only partially retained
# even at full selectivity (e.g. voluntary high-grading or regulatory caps).
r_partial <- Retention(
  Pars = list(RL50 = 25, RL50_95 = 8, MaxRet = 0.7)
)

## ---- Logistic retention-at-age ----
r_age <- Retention(
  Pars = list(RA50 = 3, RA50_95 = 2)
)

## ---- Knife-edge retention-at-length ----
# Full retention at or above RL; zero below (e.g. minimum size limit).
r_knife <- Retention(
  Pars = list(RL = 25)
)

## ---- Relative parameters (isRel = TRUE) ----
# When isRel = TRUE, LR5 and LFR are interpreted as multiples of the
# maturity L50 of the paired stock. A Maturity object must be available
# to PopulateRetention() for scaling to occur.
r_rel <- Retention(
  Pars  = list(LR5 = 0.4, LFR = 0.8, Rmaxlen = 1),
  isRel = TRUE
)
```

```
## ---- Direct MeanAtAge array ----
# When Pars is empty (default), MeanAtAge can be supplied directly.
# Values are preserved because Pars is empty.
ages      <- 0:20
ret_vals  <- ifelse(ages >= 3 & ages <= 12, 1, 0)  # slot limit: retain ages 3-12

ret_array <- array(
  ret_vals,
  dim      = c(1, length(ages), 1),
  dimnames = list(Sim = 1, Age = ages, Year = 2000)
)

r_direct <- Retention(MeanAtAge = ret_array)
MeanAtAge(r_direct)

## ---- Attaching to / extracting from a Fleet object ----
f <- Fleet(Name = "Trawl")
Retention(f) <- Retention(Pars = list(LR5 = 15, LFR = 28, Rmaxlen = 1))
Retention(f)
```

---

`retention-class` *The* `retention` *S4 Class*

---

## Description

Defines retention-at-age, retention-at-length, or retention-at-weight for use in a fleet object. Retention is optional; if not specified, all age and length classes are assumed fully retained. Objects are typically created via the `Retention()` constructor, which documents all parameters in detail.

## Slots

`Pars` `list`. Named list of retention parameters passed to `Model`. See `Retention()` and `RetentionModels()`.

`Model` `function` or `character` or `NULL`. Retention model identifier or function. See `RetentionModels()`.

`isRel` `logical`, `numeric`, or `character`. Whether length-based parameters are expressed relative to maturity `L50`. See `Retention()`.

`MeanAtAge` `numeric` array or `NULL`. Mean retention-at-age (`Sim x Age x Year x Area`). See `Retention()`.

`MeanAtLength` `numeric` array or `NULL`. Mean retention-at-length (`Sim x Length x Year x Area`). Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). See `Retention()`.

`MeanAtWeight` `numeric` array or `NULL`. Mean retention-at-weight (`Sim x Weight x Year x Area`). Values are evaluated at the midpoint of each bin; the `Class` dimension is labelled by bin lower bounds (see `Classes`). See `Retention()`.

`Classes numeric` or `NULL`. Lower bounds of length or weight bins corresponding to the second dimension of `MeanAtLength` or `MeanAtWeight`. Bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength`/`MeanAtWeight` are evaluated at bin midpoints, not at these lower bounds.

`isAtLength logical`. Whether `MeanAtLength`/`MeanAtWeight` reflect a genuinely length- (or weight-) retention schedule, as opposed to a one derived from an age-only schedule (e.g. via the age-length key). Controls whether fleet weight-at-age calculations (`WeightFleetRetained`) weight by this object's at-length schedule or fall back to the stock's plain weight-at-age. Default `TRUE`.

`Misc list`. Miscellaneous additional inputs. For internal use.

### See Also

- `Retention()` for the constructor and full parameter documentation.
- `RetentionModels()` for available model functions.
- fleet for the enclosing fleet object.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `selectivity-class`

---

## `Retention-Models` *Retention Models*

---

### Description

Logistic, knife-edge, and double-normal retention-at-age, retention-at-length, and retention-at-weight models for use in a `Retention()` object.

### Usage

```
RetentionAtAge(Ages, RA50, RA50_95, MaxRet = 1)

RetentionKnifeEdgeAge(Ages, RA)

RetentionAtLength(Length, RL50, RL50_95, MaxRet = 1)

RetentionKnifeEdgeLength(Length, RL)

RetentionAtWeight(Weight, RW50, RW50_95, MaxRet = 1)

DoubleNormalRetention(Length, LR5, LFR, Rmaxlen)

DoubleNormalRetentionWeight(Weight, WR5, WFR, Rmaxweight)

RetentionModels(full = TRUE, print = TRUE)
```

```
RetentionModelsLength(full = TRUE, print = TRUE)

RetentionModelsAge(full = TRUE, print = TRUE)

RetentionModelsWeight(full = TRUE, print = TRUE)
```

**Arguments**

| | |
|---|---|
| `Ages` | Numeric vector of age classes. |
| `RA50` | Numeric. Age at 50% retention (logistic-at-age). |
| `RA50_95` | Numeric. Interval between `RA50` and age at 95% retention (logistic-at-age). |
| `MaxRet` | Numeric. Maximum (asymptotic) retention. Default `1` (full retention). Values less than 1 produce a dome-shaped or partially selective retention curve. |
| `RA` | Numeric. Knife-edge age threshold; fish at or above this age are fully retained. |
| `Length` | Numeric vector of length classes. |
| `RL50` | Numeric. Length at 50% retention (logistic-at-length). |
| `RL50_95` | Numeric. Interval between `RL50` and length at 95% retention (logistic-at-length). |
| `RL` | Numeric. Knife-edge length threshold; fish at or above this length are fully retained. |
| `Weight` | Numeric vector of weight classes. |
| `RW50` | Numeric. Weight at 50% retention (logistic-at-weight). |
| `RW50_95` | Numeric. Interval between `RW50` and weight at 95% retention (logistic-at-weight). |
| `LR5` | Numeric. Length at 5% retention (double-normal). |
| `LFR` | Numeric. Length at full retention (double-normal). |
| `Rmaxlen` | Numeric. Retention at `max(Length)` (double-normal). Values less than 1 produce a dome-shaped retention curve. |
| `WR5` | Numeric. Weight at 5% retention (double-normal). |
| `WFR` | Numeric. Weight at full retention (double-normal). |
| `Rmaxweight` | Numeric. Retention at `max(Weight)` (double-normal). Values less than 1 produce a dome-shaped retention curve. |
| `full` | Logical. If `TRUE` (default), returns a complete table of available models. If `FALSE`, returns model names only. |
| `print` | Logical. If `TRUE` (default), prints results to the console. If `FALSE`, returns the data frame invisibly without printing. |

### Details

Three families of retention model are available:

- **Logistic**: a standard increasing retention curve parameterised by the 50% and 95% retention points, with asymptote `MaxRet`:

$$R(x) = \frac{\texttt{MaxRet}}{1 + \exp\left(-\ln(19) \cdot \frac{x - x_{50}}{x_{95} - x_{50}}\right)}$$

- **Knife-edge**: full retention at or above a threshold, zero below:

$$R(x) = \begin{cases} 0 & x < x_t \\ 1 & x \geq x_t \end{cases}$$

- **Double-normal**: a combination of ascending and descending half-normal curves, producing either asymptotic (`Rmaxlen = 1`) or dome-shaped (`Rmaxlen < 1`) retention.

At-length and at-weight schedules are converted internally to at-age using the age-length key and age-weight key respectively.

The available model functions are:

- `RetentionAtAge()`: logistic retention-at-age.
- `RetentionKnifeEdgeAge()`: knife-edge retention-at-age.
- `RetentionAtLength()`: logistic retention-at-length.
- `RetentionKnifeEdgeLength()`: knife-edge retention-at-length.
- `RetentionAtWeight()`: logistic retention-at-weight.
- `DoubleNormalRetention()`: double-normal retention-at-length.
- `DoubleNormalRetentionWeight()`: double-normal retention-at-weight.

The `RetentionModels*` functions list available models:

- `RetentionModels()`: all retention models.
- `RetentionModelsAge()`: retention-at-age models only.
- `RetentionModelsLength()`: retention-at-length models only.
- `RetentionModelsWeight()`: retention-at-weight models only.

### Value

- `RetentionAtAge()`, `RetentionKnifeEdgeAge()`, `RetentionAtLength()`, `RetentionKnifeEdgeLength()`, `RetentionAtWeight()`, `DoubleNormalRetention()`, `DoubleNormalRetentionWeight()`: a numeric vector of retention values (0–1) at each age, length, or weight class respectively.
- `RetentionModels()`, `RetentionModelsAge()`, `RetentionModelsLength()`, `RetentionModelsWeight()`: invisibly returns a data frame (if `full = TRUE`) or character vector (if `full = FALSE`) of available models. Prints to console if `print = TRUE`.

### See Also

Retention(), Selectivity(), Ages(), Length(), Weight(), Fleet()

### Examples

```
ages   <- 0:10
lengths <- seq(0, 100, by = 1)
weights <- seq(0.1, 10, by = 0.5)

# ---- Logistic Retention ----
ret_age_log   <- RetentionAtAge(ages, RA50 = 3, RA50_95 = 2)
ret_length_log <- RetentionAtLength(lengths, RL50 = 40, RL50_95 = 10)
ret_weight_log <- RetentionAtWeight(weights, RW50 = 5, RW50_95 = 2)

# ---- Knife-Edge Retention ----
ret_age_ke    <- RetentionKnifeEdgeAge(ages, RA = 4)
ret_length_ke <- RetentionKnifeEdgeLength(lengths, RL = 50)

# ---- Double-Normal Retention ----
ret_dn_length <- DoubleNormalRetention(lengths, LR5 = 20, LFR = 50, Rmaxlen = 0.8)
ret_dn_weight <- DoubleNormalRetentionWeight(weights, WR5 = 1, WFR = 5, Rmaxweight = 0.8)


# ---- Plots ----
# Age-based
plot(ages, ret_age_log, type='l', lwd=2, col='blue', ylim=c(0,1),
     main='Retention-at-Age', xlab='Age', ylab='Retention')
lines(ages, ret_age_ke, type='s', lwd=2, col='red')
legend("bottomright", legend=c("Logistic","Knife-Edge"), col=c("blue","red"), lwd=2)

# Length-based
plot(lengths, ret_length_log, type='l', lwd=2, col='blue', ylim=c(0,1),
     main='Retention-at-Length', xlab='Length', ylab='Retention')
lines(lengths, ret_length_ke, type='s', lwd=2, col='red')
lines(lengths, ret_dn_length, lwd=2, col='green')
legend("bottomright", legend=c("Logistic","Knife-Edge","Double-Normal"),
       col=c("blue","red","green"), lwd=2)

# Weight-based
plot(weights, ret_weight_log, type='l', lwd=2, col='blue', ylim=c(0,1),
     main='Retention-at-Weight', xlab='Weight', ylab='Retention')
lines(weights, ret_dn_weight, lwd=2, col='green')
legend("bottomright", legend=c("Logistic","Double-Normal"),
       col=c("blue","green"), lwd=2)
```

---

## runCOSEWIC *COSEWIC MSE run using the correct MPs and projected time horizon*

---

## Description

Dedicated functions for MSE run and reporting for COSEWIC (Committee on the Status of Endangered Wildlife in Canada). MSE projects for 6x maximum age using NFref, FMSYref and curE management procedures.

## Usage

```
runCOSEWIC(OM, ...)

COSEWIC_Pplot(
  MSEobj,
  syear = 2017,
  qcol = "#FFCB62",
  quants = c(0.05, 0.25, 0.5, 0.75, 0.95)
)

COSEWIC_Dplot(
  MSEobj,
  syear = 2017,
  qcol = "#79F48D",
  quants = c(0.05, 0.25, 0.5, 0.75, 0.95),
  nGT = 3
)

COSEWIC_Blow(
  MSEobj,
  syear = 2017,
  qcol = rgb(0.4, 0.8, 0.95),
  quants = c(0.05, 0.25, 0.5, 0.75, 0.95),
  nGT = 3
)

COSEWIC_Hplot(
  MSEobj,
  syear = 2017,
  qcol = rgb(0.4, 0.8, 0.95),
  quants = c(0.05, 0.25, 0.5, 0.75, 0.95)
)

COSEWIC_report(
  MSEobj,
  output_file = NA,
  author = "Author not specified",
  title = NA
)

COSEWIC_tab(MSEobj, rnd = 0, GTs = c(3, 6), syear = 2017, nGT = 3)
```

```
COSEWIC_tab_formatted(
  Ptab1,
  thresh = c(20, 40, 40, 20, 40, 40, 40, 30, 5),
  ret_thresh = F
)
```

## Arguments

| | |
|---|---|
| OM | An operating model object of class OM |
| ... | Other named arguments to pass to runMSE |
| MSEobj | An object of class MSE with MPs = c("NFref", "FMSYref", "curE") |
| syear | Current year, starting year for projections (e.g. 2017) |
| qcol | Color of shaded regions (bars, quantiles) |
| quants | Quantiles of the shaded regions (vector 5 long e.g. 0.1, 0.2, 0.5, 0.8, 0.9) |
| nGT | Number of generation times. For COSEWIC_tab, for moving window of SSB chance (metrics A1 and A2). For COSEWIC_Blow and COSEWIC_Dplot, used for projections (the number of projection years should be greater than MaxAge * nGT). |
| output_file | The directory and filename you wish to use for the report e.g. "C:/temp/myMSEreport.html" |
| author | The person who made this report |
| title | The title of the report |
| rnd | The number of significant figures for rounding. |
| GTs | A vector of mean generation times to evaluate performance metrics over |
| Ptab1 | A COSEWIC performance table made by COSEWIC_tab |
| thresh | A vector of thresholds for each column Health, Yield and Reb are 'greater than threshold' conditions |
| ret_thresh | Logical: if true just the threshold levels are returned |

## Functions

- runCOSEWIC(): Calls runMSE with number of projection years for 6x maximum age and uses NFref, FMSYref, and curE MPs.
- COSEWIC_Pplot(): Projection plots of spawning stock biomass under three scenarios: no catch, FMSY fishing and status quo fishing effort.
- COSEWIC_Dplot(): Depletion plots evaluate whether significant declines have occurred over three generation times in both historical and projection years.
- COSEWIC_Blow(): Plots that evaluate the likelihood of declining below Blow, by default, biomass that takes 3 generation times to reach half BMSY with zero fishing
- COSEWIC_Hplot(): Plots of historical spawning stock relative to unfished and MSY levels.
- COSEWIC_report(): Create a standard DFO COSEWIC report (provides performance plots to inform COSEWIC processes in Canadian fish stocks).
- COSEWIC_tab(): Creates a standard COSEWIC performance table:

  - P_Cr is the probability of being in the critical zone (less than 20% depletion)
  - P_Ct is the probability of being in the cautious zone (between 20% and 40% depletion)
  - P_H is the probability of being in the healthy zone (above 40% depletion)
  - P_Cr_MSY is the probability of being in the critical zone (less than 40% BMSY)
  - P_Ct_MSY is the probability of being in the cautious zone (between 40% and 80% BMSY)
  - P_H_MSY is the probability of being in the healthy zone (above 80% BMSY)
  - Caut is the probability of being in the cautious zone in the last 10 projected years
  - P_A1 is the probability of being designated threatened according to COSEWIC Indicator A1 (Spawning biomass less than 70% that three generation times previously)
  - P_A2 is the probability of being designated threatened according to COSEWIC Indicator A2 (Spawning biomass less than 50% that three generation times previously)
  - Blow is the probability that the stock is below the biomass for which it takes 3 generation times to reach 50% BMSY with zero fishing
- COSEWIC_tab_formatted(): A formatted version of the standard COSEWIC performance plot, color coded by thresholds.

## Author(s)

T. Carruthers

## References

https://cosewic.ca/index.php/en/

---

`runInMP` *Runs input control MPs on a Data object.*

---

## Description

Function runs a MP (or MPs) of class 'Input' and returns a list: input control recommendation(s) in element 1 and Data object in element 2.

## Usage

```
runInMP(Data, MPs = NA, reps = 100)
```

## Arguments

| | |
|---|---|
| Data | A object of class Data |
| MPs | A vector of MPs of class 'Input' |
| reps | Number of stochastic repetitions - often not used in input control MPs. |

### Author(s)

A. Hordyk

---

`runMP` *Run a Management Procedure*

---

### Description

Run a Management Procedure

### Usage

```
runMP(Data, MPs = NA, reps = 100, perc = 0.5, chkMPs = FALSE, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `Data` | A MSEtool Data object |
| `MPs` | The name of the MP to run (or a vector or names) |
| `reps` | Number of repetitions |
| `perc` | Percentile to summarize reps (default is median) |
| `chkMPs` | Logical. Should the MPs be checked before attempting to run them? |
| `silent` | Logical. Should messages by suppressed? |

### Value

invisibly returns the Data object

---

`runMSE` *Run a Management Strategy Evaluation*

---

### Description

The primary user-facing function for running a complete MSE. Simulates historical dynamics with `Simulate()`, then projects forward under one or more management procedures (MPs) with `Project()`, returning a completed `MSE` object.

### Usage

```
runMSE(
  OM = NULL,
  MPs = NULL,
  Hist = FALSE,
  silent = FALSE,
  parallel = FALSE,
  extended = FALSE,
  checkMPs = FALSE,
  nSim = NULL,
  nsim = NULL,
  DoDynamicUnfished = TRUE,
  DoRefLandings = TRUE,
  DoRefRemovals = FALSE,
  DoConditionObs = TRUE,
  DoGenerateData = TRUE,
  DoMSYRefs = TRUE,
  Reduce = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| `OM` | An om, OM-legacy, hist, or Hist-legacy object. If `NULL` (default), uses SingleStockOM. |
| `MPs` | MPs to project: a character vector of names of MP functions available in the current environment, or (for hist objects) a named list of MP functions, optionally mixed with MP names, e.g. `list(IR5 = SetMPArgs(IndexRate, CalibYears = 5), 'IndexTarget')`. List names are used as the MP names. If `NULL` (default), projects `c("CurrentEffort", "CurrentCatch")`. |
| `Hist` | Logical. If `TRUE`, returns the hist or Hist-legacy object from `Simulate()` without running forward projections. Useful for inspecting historical dynamics or saving the conditioned OM for later use with `Project()`. Default `FALSE`. |
| `silent` | Logical. Suppress progress messages if `TRUE`. Default `FALSE`. |
| `parallel` | Logical or named list controlling parallel execution of MPs. For legacy `OM` and `Hist` objects only:<br>• if `TRUE`, all MPs are run in parallel.<br>• if a named list, names correspond to individual MPs to run in parallel.<br>• pass `"sac"` to use the SAC parallel backend (OM-legacy only). Default `FALSE`. |
| `extended` | Logical. If `TRUE`, stores full age- and area-structured arrays for all years in `MSE@Misc$extended` (legacy only). Substantially increases object size. Default `FALSE`. |
| `checkMPs` | Logical. Validate MP names and availability before projecting. Default `FALSE`. Legacy only. |

`nSim` Integer. Reduce the number of simulations to `nSim`. Passed to both `Simulate()` and `Project()`. If `NULL` (default), all simulations are used.

`nsim` Integer. Alternative to `nSim` for OM-legacy objects. Sets `OM@nsim` before simulating. Default `NULL`.

`DoDynamicUnfished` Logical. Calculate dynamic unfished reference points? Passed to `Simulate()`. Default `TRUE`. `om` and `hist` class only.

`DoRefLandings` Logical. Calculate landings-based reference points? Passed to `Simulate()`. Default `TRUE`. `om` and `hist` class only.

`DoRefRemovals` Logical. Calculate removals-based reference points? Passed to `Simulate()`. Default `FALSE`. `om` and `hist` class only.

`DoConditionObs` Logical. Condition observation model on historical data? Passed to `Simulate()`. Default `TRUE`. `om` and `hist` class only.

`DoGenerateData` Logical. Generate observed data for MPs? Passed to `Simulate()`. Default `TRUE`. `om` and `hist` class only.

`DoMSYRefs` Logical. Calculate MSY-based reference points? Passed to `Simulate()`. Default `TRUE`. `om` and `hist` class only.

`Reduce` Logical. Reserved for future use. Default `TRUE`. `om` and `hist` class only.

`...` Additional arguments. Reserved for future use.

## Details

Accepts either an operating model (om or OM-legacy) or a pre-conditioned historical object (hist or Hist-legacy).

When a historical object is provided, `Simulate()` is skipped and the existing historical dynamics are used directly, saving computation time when projecting new MPs.

If `OM = NULL`, uses SingleStockOM and projects example MPs CurrentEffort and CurrentCatch as a quick demonstration.

## Value

If `Hist = TRUE`, a hist or Hist-legacy object containing the conditioned historical dynamics. Otherwise, a MSE-legacy object containing projection results for all MPs. If an error occurs during `Project()`, the historical simulations are returned with a warning.

## See Also

`Simulate()`, `Project()`, OMControl

`S4SlotSizes` *Summarise S4 Object Memory Usage by Slot*

## Description

Recursively inspects all slots of an S4 object, including nested S4 objects and lists containing S4 objects, and reports their memory footprint. Useful for identifying which slots dominate memory usage in complex S4 objects.

## Usage

```
S4SlotSizes(obj, n = 5, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `obj` | An S4 object. |
| `n` | Integer. Number of largest slots to display in the printed summary. Default is `5`. |
| `verbose` | Logical. If `TRUE` (default), prints a summary table of the top `n` slots. If `FALSE`, returns the full results silently. |

## Details

Slot paths reflect the access pattern needed to reach each element. For example, a nested S4 slot would appear as `obj@SlotA@SlotB`, and a named list element inside a slot as `obj@SlotA[["item"]]`.

Note that sizes are **not additive** — parent slots include the memory of their children, so a slot containing a large nested S4 object will itself appear large.

## Value

A `data.frame` with columns:

- `path`: Full slot path using `@` notation for S4 slots and `[["name"]]` or `[[i]]` notation for list elements.
- `mb`: Size of the slot or element in megabytes.

Returned invisibly, sorted descending by size.

## Examples

```
## Not run:
# Inspect top 10 slots of a complex S4 model object
S4SlotSizes(my_model, n = 10)

# Retrieve full results without printing
sizes <- S4SlotSizes(my_model, verbose = FALSE)
sizes[sizes$mb > 1, ]  # slots larger than 1 MB
```

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

---

`Save` *Save an Object to Disk*

---

## Description

A wrapper for `saveRDS()` that automatically creates the required directory structure and prints a informational message to the console.

## Usage

```
Save(object, path = NULL, overwrite = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `object` | Any R object to save. |
| `path` | Character. File path including file name and extension (e.g., `"results/my_om.rds"`). If `NULL` (default), a temporary file path is generated via `tempfile()`. |
| `overwrite` | Logical. If `FALSE` (default), an error is thrown if `path` already exists. Set to `TRUE` to overwrite an existing file. |
| `...` | Additional arguments passed to `saveRDS()`. |

## Value

Invisibly returns the full file path of the saved object.

## See Also

`saveRDS()`

## Examples

```
x <- list(a = 1, b = 2)
path <- Save(x, path = tempfile(fileext = ".rds"))
path
```

`SeasonalSpatialExStock`

*Seasonal Spatial Example Stock*

## Description

An example stock object representing a short-lived tropical reef fish with monthly time steps, seasonal recruitment, and an ontogenetic habitat shift across three spatial areas. All biological parameters are fixed (no stochastic bounds). This object is intended to demonstrate seasonal and spatial model features rather than reflect a real species.

## Usage

```
SeasonalSpatialExStock
```

## Format

A stock object with the following slots populated:

- `Name`: `"Example Seasonal Spatial Stock"`.
- `CommonName`: `"Reef Fish"`.
- `Species`: `"Example species"`.
- `Ages`: An `Ages()` object with `MaxAge = 36` months (`Units = "month"`), giving 37 monthly age classes expressed in years (0, 1/12, 2/12, ..., 3).
- `Length`: A `Length()` object with fixed von Bertalanffy parameters — `Linf = 30` cm, `K = 0.8` yr^-1^, `t0 = -0.1` yr — and `CVatAge = 0.1`.
- `Weight`: A `Weight()` object with allometric parameters `alpha = 1e-05` and `beta = 3`.
- `NaturalMortality`: A `NaturalMortality()` object with fixed `M = 0.09` per month (~1.1 yr^-1^). This is a per-time-step rate and must match `Ages@Units = "month"` (see Details).
- `Maturity`: A `Maturity()` object with fixed logistic maturity-at-length parameters `L50 = 15` cm and `L50_95 = 3` cm.
- `Fecundity`: An empty `Fecundity()` object; fecundity-at-age is computed internally during population (see Details).
- `SRR`: A `SRR()` object with Beverton-Holt steepness `h = 0.7`, seasonal unfished recruitment from `SetSeasonalR0()` with an annual pulse peaking in month 6 (`Sigma = 1.5` months), recruitment variability `SD = 0.4`, and autocorrelation `AC = 0.3`.
- `Spatial`: A `Spatial()` object specifying a three-area ontogenetic habitat shift (see Details).
- `Depletion`: A `Depletion()` object with terminal depletion of 35% of `B0`.
- `nYear`, `pYear`, `nSim`, `CurrentYear`, `Years`, `Seasons`: Set by `PopulateStock()`.
- `Misc`: Empty list for user-defined information.
- `Log`: Internal list for tracking object state.

## Details

### Monthly Time Steps:

`Ages@Units = "month"` sets the within-year resolution to monthly time steps. This stock should be paired with an `OM()` with `Seasons = 12`. `CalcAgeClasses()` produces 37 age classes from 0 to 3 years in steps of 1/12 year.

### Seasonal Recruitment:

`SetSeasonalR0()` distributes annual unfished recruitment (`AnnualR0 = 1000`) across months using a Gaussian curve centred on month 6 with standard deviation 1.5 months, creating a sharply seasonal pulse.

### Spatial Structure:

The three areas represent a linear inshore-to-offshore gradient:

| Area | Habitat | Relative size |
|---|---|---|
| 1 | Inshore lagoon / seagrass | 15% |
| 2 | Mid-shelf reef flat | 35% |
| 3 | Outer reef slope | 50% |

The unfished biomass distribution shifts progressively from Area 1 (juveniles) to Area 3 (adults) with age, specified via a `Sim × Area × Age` array of Gaussian curves centred at months 1, 18, and 36 respectively. Movement is constrained by adjacency: `FracOther` connects Areas 1 2 and 2 3 freely, while Areas 1 3 are weakly connected (`FracOther = 0.02`). `PopulateSpatial()` fits the movement matrix to reproduce the target distribution at each age.

### Length:

`PopulateLength()` computes mean length-at-age from the fixed von Bertalanffy parameters using the growth equation, then applies log-normal variation scaled by `CVatAge = 0.1` to construct an age-length key.

See also `LengthModels()`.

### Weight:

`PopulateWeight()` applies the allometric relationship $W =$ `alpha` $\times L\hat{}$`beta`$\hat{}$ to mean length-at-age to produce mean weight-at-age.

See also `WeightModels()`.

### Natural Mortality:

`PopulateNaturalMortality()` expands the fixed `M` into a constant natural mortality-at-age array. Stochastic annual deviates are added using `SD` and `AC` from the `SRR` slot.

See also `NaturalMortalityModels()`.

### Maturity:

`PopulateMaturity()` applies the logistic maturity-at-length curve, parameterised by `L50` and `L50_95`, to the populated length-at-age array to produce a maturity-at-age array.

See also `MaturityModels()`.

**Fecundity:**
Because the `Fecundity` slot is empty, `PopulateFecundity()` computes fecundity-at-age as the element-wise product of weight-at-age and maturity-at-age.
See also `FecundityModels()`.

**Stock-Recruitment:**
`PopulateSRR()` uses the fixed steepness `h = 0.7` and generates a time series of log-normal recruitment deviates with `SD = 0.4` and autocorrelation `AC = 0.3`.
See also `SRRModels()`.

**Depletion:**
`PopulateDepletion()` sets the terminal historical biomass to a fixed 35% of `B0` across all simulations.

## See Also

AlbacoreExStock, ButterfishExStock, `Stock()`, `PopulateStock()`, `Populate()`, stock, `SetSeasonalR0()`, `Spatial()`

## Examples

```
SeasonalSpatialExStock

## Not run:
PopulatedStock <- PopulateStock(
  SeasonalSpatialExStock,
  nYear = 20,
  pYear = 10,
  nSim  = 48,
  seed  = 42
)

## End(Not run)
```

---

# SeasonalSpatialOM *Seasonal Spatial Example Operating Model*

---

## Description

An example om object pairing SeasonalSpatialExStock with DomeExFleet and CommercialFleetObs. Intended to demonstrate operating models with monthly time steps, seasonal recruitment, and age-based movement across three spatial areas. Parameter values are illustrative only and do not represent a real stock or fishery. See also SingleStockOM, TwoFleetOM, MultiStockOM, and ComplexOM for other structural configurations.

## Usage

```
SeasonalSpatialOM
```

### Format

An om object with the following slots populated:

- `Name`: `"Seasonal Spatial - Single Fleet"`.
- `nSim`: `8`. Number of stochastic simulations.
- `nYear`: `20`. Number of historical years.
- `pYear`: `10`. Number of projection years.
- `Seasons`: `12`. Monthly time steps within each calendar year. Must match `SeasonalSpatialExStock@Ages@Un` `= "month"`.
- `Stock`: SeasonalSpatialExStock. A single stock object.
- `Fleet`: DomeExFleet. A single fleet object.
- `Obs`: CommercialFleetObs. A single obs object.
- `Imp`: FullComplianceImp. Perfect TAC/effort implementation and full immediate size-regulation compliance.

### Details

**Stock:**

SeasonalSpatialExStock represents a short-lived reef fish (maximum age 36 months) with fixed biological parameters, a seasonal recruitment pulse peaking in month 6, and an ontogenetic habitat shift across three spatial areas (inshore lagoon → mid-shelf reef → outer slope). See SeasonalSpatialExStock for full parameter details.

**Seasons:**

`Seasons = 12` sets the OM to monthly time steps, matching `Ages@Units = "month"` in SeasonalSpatialExStock. All within-year dynamics — recruitment pulses, seasonal movement, and seasonal mortality — operate at the monthly scale during simulation.

**Fleet:**

DomeExFleet uses dome-shaped length-based selectivity, with reduced retention of the largest fish. This reflects a gear type (e.g. traps or gillnets) that is size-selective rather than retaining everything above a knife-edge length. The effort trajectory rises gradually across the historical period to full effort at the terminal year.

See DomeExFleet for full parameter details.

**Observation Model:**

CommercialFleetObs provides landings, discards, fishing effort, and a commercial CPUE index. There are no fishery-independent survey or composition data, representing a data-limited monitoring scenario typical of small-scale reef fisheries.

See CommercialFleetObs for full parameter details.

**Implementation Error:**

FullComplianceImp models perfect implementation: `TAC@Mean = 1`/ `SD = 0` and `Effort@Mean = 1`/`SD = 0` mean realised TAC/effort always equal advice exactly, and `Size@Compliance = 1` means any size-regulation advice (a change to `Advice@Retention`/`Advice@Selectivity`) is adopted immediately by the whole fleet. See `Imp()` for details, and OverageImp, UnderageImp, PartialSizeComplianceImp for contrasting examples.

### See Also

SingleStockOM, TwoFleetOM, MultiStockOM, ComplexOM for structural variants. SeasonalSpatialExStock, DomeExFleet, CommercialFleetObs, FullComplianceImp for the component objects. `OM()`, om, `runMSE()`, `PopulateOM()`, `ExampleMPs()`

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SingleStockOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

### Examples

```
SeasonalSpatialOM

## Not run:
Hist <- runMSE(SeasonalSpatialOM, MPs = ExampleMPs())

## End(Not run)
```

---

## `select_MP` *Select DataList for an MP from* `MMSE@PPD`

---

### Description

Select DataList for an MP from `MMSE@PPD`

### Usage

```
select_MP(PPD, MP = 1)
```

### Arguments

| | |
|---|---|
| `PPD` | `PPD` slot from an MMSE object |
| `MP` | Numeric value indicating the MP to return DataList |

### Value

A nested list `Data` objects (`nstock` by `nfleet`)

`Selectivity` *Selectivity*

## Description

Construct and manipulate a selectivity object defining selectivity-at-age, selectivity-at-length, or selectivity-at-weight for a `Fleet()` object. Selectivity is required for all fleets.

## Usage

```
Selectivity(
  Pars = list(),
  Model = NULL,
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  MeanAtWeight = NULL,
  Classes = NULL,
  isRel = FALSE,
  isAtLength = TRUE,
  Misc = list()
)

isRel(x)

isRel(x) <- value

Selectivity(x) <- value
```

## Arguments

| | |
|---|---|
| `Pars` | Named list of selectivity parameters passed to the selectivity model function. Parameter names must match the arguments of the chosen model (see `SelectivityModels()`). Each element may be a scalar, vector, or array; see Specifying Biological and Fleet Schedules for accepted formats. Default `list()` (empty).<br>If `Pars` and `Model` are both provided, `MeanAtAge` is computed from the model and any values already in `MeanAtAge` are overwritten. To preserve a user-specified `MeanAtAge`, leave `Pars = list()` (the default).<br>If `Pars` is a fleet object, the `Selectivity` slot of that fleet is returned. |
| `Model` | Character or function or `NULL`. Selectivity model identifier. If `NULL` (default), the model is inferred automatically from the names in `Pars`. May also be supplied as a custom R function with arguments matching those in `Pars`. See `SelectivityModels()` for built-in options. |
| `MeanAtAge` | Numeric array or `NULL`. Mean selectivity-at-age with dimensions `Sim x Age x Year` (area dimension added during population). Used directly when `Pars` is empty and `MeanAtLength` is not supplied. See Specifying Biological and |

Fleet Schedules for accepted array formats. **Note:** if `Pars` and `Model` are both provided, any values supplied here will be overwritten during population.

`MeanAtLength` Numeric array or `NULL`. Mean selectivity-at-length with dimensions `Sim x Class x Year`. If provided and `Pars` is empty, takes precedence over `MeanAtAge`; `MeanAtAge` is derived from it via the age-length key. A numeric vector is also accepted and converted to a `1 x nClass x 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`.

`MeanAtWeight` Numeric array or `NULL`. Mean selectivity-at-weight with dimensions `Sim x Class x Year`. A numeric vector is also accepted and converted the same way as `MeanAtLength`, using the weight object's `AWK`. Default `NULL`.

`Classes` Numeric vector or `NULL`. Length or weight class midpoints corresponding to the second dimension of `MeanAtLength` or `MeanAtWeight`. Default `NULL`.

`isRel` Logical. If `TRUE`, length-based parameters (e.g., `L5`, `LFS`) are interpreted as multiples of the length-at-50%-maturity (`L50`) of the paired stock rather than absolute length values. A `Maturity()` object must be available to `PopulateSelectivity()` for scaling to occur. Default `FALSE`.

`isAtLength` Logical. Whether `MeanAtLength`/`MeanAtWeight` (or an at-length/at-weight `Model`) reflect a genuinely length- or weight- selectivity schedule. Set `FALSE` when selectivity is only meaningfully defined at age (e.g. imported from an age-structured assessment) so that fleet weight-at-age calculations skip weighting by this object's at-length schedule and use the stock's plain weight-at-age instead. Default `TRUE`.

`Misc` List. Miscellaneous additional inputs. Default `list()`.

`x` A selectivity object, or a fleet object for `Selectivity<-`.

`value` For `Selectivity<-`: a selectivity object. For `isRel<-`: a logical value.

## Details

Selectivity is required for all fleet objects. It defines the probability that a fish of a given age, length, or weight is caught by the gear, on a scale from 0 to 1.

**Specifying Selectivity:**

There are two ways to specify selectivity:

**Model-based** (recommended): supply `Pars` as a named list whose element names match the arguments of a built-in or custom model function. The model is resolved automatically unless `Model` is specified explicitly. See `SelectivityModels()` for available models and their required parameters, and Specifying Biological and Fleet Schedules for how parameter values are structured across simulations and years.

Available model families are: logistic (at-age, at-length, at-weight), knife-edge (at-age, at-length), and double-normal (at-length, at-weight). At-length and at-weight schedules are converted to at-age internally using the age-length or age-weight key.

**Direct array**: leave `Pars = list()` and supply `MeanAtAge`, `MeanAtLength`, or `MeanAtWeight` directly. If `MeanAtLength` is provided, it takes precedence and `MeanAtAge` is derived from

it. If only `MeanAtAge` is provided, `MeanAtLength` is calculated from it unless already populated.

These two approaches are mutually exclusive: if `Pars` is non-empty and `Model` can be resolved, any values in `MeanAtAge` will be overwritten.

**Relative Parameters:**

When `isRel = TRUE`, length-based parameters (e.g., `L5`, `LFS`) are scaled by the maturity `L50` of the paired stock before the selectivity curve is computed. This allows selectivity to be expressed as a fraction of the length at 50% maturity rather than an absolute length, which is useful when the same fleet configuration is applied across stocks with different growth characteristics.

**Attaching to a Fleet:**

A selectivity object can be attached to a `Fleet()` with `Selectivity(Fleet) <- MySelectivity` and retrieved with `Selectivity(Fleet)`.

Individual slots may be accessed or modified using `Pars()`, `Model()`, `MeanAtAge()`, `MeanAtLength()`, `MeanAtWeight()`, `Classes()`, and `isRel()`.

## Value

- `Selectivity()` returns a selectivity object. If `Pars` is a fleet object, the `Selectivity` slot of that fleet is returned.
- `Selectivity<-` returns `x` with the `Selectivity` slot replaced by `value`.
- `isRel()` returns the `isRel` slot from `x`.
- `isRel<-` returns `x` with the `isRel` slot updated.

## See Also

- selectivity for the class definition and slot-level documentation.
- `SelectivityModels()` for available model functions and their parameters.
- `Fleet()` for the enclosing fleet constructor.
- `Retention()`, `DiscardMortality()` for related fleet components.
- Specifying Biological and Fleet Schedules for how `Pars`, `Model`, and `MeanAt*` arrays are structured.
- `PopulateSelectivity()` for population details.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`, `selectivity-class`

## Examples

```
## ---- List available selectivity models ----
SelectivityModels()

## ---- Double-normal selectivity-at-length (asymptotic) ----
# The double-normal model is parameterised by L5 (length at 5% selectivity),
# LFS (length at full selectivity), and Vmaxlen (selectivity at max length).
```

```
# Vmaxlen = 1 produces an asymptotic curve; values < 1 produce dome-shaped.
s_asymptotic <- Selectivity(
  Pars = list(L5 = 20, LFS = 35, Vmaxlen = 1)
)
isRel(s_asymptotic)

## ---- Double-normal selectivity-at-length (dome-shaped) ----
s_dome <- Selectivity(
  Pars = list(L5 = 20, LFS = 35, Vmaxlen = 0.5)
)

## ---- Logistic selectivity-at-length ----
# Parameterised by SL50 (length at 50% selectivity) and SL50_95 (the
# length increment from 50% to 95% selectivity).
s_logistic <- Selectivity(
  Pars = list(SL50 = 30, SL50_95 = 10)
)

## ---- Logistic selectivity-at-age ----
s_age <- Selectivity(
  Pars = list(SA50 = 3, SA50_95 = 2)
)

## ---- Knife-edge selectivity-at-length ----
# Full selectivity at or above SL; zero below.
s_knife <- Selectivity(
  Pars = list(SL = 25)
)

## ---- Relative parameters (isRel = TRUE) ----
# When isRel = TRUE, L5 and LFS are interpreted as multiples of the
# maturity L50 of the paired stock. This is useful when applying the same
# fleet configuration across stocks with different growth characteristics.
# PopulateSelectivity() scales L5 and LFS by L50 before computing the curve;
# a Maturity object must be available at population time.
s_rel <- Selectivity(
  Pars  = list(L5 = 0.5, LFS = 0.9, Vmaxlen = 1),
  isRel = TRUE
)
isRel(s_rel)

## ---- Direct MeanAtAge array ----
# When Pars is empty (default), MeanAtAge can be supplied directly.
# The array must have dimensions Sim x Age x Year with named dimnames.
# Any values in MeanAtAge are preserved because Pars is empty.
ages     <- 0:20
sel_vals <- pmin(ages / 5, 1)   # ramp from 0 to 1, plateau at age 5

sel_array <- array(
  sel_vals,
  dim      = c(1, length(ages), 1),
  dimnames = list(Sim = 1, Age = ages, Year = 2000)
)
```

```
s_direct <- Selectivity(MeanAtAge = sel_array)
MeanAtAge(s_direct)

## ---- isRel accessor and replacement ----
s <- Selectivity(Pars = list(L5 = 20, LFS = 35, Vmaxlen = 1))
isRel(s)
isRel(s) <- TRUE
isRel(s)

## ---- Attaching to / extracting from a Fleet object ----
# Passing a fleet object as the first argument returns its Selectivity slot.
f <- Fleet(Name = "Trawl")
Selectivity(f) <- Selectivity(Pars = list(L5 = 20, LFS = 35, Vmaxlen = 1))
Selectivity(f)
isRel(Selectivity(f))
```

---

`selectivity-class` *The* `selectivity` *S4 Class*

---

## Description

Defines selectivity-at-age, selectivity-at-length, or selectivity-at-weight for use in a fleet object. Selectivity is required for all fleets. Objects are typically created via the `Selectivity()` constructor, which documents all parameters in detail.

## Slots

`Pars list`. Named list of selectivity parameters passed to `Model`. See `Selectivity()` and `SelectivityModels()`.

`Model function` or `character` or `NULL`. Selectivity model identifier or function. See `SelectivityModels()`.

`isRel logical`, `numeric`, or `character`. Whether length-based parameters are expressed relative to maturity `L50`. See `Selectivity()`.

`MeanAtAge numeric` array or `NULL`. Mean selectivity-at-age (`Sim x Age x Year x Area`). See `Selectivity()`.

`MeanAtLength numeric` array or `NULL`. Mean selectivity-at-length (`Sim x Length x Year x Area`). Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). See `Selectivity()`.

`MeanAtWeight numeric` array or `NULL`. Mean selectivity-at-weight (`Sim x Weight x Year x Area`). Values are evaluated at the midpoint of each bin; the `Class` dimension is labelled by bin lower bounds (see `Classes`). See `Selectivity()`.

`Classes numeric` or `NULL`. Lower bounds of length or weight bins corresponding to the second dimension of `MeanAtLength` or `MeanAtWeight`. Bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength`/`MeanAtWeight` are evaluated at bin midpoints, not at these lower bounds.

`isAtLength` `logical`. Whether `MeanAtLength`/`MeanAtWeight` reflect a genuinely length- (or weight-) selectivity schedule, as opposed to one derived from an age-only schedule (e.g. via the age-length key). Controls whether fleet weight-at-age calculations (`WeightFleetRetained`/`WeightFleetSelected`) weight by this object's at-length schedule or fall back to the stock's plain weight-at-age. Default `TRUE`.

`Misc` `list`. Miscellaneous additional inputs. Used internally.

### See Also

- `Selectivity()` for the constructor and full parameter documentation.
- `SelectivityModels()` for available model functions.
- fleet for the enclosing fleet object.

Other fleet: `Bioeconomic()`, `Catchability()`, `DiscardMortality()`, `Effort()`, `Fleet()`, `Retention()`, `Selectivity()`, `bioeconomic-class`, `catchability-class`, `discardmortality-class`, `effort-class`, `fleet-class`, `retention-class`

---

## `Selectivity-Models` *Selectivity Models*

---

### Description

Logistic, knife-edge, and double-normal selectivity-at-age, selectivity-at-length, and selectivity-at-weight models for use in a `Selectivity()` object.

### Usage

```
SelectivityAtAge(Ages, SA50, SA50_95)

SelectivityKnifeEdgeAge(Ages, SA)

SelectivityAtLength(Length, SL50, SL50_95)

SelectivityKnifeEdgeLength(Length, SL)

SelectivityAtWeight(Weight, SW50, SW50_95)

DoubleNormal(Length, L5, LFS, Vmaxlen)

DoubleNormalWeight(Weight, W5, WFS, Vmaxweight)

SelectivityModels(full = TRUE, print = TRUE)

SelectivityModelsLength(full = TRUE, print = TRUE)

SelectivityModelsAge(full = TRUE, print = TRUE)

SelectivityModelsWeight(full = TRUE, print = TRUE)
```

### Arguments

| | |
|---|---|
| `Ages` | Numeric vector of age classes. |
| `SA50` | Numeric. Age at 50% selectivity (logistic-at-age). |
| `SA50_95` | Numeric. Interval between `SA50` and age at 95% selectivity (logistic-at-age). |
| `SA` | Numeric. Knife-edge age threshold; fish at or above this age are fully selected. |
| `Length` | Numeric vector of length classes. |
| `SL50` | Numeric. Length at 50% selectivity (logistic-at-length). |
| `SL50_95` | Numeric. Interval between `SL50` and length at 95% selectivity (logistic-at-length). |
| `SL` | Numeric. Knife-edge length threshold; fish at or above this length are fully selected. |
| `Weight` | Numeric vector of weight classes. |
| `SW50` | Numeric. Weight at 50% selectivity (logistic-at-weight). |
| `SW50_95` | Numeric. Interval between `SW50` and weight at 95% selectivity (logistic-at-weight). |
| `L5` | Numeric. Length at 5% selectivity (double-normal). |
| `LFS` | Numeric. Length at full selectivity (double-normal). |
| `Vmaxlen` | Numeric. Selectivity at `max(Length)` (double-normal). Values less than 1 produce a dome-shaped selectivity curve. |
| `W5` | Numeric. Weight at 5% selectivity (double-normal). |
| `WFS` | Numeric. Weight at full selectivity (double-normal). |
| `Vmaxweight` | Numeric. Selectivity at `max(Weight)` (double-normal). Values less than 1 produce a dome-shaped selectivity curve. |
| `full` | Logical. If `TRUE` (default), returns a complete table of available models. If `FALSE`, returns model names only. |
| `print` | Logical. If `TRUE` (default), prints results to the console. If `FALSE`, returns the data frame invisibly without printing. |

### Details

Three families of selectivity model are available:

- **Logistic**: a standard increasing selectivity curve parameterised by the 50% and 95% selectivity points:

$$S(x) = \frac{1}{1 + \exp\left(-\ln(19) \cdot \frac{x - x_{50}}{x_{95} - x_{50}}\right)}$$

- **Knife-edge**: full selectivity at or above a threshold, zero below:

$$S(x) = \begin{cases} 0 & x < x_t \\ 1 & x \geq x_t \end{cases}$$

- **Double-normal**: a combination of ascending and descending half-normal curves, producing either asymptotic (`Vmaxlen = 1`) or dome-shaped (`Vmaxlen < 1`) selectivity.

At-length and at-weight schedules are converted internally to at-age using the age-length key and age-weight key respectively.

The available model functions are:

- `SelectivityAtAge()`: logistic selectivity-at-age.
- `SelectivityKnifeEdgeAge()`: knife-edge selectivity-at-age.
- `SelectivityAtLength()`: logistic selectivity-at-length.
- `SelectivityKnifeEdgeLength()`: knife-edge selectivity-at-length.
- `SelectivityAtWeight()`: logistic selectivity-at-weight.
- `DoubleNormal()`: double-normal selectivity-at-length.
- `DoubleNormalWeight()`: double-normal selectivity-at-weight.

The `SelectivityModels*` functions list available models:

- `SelectivityModels()`: all selectivity models.
- `SelectivityModelsAge()`: selectivity-at-age models only.
- `SelectivityModelsLength()`: selectivity-at-length models only.
- `SelectivityModelsWeight()`: selectivity-at-weight models only.

## Value

- `SelectivityAtAge()`, `SelectivityKnifeEdgeAge()`, `SelectivityAtLength()`, `SelectivityKnifeEdgeLen`
  `SelectivityAtWeight()`, `DoubleNormal()`, `DoubleNormalWeight()`: a numeric vector of selectivity values (0–1) at each age, length, or weight class respectively.
- `SelectivityModels()`, `SelectivityModelsAge()`, `SelectivityModelsLength()`, `SelectivityModelsWeig`
  invisibly returns a data frame (if `full = TRUE`) or character vector (if `full = FALSE`) of available models. Prints to console if `print = TRUE`.

## See Also

`Selectivity()`, `Retention()`, `Ages()`, `Length()`, `Weight()`, `Fleet()`

## Examples

```
ages   <- 0:10
lengths <- seq(0, 100, by = 1)
weights <- seq(0.1, 10, by = 0.5)

# ---- Logistic Selectivity ----
sel_age_log   <- SelectivityAtAge(ages, SA50 = 3, SA50_95 = 2)
sel_length_log <- SelectivityAtLength(lengths, SL50 = 40, SL50_95 = 10)
sel_weight_log <- SelectivityAtWeight(weights, SW50 = 5, SW50_95 = 2)

# ---- Knife-Edge Selectivity ----
```

```
sel_age_ke    <- SelectivityKnifeEdgeAge(ages, SA = 4)
sel_length_ke <- SelectivityKnifeEdgeLength(lengths, SL = 50)


# ---- Double-Normal Selectivity ----
sel_dn_length <- DoubleNormal(lengths, L5 = 20, LFS = 50, Vmaxlen = 0.8)
sel_dn_weight <- DoubleNormalWeight(weights, W5 = 1, WFS = 5, Vmaxweight = 0.8)


# ---- Plots ----
# Age-based
plot(ages, sel_age_log, type='l', lwd=2, col='blue', ylim=c(0,1),
     main='Selectivity-at-Age', xlab='Age', ylab='Selectivity')
lines(ages, sel_age_ke, type='s', lwd=2, col='red')
legend("bottomright", legend=c("Logistic","Knife-Edge"), col=c("blue","red"), lwd=2)

# Length-based
plot(lengths, sel_length_log, type='l', lwd=2, col='blue', ylim=c(0,1),
     main='Selectivity-at-Length', xlab='Length', ylab='Selectivity')
lines(lengths, sel_length_ke, type='s', lwd=2, col='red')
lines(lengths, sel_dn_length, lwd=2, col='green')
legend("bottomright", legend=c("Logistic","Knife-Edge","Double-Normal"),
       col=c("blue","red","green"), lwd=2)

# Weight-based
plot(weights, sel_weight_log, type='l', lwd=2, col='blue', ylim=c(0,1),
     main='Selectivity-at-Weight', xlab='Weight', ylab='Selectivity')
lines(weights, sel_dn_weight, lwd=2, col='green')
legend("bottomright", legend=c("Logistic","Double-Normal"),
       col=c("blue","green"), lwd=2)
```

---

`Sense` *Sensitivity analysis*

---

## Description

A function that determines the inputs for a given data-limited method of class Output and then analyses the sensitivity of TAC estimates to marginal differences in each input. The range used for sensitivity is based on the user-specified CV for that input (e.g. CV_Mort, Mort)

## Usage

```
Sense(Data, MP, nsense = 6, reps = 100, perc = c(0.05, 0.5, 0.95), ploty = T)
```

## Arguments

| | |
|---|---|
| `Data` | A data-limited methods data object |
| `MP` | A character string representing an MP applied in calculating the TAC recommendations in the DLM object |

| | |
|---|---|
| nsense | The number of points over which to calculate the TAC (resolution) |
| reps | The number of samples of the quota taken for the calculation of the TAC |
| perc | The percentile of the sample TAC |
| ploty | A logical switch, (T/F, should a plot be drawn?) |

### Author(s)

T. Carruthers

### Examples

```
## Not run:
Data <- Sense(MSEtool::Cobia, "AvC")

## End(Not run)
```

---

## SetMPArgs *Set Argument Values of a Management Procedure*

---

### Description

Returns a copy of a management procedure (MP) with new default values for some of its arguments, e.g. to define a variant of a built-in MP with different settings.

### Usage

```
SetMPArgs(MP, ...)
```

### Arguments

| | |
|---|---|
| MP | An MP function (class mp or mmp), or the name of one. |
| ... | Named argument values. Each name must be an argument of MP (unless MP has a ... argument). A value of NULL sets that argument's default to NULL. |

### Value

A function of the same class as MP.

### See Also

Project(), TuneMP()

### Examples

```
IR5 <- SetMPArgs(IndexRate, CalibYears = 5, Smooth = FALSE)
formals(IR5)$CalibYears

## Not run:
MSE <- Project(Hist, MPs = list(IR5 = IR5, IR2 = IndexRate))

## End(Not run)
```

---

## SetSeasonalR0 *Set Seasonal R0 Using a Wrapped Normal Distribution*

---

### Description

Distributes annual unfished recruitment (`AnnualR0`) across seasons using a wrapped normal distribution, producing an `nSim x nSeason` matrix of per-season R0 values that sum to `AnnualR0` for each simulation.

### Usage

```
SetSeasonalR0(AnnualR0, Seasons, PeakSeason, Sigma, Years = NULL)
```

### Arguments

| | |
|---|---|
| `AnnualR0` | Numeric. Annual unfished recruitment. Accepted forms: a scalar (same value for all simulations), a numeric vector of length `nSim`, or an `nSim x 1` matrix. Per-season values sum to `AnnualR0` for each simulation. |
| `Seasons` | Integer. Number of seasons per year (must match `OM@Seasons`). |
| `PeakSeason` | Numeric. Season of peak recruitment. Doesn't need to be an integer; e.g. `3.5` places the peak midway between seasons 3 and 4. Passed as `mu` to `WrappedNormal()`. |
| `Sigma` | Numeric. Spread of recruitment around the peak, in units of seasons. Smaller values concentrate recruitment; larger values spread it across the year. Passed as `sigma` to `WrappedNormal()`. |
| `Years` | Optional numeric vector of year labels, e.g. from `Years(OM, 'Historical')`. If provided, the `Year` dimension of the returned matrix is named using the first `Seasons` elements (`Years[1:Seasons]`). If `NULL` (default), dimension names are omitted. |

### Details

Seasonal recruitment is controlled entirely by `SRR@R0`: at unfished equilibrium the stock-recruit relationship returns exactly `R0[m]` recruits in season `m`. Setting `R0[m] = pi_m * AnnualR0` via this function is therefore both necessary and sufficient to produce the target seasonal recruitment pattern. Optionally call `AdjustSeasonalFecundity()` afterwards to also concentrate reported seasonal spawning production in the same seasons, reflecting the biological reality that fish only spawn during certain months.

### Value

An `nSim x nSeason` numeric matrix of per-season R0 values. If `Years` is supplied, `dimnames = list(Sim = 1:nSim, Year = Years[1:Seasons])`.

### See Also

`WrappedNormal()`, `AdjustSeasonalFecundity()`

---

## `setup` *Setup parallel processing*

---

### Description

Sets up parallel processing using the snowfall package

### Usage

```
setup(cpus = NULL, logical = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `cpus` | the number of CPUs to use for parallel processing. If left empty all physical cores will be used, unless `logical=TRUE`, in which case both physical and logical (virtual) cores will be used. |
| `logical` | Use the logical cores as well? Using the virtual cores may not lead to any significant decrease in run time. |
| `...` | other arguments passed to 'snowfall::sfInit' |

### Examples

```
## Not run:
setup() # set-up the physical processors
setup(6) # set-up 6 processors
setup(logical=TRUE) # set-up physical and logical cores

## End(Not run)
```

`SetupParallel` *Set Up Parallel Processing for openMSE Workflows*

## Description

Configure the global parallel processing strategy used throughout the openMSE framework via the `future` backend.

## Usage

```
SetupParallel(
  workers = future::availableCores(),
  backend = c("auto", "multisession", "multicore", "sequential"),
  max_workers = c("physical", "logical"),
  silent = FALSE
)

DisableParallel(silent = FALSE)

CheckParallel(parallel)
```

## Arguments

| | |
|---|---|
| `workers` | Integer. Number of parallel workers. Defaults to `future::availableCores()`. |
| `backend` | Character. Parallel backend to use. One of `"auto"` (default), `"multisession"`, `"multicore"`, or `"sequential"`. `"auto"` selects a safe backend based on the operating system. |
| `max_workers` | Character. Which CPU cores to count when determining the worker ceiling. `"physical"` (default) limits to physical cores; `"logical"` allows hyperthreads. Using `"physical"` is generally recommended. |
| `silent` | Logical. If `FALSE` (default), prints a message describing the active parallel plan. |
| `parallel` | Logical. Has `parallel` been requested? |

## Details

These functions are intended to be called at the beginning (and optionally the end) of a workflow:

- `SetupParallel()` establishes a parallel execution plan
- `DisableParallel()` restores sequential execution
- `CheckParallel()` validates that a parallel plan is active

`SetupParallel()` spawns background worker processes that persist until they are explicitly shut down. Always call `DisableParallel()` once your workflow is finished. If a session ends abnormally (crash, forced kill) before that happens, its workers are orphaned and keep

running; use `ListParallelWorkers()` to check for these and `CleanupOrphanWorkers()` to terminate them.

By default, `SetupParallel()` selects a safe, platform-appropriate backend:

- **Windows**: `multisession`
- **macOS / Linux**: `multisession` (default), with optional `multicore` support

Although `future::multicore` is often faster on Unix-like systems, `multisession` is used by default because it is more robust.

**Global side effects:**

`SetupParallel()` modifies the global future plan for the current R session. All packages relying on `future` (including `furrr`) will inherit this plan until it is changed or reset with `DisableParallel()`.

**Backend options:**

- `"multisession"` — background R sessions; safe and portable on all platforms
- `"multicore"` — forked processes (macOS / Linux only); faster for large workloads but may be unsafe with certain compiled code or external pointers
- `"sequential"` — disables parallel processing entirely

**Recommended usage:**

```
SetupParallel()          # start of workflow
# ... analysis code ...
DisableParallel()        # restore sequential execution
```

## Value

All three functions invisibly return a logical scalar:

- `SetupParallel()` and `DisableParallel()` return `TRUE`
- `CheckParallel()` returns `FALSE` if `parallel = FALSE`, `TRUE` if `parallel = TRUE` and a parallel plan is active, and throws an error if `parallel = TRUE` but no parallel plan is active

## See Also

`future::plan()`, `future::availableCores()`, `furrr::future_map()`, `ListParallelWorkers()`, `CleanupOrphanWorkers()`

## Examples

```
## Not run:
SetupParallel()                       # auto-select backend
SetupParallel(workers = 4)            # explicit worker count
SetupParallel(backend = "multicore")  # multicore on Linux/macOS
DisableParallel()                     # restore sequential execution

## End(Not run)
```

---

`show-methods` *Show Methods*

---

### Description

Display a formatted summary of MSEtool S4 objects in the console.

### Arguments

| | |
|---|---|
| `object` | An S4 object of the relevant class. |

---

`show-MSEtool` *Show MSEtool S4 objects*

---

### Description

Briefly prints a couple of lines from str to avoid swamping the console with the contents of very large objects.

### Usage

```
## S4 method for signature 'Data'
show(object)

## S4 method for signature 'OM'
show(object)

## S4 method for signature 'Hist'
show(object)

## S4 method for signature 'MSE'
show(object)

## S4 method for signature 'MOM'
show(object)

## S4 method for signature 'MMSE'
show(object)
```

### Arguments

| | |
|---|---|
| `object` | S4 object from MSEtool |

---

`show,PMobj-method` *Show the output of a PM*

---

### Description

Show the output of a PM

### Usage

```
## S4 method for signature 'PMobj'
show(object)
```

### Arguments

| | |
|---|---|
| `object` | object of class MSE |

---

`show,Rec-method` *Show the output of a single MP recommendation*

---

### Description

Show the output of a single MP recommendation

### Usage

```
## S4 method for signature 'Rec'
show(object)
```

### Arguments

| | |
|---|---|
| `object` | object of class Rec |

---

`SIL` *Slot in list: get the slot values from a list of objects*

---

### Description

Create of vector of values that correspond with a slot in a list of objects

### Usage

```
SIL(listy, sloty)
```

### Arguments

| | |
|---|---|
| `listy` | A list of objects |
| `sloty` | A character vector representing the slot name |

### Author(s)

T. Carruthers

---

`simCAA` *Simulate Catch-at-Age Data*

---

### Description

CAA generated with either a multinomial or logistic normal observation model from retained catch-at-age array

### Usage

```
simCAA(nsim, yrs, n_age, Cret, CAA_ESS, CAA_nsamp)
```

### Arguments

| | |
|---|---|
| `nsim` | Number of simulations |
| `yrs` | Number of years |
| `n_age` | Number of age classes |
| `Cret` | Retained Catch at age in numbers - array(sim, years, maxage+1) |
| `CAA_ESS` | CAA effective sample size. If greater than 1, then this is the multinomial distribution sample size. If less than 1, this is the coefficient of variation for the logistic normal distribution (see details). |
| `CAA_nsamp` | CAA sample size |

### Details

The logistic normal generates the catch-at-age sample by first sampling once from a multivariate normal distribution with the mean vector equal to the logarithm of the proportions-at-age and the diagonal of the covariance matrix is the square of the product of the CV and the log proportions (all off-diagonals are zero). The sampled vector is then converted to proportions with the softmax function and expanded to numbers (CAA_nsamp). This method allows for simulating fractional values in the catch-at-age matrix.

### Value

CAA array

`simCAL` *Simulate Catch-at-Length Data*

## Description

Simulate CAL and calculate length-at-first capture (LFC), mean length (ML), modal length (Lc), and mean length over modal length (Lbar)

## Usage

```
simCAL(
  nsim,
  nyears,
  maxage,
  CAL_ESS,
  CAL_nsamp,
  nCALbins,
  CAL_binsmid,
  CAL_bins,
  vn,
  retL,
  Linfarray,
  Karray,
  t0array,
  LenCV
)
```

## Arguments

| | |
|---|---|
| `nsim` | Number of simulations |
| `nyears` | Number of years |
| `maxage` | Maximum age |
| `CAL_ESS` | CAA effective sample size |
| `CAL_nsamp` | CAA sample size |
| `nCALbins` | number of CAL bins |
| `CAL_binsmid` | mid-points of CAL bins |
| `CAL_bins` | Boundary of CAL bins |
| `vn` | Vulnerable numbers-at-age |
| `retL` | Retention at length curve |
| `Linfarray` | Array of Linf values by simulation and year |
| `Karray` | Array of K values by simulation and year |
| `t0array` | Array of t0 values by simulation and year |
| `LenCV` | CV of length-at-age#' |

### Value

named list with CAL array and LFC, ML, & Lc vectors

---

`SimControl` *Simulation Control Options*

---

### Description

Constructs a named list of optional-calculation switches for `Simulate()`. Pass the result to the `control` argument of `Simulate()` to toggle which computations are performed.

### Usage

```
SimControl(
  DynamicUnfished = TRUE,
  RefLandings = FALSE,
  RefRemovals = FALSE,
  ConditionObs = TRUE,
  EstimateBeta = FALSE,
  GenerateData = TRUE,
  MSYRefs = TRUE,
  MSYRefsCpp = TRUE,
  RefPoints = TRUE,
  MGT = TRUE,
  BLow = FALSE,
  CalcCatchAtSize = NA,
  CalcCatchAtSizeCpp = TRUE,
  ...
)
```

### Arguments

`DynamicUnfished`
: Logical. Calculate the dynamic unfished population dynamics? Default `TRUE`.

`RefLandings`
: Logical. Calculate reference yield based on landings? Computationally expensive. Default `FALSE`.

`RefRemovals`
: Logical. Calculate reference yield based on total removals (landings + discards)? Default `FALSE`.

`ConditionObs`
: Logical. Condition the observation model on historical fishery data? Default `TRUE`.

`EstimateBeta`
: Logical. Estimate the index hyperstability/ hyperdepletion parameter `Beta` by regression when conditioning on real data (see `IndicesObs()`)? Default `FALSE`. If `FALSE`, `Beta` is fixed at `1` unless the user already supplied a value for that index. If `TRUE`, `Beta` is estimated per simulation; see `EstimateBeta()` and `Index_Obs@Misc$BetaFit` for diagnostics.

GenerateData — Logical. Generate historical fishery data from the observation model? Default TRUE.

MSYRefs — Logical. Calculate MSY-based reference points (see CalcMSY())? Numerically optimizes apical F per simulation, so cost scales with nSim; use parallel = TRUE in Simulate() for large nSim. Default TRUE.

MSYRefsCpp — Logical. Use the C++ implementation of the MSY reference point calculation instead of the R implementation (CalcMSY())? Default TRUE.

RefPoints — Logical. Calculate F0.1, Fmax, Fx%SPR, Fmed, Fcrash and SPRcrash reference points (see CalcRefPoints())? Default TRUE.

MGT — Logical. Calculate mean generation time (see CalcMGT())? Default TRUE.

BLow — Logical. Calculate the BLow rebuilding reference point (see CalcBLow())? Substantially more expensive than the other reference points. Default FALSE.

CalcCatchAtSize — Logical, or NA. Calculate historical and projected landings/discards-at-size? Computationally expensive for models with many fleets and/or fine seasonal time steps. NA (default) auto-detects, per stock: skipped unless either a fleet's size-composition observation model is configured (relevant when GenerateData = TRUE) or real size-composition data is supplied for conditioning (relevant when ConditionObs = TRUE). Set TRUE to always calculate it regardless or FALSE to always skip it.

CalcCatchAtSizeCpp — Logical. Use the C++ implementation of the landings/discards-at-size projection instead of its pure-R equivalent? Default TRUE. Set FALSE as a fallback if the C++ version has problems

... — Additional named arguments. Any unrecognised names trigger a warning.

## Value

A named list of simulation control settings.

## See Also

Simulate(), CalcRefPoints(), CalcMSY(), CalcMGT(), CalcBLow()

## Examples

```
# Default settings
SimControl()

# Skip MSY reference points and reference yield calculations
SimControl(MSYRefs = FALSE)

# Calculate reference yield based on landings
SimControl(RefLandings = TRUE)

# Also calculate the (expensive) BLow rebuilding reference point
SimControl(BLow = TRUE)
```

`simmov` *Calculates movement matrices from user inputs*

## Description

A wrapper function for makemov used to generate movement matrices for the operating model. Calculates a movement matrix from user-specified unfished stock biomass fraction in each area and probability of staying in the area in each time step.

## Usage

```
simmov(
  OM,
  dist = c(0.1, 0.2, 0.3, 0.4),
  prob = 0.5,
  distE = 0.1,
  probE = 0.1,
  prob2 = NA,
  figure = TRUE
)

plot_mov(mov, age = 1, type = c("matrix", "all"), year = 1, qval = 0.9)
```

## Arguments

| | |
|---|---|
| `OM` | Operating model, an object of class `OM`. |
| `dist` | A vector of fractions of unfished stock in each area. The length of this vector will determine the number of areas (`nareas`) in the OM. |
| `prob` | Mean probability of staying across all areas (single value) or a vector of the probability of individuals staying in each area (same length as dist). |
| `distE` | Logit (normal) St.Dev error for sampling stock fractions from the fracs vector |
| `probE` | Logit (normal) St.Dev error for sampling desired probability of staying either by area (prob is same length as dist) or the mean probability of staying (prob is a single number). |
| `prob2` | Optional vector as long as prob and dist. Upper bounds on uniform sampling of probability of staying, lower bound is prob. |
| `figure` | Logical to indicate if the movement matrix will be plotted (mean values and range across `OM@nsim` simulations.) |
| `mov` | A four-dimensional array of dimension `c(nsim, maxage, nareas, nareas)` or a five-dimensional array of dimension `c(nsim, maxage, nareas, nareas, nyears + proyears)` specifying movement in the operating model. |

| | |
|---|---|
| age | An age from 0 to maxage for the movement-at-age matrix figure when type = "matrix". |
| type | Whether to plot a movement matrix for a single age ("matrix") or the full movement versus age figure ("all") |
| year | If mov is a 5-dimensional array, the year (from 1 to nyears + proyears) for which to plot movement. |
| qval | The quantile to plot or report the range of values among simulations. |

### Value

The operating model OM with movement parameters in slot cpars. The mov array is of dimension nsim, maxage, nareas, nareas.

### Functions

- simmov(): Estimation function for creating movement matrix.
- plot_mov(): Plotting function.

### Note

Array mov is age-specific, but currently the movement generated by simmov is independent of age.

### Author(s)

T. Carruthers and Q. Huynh

### Examples

```
## Not run:
movOM_5areas <- simmov(testOM, dist = c(0.01,0.1,0.2,0.3,0.39), prob = c(0.1,0.6,0.6,0.7,0.9))
movOM_5areas@cpars$mov[1, 1, , ] # sim 1, age 1, movement from areas in column i to areas in row j
plot_mov(movOM_5areas@cpars$mov)
plot_mov(movOM_5areas@cpars$mov, type = "all")


## End(Not run)
```

---

## simmov2 *Calculates movement matrices from user specified distribution among other areas*

---

### Description

A wrapper function for makemov2 used to generate movement matrices for the operating model. Calculates a movement matrix from user-specified relative movement to other areas and probability of staying in the area in each time step.

### Usage

```
simmov2(
  OM,
  dist = c(0.05, 0.6, 0.35),
  distE = 0.01,
  frac_other = matrix(c(NA, 2, 1, 3, NA, 1, 1, 4, NA), nrow = 3, byrow = T),
  frac_otherE = 0.01,
  prob = 0.8,
  probE = 1,
  figure = TRUE
)
```

### Arguments

| | |
|---|---|
| `OM` | Operating model, an object of class `OM`. |
| `dist` | A vector of fractions of unfished stock in each area. The length of this vector will determine the number of areas (`nareas`) in the OM. |
| `distE` | Logit (normal) St.Dev error for sampling desired fraction in each area |
| `frac_other` | A matrix (nareas rows from, nareas columns to) of relative fractions moving to other areas (the positive diagonal (staying) is unspecified). |
| `frac_otherE` | Logit (normal) St.Dev error for sampling desired fraction moving to other areas. |
| `prob` | the mean probability of staying in the same area among all areas |
| `probE` | Logit (normal) St.Dev error for sampling desired probability of staying in each area |
| `figure` | Logical to indicate if the movement matrix will be plotted (mean values and range across `OM@nsim` simulations.) |

### Value

The operating model `OM` with movement parameters in slot `cpars`. The `mov` array is of dimension `nsim`, `maxage`, `nareas`, `nareas`.

### Functions

- `simmov2()`: Estimation function for creating movement matrix.

### Note

Array `mov` is age-specific, but currently the movement generated by `simmov` is independent of age.

### Author(s)

T. Carruthers and Q. Huynh

### Examples

```
## Not run:
movOM_3areas <- simmov2(testOM, frac_other = matrix(c(NA,2,1, 2,NA,1, 1,2,NA),
nrow=3, byrow=T), frac_otherE = 0.01, prob = 0.8, probE = 0.3)
# sim 1, age 1, movement from areas in column i to areas in row j
movOM_3areas@cpars$mov[1, 1, , ]
plot_mov(movOM_3areas@cpars$mov)
plot_mov(movOM_3areas@cpars$mov, type = "all")


## End(Not run)
```

---

## Simulate *Simulate Operating Model Dynamics*

---

### Description

Simulate historical fishery dynamics and calculate reference points for an om or OM-legacy object. Returns a populated hist or Hist-class object containing historical population dynamics, fishery data, and reference points.

### Usage

```
Simulate(
  OM = NULL,
  parallel = FALSE,
  silent = FALSE,
  nSim = NULL,
  nsim = NULL,
  control = SimControl(),
  Reduce = TRUE,
  refpointsMSY = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| `OM` | An om or OM-legacy object. If `NULL` (default), `MSEtool::SingleStockOM` is used. |
| `parallel` | Logical. Use parallel processing (across simulations) for MSY reference point calculation (see `CalcMSY()`), final-depletion catchability calibration, reference yield calculation, and historical data generation? Requires a `future` plan established by `SetupParallel()` – errors if `TRUE` and no plan is active. Only used for om objects. Default `FALSE`. |
| `silent` | Logical. Suppress progress messages if `TRUE`. Default `FALSE`. |
| `nSim` | Integer. Number of simulation replicates. If `NULL` (default), the value in `OM` is used. Only used for om objects; use `nsim` for OM-legacy objects. |

| | |
|---|---|
| `nsim` | Integer. Synonym for `nSim` for OM-legacy objects. If `NULL` (default), `nSim` is used if provided. |
| `control` | A `SimControl()` object controlling which optional calculations are performed. Only used for om objects. Default `SimControl()`. |
| `Reduce` | Logical. Reduce object size after simulation for memory efficiency? Default `TRUE`. See `ReduceDims()`. |
| `refpointsMSY` | A pre-calculated refpointsMSY object (e.g. from a previous `Simulate()` or `CalcMSY()` call), or `NULL` (default). If supplied, it is used directly as `Hist@Reference@MSY` and the (expensive) MSY reference point calculation is skipped – `control`'s `MSYRefs` setting is ignored in this case. Only valid if it was calculated for an `OM` with the same `nSim`, stock/complex names, and reference year as the `OM` supplied here; an incompatible object throws an error rather than being silently ignored. Only used for om objects. |
| `...` | Additional arguments passed to sub-functions. Not currently used |

## Details

`Simulate()` accepts both om and legacy OM-legacy objects, dispatching on the class of `OM`.

`OM@Control:`

`OM@Control` holds optional settings that modify the simulated dynamics, such as the basis for MSY reference points (`MSYType`), the years they are calculated at (`RefYears`), and which operating model quantities are passed to MPs (`DataOM`). See OMControl.

## Value

- If `OM` is an om object: a hist object.
- If `OM` is an OM-legacy object: a Hist-legacy object.

## See Also

`OM()`, `runMSE()`

## Examples

```
## Not run:
# Simulate using the default example OM
hist <- Simulate()

# Simulate with specific options
hist <- Simulate(
  OM      = MyOM,
  control = SimControl(RefLandings = TRUE, GenerateData = FALSE),
  silent  = TRUE
)

## End(Not run)
```

---

`SimulatedData` *SimulatedData Data*

---

### Description

An object of class Data

### Usage

```
SimulatedData
```

### Format

An object of class `Data` of length 1.

---

`SimulateMOM` *Run a multi-fleet multi-stock Management Strategy Evaluation*

---

### Description

Functions for running a multi-stock and/or multi-fleet Management Strategy Evaluation (closed-loop simulation) for a specified operating model

### Usage

```
SimulateMOM(MOM = MSEtool::Albacore_TwoFleet, parallel = TRUE, silent = FALSE)

ProjectMOM(
  multiHist = NULL,
  MPs = NA,
  parallel = FALSE,
  silent = FALSE,
  checkMPs = FALSE,
  dropHist = FALSE,
  extended = FALSE
)

multiMSE(
  MOM = MSEtool::Albacore_TwoFleet,
  MPs = list(list(c("AvC", "DCAC"), c("FMSYref", "curE"))),
  Hist = FALSE,
  silent = FALSE,
  parallel = TRUE,
  checkMPs = FALSE,
  dropHist = TRUE,
  extended = FALSE
)
```

### Arguments

| | |
|---|---|
| `MOM` | A multi-fleet multi-stock operating model (class `MOM`) |
| `parallel` | Logical or a named list. Should MPs be run using parallel processing? See Details for more information. |
| `silent` | Should messages be printed out to the console? |
| `multiHist` | An Historical Simulation object (class `multiHist`) |
| `MPs` | A matrix of methods (nstock x nfleet) (character string) of class MP |
| `checkMPs` | Logical. Check if the specified MPs exist and can be run on `SimulatedData`? |
| `dropHist` | Logical. Drop the (very large) `multiHist` object from the returned `MMSE` object? The `multiHist` object can be (re-)created using `SimulateMOM` or kept in `MMSE@multiHist` if `dropHist=FALSE` |
| `extended` | Logical. Return extended projection results? if TRUE, `MMSE@Misc$extended` is a named list with extended data: FM for overall F across fleets `[nsim, nstock, n_age, nMP,` `Misc` slot in `MMSE@PPD` will also contain StockPars, FleetPars, and ReferencePoints |
| `Hist` | Should model stop after historical simulations? Returns a list containing all historical data |

### Details

**Running MPs in parallel:**

For most MPs, running in parallel can actually lead to an increase in computation time, due to the overhead in sending the information over to the cores. Consequently, by default the MPs will not be run in parallel if `parallel=TRUE` (although other internal code will be run in parallel mode).

To run MPs in parallel, specify a named list with the name of the MP(s) assigned as TRUE. For example,`parallel=list(AvC=TRUE)` will run the `AvC` MP in parallel mode.

### Value

Functions return objects of class `MMSE` and `multiHist` #'

- SimulateMOM - An object of class `multiHist`
- ProjectMOM - An object of class `MMSE`
- multiMSE - An object of class `MMSE`

### Functions

- `SimulateMOM()`: Simulate historical dynamics for multi-OM
- `ProjectMOM()`: Run Forward Projections for a `MOM` object
- `multiMSE()`: Run a multi-stock, multi-fleet MSE

### Author(s)

T. Carruthers and A. Hordyk

`SingleStockOM` *Single Stock Example Operating Model*

## Description

An example om object representing the simplest operating model structure: one stock and one fleet. Intended to illustrate basic OM construction and use in `MSEtool`. Pairings of stocks, fleets, and observation models are for illustration only and should not be taken as ecologically meaningful. See also TwoFleetOM, ComplexOM, and MultiStockOM for more complex structural examples.

## Usage

```
SingleStockOM
```

## Format

An om object with the following slots populated:

- `Name`: `"Single Stock - Single Fleet"`.
- `nSim`: `8`. Number of stochastic simulations.
- `nYear`: `20`. Number of historical years.
- `pYear`: `30`. Number of projection years.
- `Stock`: AlbacoreExStock. A single stock object.
- `Fleet`: a length-1 list containing a length-1 list of AsympExFleet. Structure is `[[stock]][[fleet]]`.
- `Obs`: a length-1 list containing a length-1 list of AgeStructuredObs. Structure is `[[stock]][[fleet]]`.
- `Imp`: a length-1 list containing a length-1 list of FullComplianceImp. Structure is `[[stock]][[fleet]]`. Perfect TAC/ effort implementation and full immediate size-regulation compliance.

## Details

**Stock:**
AlbacoreExStock represents a long-lived, slow-growing pelagic species with low natural mortality. See AlbacoreExStock for full parameter details.

**Fleet:**
AsympExFleet represents a fleet with asymptotic length-based selectivity and effort that stabilises early in the historical period. See AsympExFleet for full parameter details.

**Observation Model:**
AgeStructuredObs provides landings, a spawning-biomass survey, and annual age-composition samples from both landed and discarded fish — a data-moderate scenario typical of stocks assessed with a statistical catch-at-age model. See AgeStructuredObs for full parameter details.

**Implementation Error:**
FullComplianceImp models perfect implementation: `TAC@Mean = 1/ SD = 0` and `Effort@Mean = 1/SD = 0` mean realised TAC/effort always equal advice exactly, and `Size@Compliance = 1` means any size-regulation advice (a change to `Advice@Retention`/`Advice@Selectivity`) is adopted immediately by the whole fleet. See `Imp()` for details, and OverageImp, UnderageImp, PartialSizeComplianceImp for contrasting examples.

## See Also

TwoFleetOM, ComplexOM, MultiStockOM for structural variants. AlbacoreExStock, AsympExFleet, AgeStructuredObs, FullComplianceImp for the component objects. `OM()`, om, `runMSE()`, `PopulateOM()`, `ExampleMPs()`

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `TwoFleetOM`, `om-class`, `stocktransition-class`

## Examples

```
SingleStockOM
Stock(SingleStockOM)
Fleet(SingleStockOM)

## Not run:
Hist <- runMSE(SingleStockOM, MPs = ExampleMPs())

## End(Not run)
```

---

`SketchFun` *Manually map the historical relative fishing effort trajectory.*

---

## Description

Internal function for interactive plot which allows users to specify the relative trajectory and variability in the historical fishing effort.

## Usage

```
SketchFun(nyears, Years=NULL)
```

## Arguments

| | |
|---|---|
| `nyears` | Number of years |
| `Years` | An optional vector of years. Should be nyears long. |

## Author(s)

A. Hordyk

`SmoothSeries` *Smooth a Time Series with a Loess Polynomial Smoother*

## Description

Fits a `stats::loess()` smoother to a numeric vector, for use inside a management procedure to reduce the influence of observation noise in a data series (an index, effort, catch, etc.) before it is used to set advice. `NA`s in `x` are dropped before fitting and left as `NA` in the returned vector; all other positions are replaced by their fitted value.

## Usage

```
SmoothSeries(x, Time = NULL, ENPMult = 0.3, Log = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | Numeric vector to smooth. May contain `NA`s. When `Log = TRUE` (the default), every non-`NA` value must be strictly positive. |
| `Time` | Numeric vector the same length as `x` giving the position of each observation (e.g. a vector of years). `NULL` (default) uses `seq_along(x)`, i.e. assumes `x` is regularly spaced. |
| `ENPMult` | Fraction. Effective-number-of-parameters multiplier controlling the degree of smoothing; see Details. Default `0.3`. |
| `Log` | Logical. If `TRUE` (default), the smoother is fit in log-space (`log(x)`) and back-transformed, appropriate for series that are strictly positive and multiplicative in their variability (most abundance indices, catch, and effort series). If `FALSE`, the smoother is fit to `x` directly. |

## Details

The number of effective parameters used by the smoother is `sum(!is.na(x)) * ENPMult`, so the degree of smoothing scales with the length of the series: `ENPMult` close to `0` gives a very smooth fit (few effective parameters), while `ENPMult` close to `1` tracks the observations closely.

## Value

A numeric vector the same length as `x`, with smoothed values in place of every non-`NA` input and `NA` retained where `x` was `NA`.

## See Also

`stats::loess()`, `IndexRate()`, `IndexTarget()`

### Examples

```
set.seed(1)
index <- 1.5 * exp(-0.05 * (1:20)) * rlnorm(20, 0, 0.2)
index[15] <- NA # a missed survey year
SmoothSeries(index, ENPMult = 0.3)

# An irregularly-spaced series (e.g. surveys in specific calendar years)
years <- c(2001, 2003, 2004, 2008, 2012, 2015, 2018, 2020)
SmoothSeries(index[1:8], Time = years, ENPMult = 0.5)

# A series that isn't strictly positive/multiplicative (e.g. an anomaly)
SmoothSeries(rnorm(20), Log = FALSE)
```

---

## smoothy *General purpose polynomial smoother*

---

### Description

Polynomial smoother (no gradient prediction) applied to a vector that can include NA values. Intended to be rapid for use in management procedures

### Usage

```
smoothy(xx, plot = F, enp_mult, plotname = "", xlab = "x", ylab = "y", x = NA)
```

### Arguments

| | |
|---|---|
| xx | Vector of real numbers, data to be smoothed. |
| plot | Logical, should the 'fit' of the smoother be plotted? |
| enp_mult | Fraction, effective number of parameters multiplier. The smoother parameter number is length(xx) x enp_mult. So higher values of enp_mult means less smoothing (more parameters). |
| plotname | Character, in case you want to put a label on the plot (plot = T). |
| xlab | Character, in case you want an xaxis label on the plot (plot = T) |
| ylab | Character, in case you want a yaxis label on the plot (plot = T) |
| x | Numeric vector same length as xx, in case you want to have a custom xaxis (e.g. years) |

### Author(s)

T. Carruthers

---

`Spatial` *Spatial Constructor and Accessors*

---

## Description

Construct a spatial object defining the spatial structure and movement dynamics for a stock, or access and replace the `Spatial` slot of a stock and its individual slots. `Spatial` is optional — leave it empty for non-spatial (single-area) models.

## Usage

```
Spatial(
  UnfishedDist = NULL,
  ProbStaying = NULL,
  RelativeSize = NULL,
  Movement = NULL,
  FracOther = NULL,
  Arrangement = NULL,
  CVDist = 0.1,
  CVStay = 1,
  Misc = list()
)

UnfishedDist(x)

UnfishedDist(x) <- value

ProbStaying(x)

ProbStaying(x) <- value

RelativeSize(x)

RelativeSize(x) <- value

Movement(x)

Movement(x) <- value

FracOther(x)

FracOther(x) <- value

Arrangement(x)

Arrangement(x) <- value
```

```
CVDist(x)

CVDist(x) <- value

CVStay(x)

CVStay(x) <- value

Spatial(x) <- value
```

### Arguments

| | |
|---|---|
| `UnfishedDist` | `numeric`, `array`, or `NULL`. Relative unfished biomass distribution across areas. For two-area models, a single value giving the fraction in Area 1 (Area 2 is derived as `1 - UnfishedDist`). For multi-area models, must sum to 1 across areas. Accepted formats:<br>• Scalar: constant across all simulations, ages, and years.<br>• Length-2 bounds vector: sampled from `Uniform(lower, upper)` once per simulation (two-area models only).<br>• Named array `Sim × Area × Age` or `Sim × Area × Age × Year`: full age- and time-varying specification. When `UnfishedDist` is a stock object, `Spatial()` acts as a pass-through accessor and returns `x@Spatial`. Default `NULL`. |
| `ProbStaying` | `numeric` or `array`. Probability of remaining in an area between time steps. For two-area models, specifies Area 1 only; the Area 2 value is solved internally. Accepts the same formats as `UnfishedDist`. Default `NULL`. |
| `RelativeSize` | `numeric`, `array`, `character(1)`, or `NULL`. Relative size of each area, used for biomass density calculations. For two-area models, a single value gives the fraction of Area 1 (Area 2 = `1 - RelativeSize`). For multi-area models, a vector of length `nArea` summing to 1, or a `Sim × Area` array. The special value `"EqualDensity"` sets `RelativeSize` equal to the mean `UnfishedDist` across ages and years, producing constant density across areas regardless of their relative size. When `NULL` (default), all areas are assumed equal in size. |
| `Movement` | `array` or `NULL`. Movement probability matrix with dimensions `Sim × FromArea × ToArea`, optionally extended to `Sim × FromArea × ToArea × Age × Year`. Row `[i, ]` gives the probabilities of moving from area `i` to each area; rows must sum to 1. When supplied, the asymptotic unfished distribution is derived from `Movement` and assigned to `UnfishedDist`, overwriting any existing values. When `NULL` (default), the movement matrix is fitted during `Populate()` from `UnfishedDist`, `ProbStaying`, and `FracOther`. |
| `FracOther` | `array` or `NULL`. Required for models with more than two areas. Array with dimensions `Sim × Area × Area`, optionally extended to include `Age` and `Year` dimensions. Diagonal elements must be `NA` (staying probabilities are taken from `ProbStaying`). Off-diagonal element `[i, j]` gives the relative probability of moving from area `i` to area `j`; values need not sum to 1 as they are normalised internally. Default `NULL`. |

| | |
|---|---|
| `Arrangement` | `array` or `NULL`. Spatial layout of areas for plotting purposes. Reserved for future use; currently stored but not applied. Default `NULL`. |
| `CVDist` | `numeric(1)`. Standard deviation of the log-scale penalty on the difference between the optimised asymptotic distribution and the target `UnfishedDist` during multi-area movement optimisation. Larger values allow the fitted movement matrix to deviate further from the target distribution. Has no effect for two-area models. Default `0.1`. |
| `CVStay` | `numeric(1)`. Standard deviation of the logit-scale penalty on the difference between the optimised diagonal (staying probability) and the target `ProbStaying` during multi-area movement optimisation. Larger values allow more deviation from the target staying probability. Has no effect for two-area models. Default `1`. |
| `Misc` | `list`. Used internally. Default `list()`. |
| `x` | A spatial object for slot accessors, or a stock object for `Spatial<-`. |
| `value` | For `Spatial<-`: a spatial object. For slot replacement functions: the replacement value for the corresponding slot. |

## Details

### Default Behaviour for Non-Spatial Models:

`Spatial` is optional. When the `Spatial` slot of a stock is empty, `Populate()` creates a default single-area structure automatically — no `Spatial` object needs to be specified for non-spatial models.

### Two-Area Models:

Specify `UnfishedDist` as the unfished fraction in Area 1 and `ProbStaying` as the probability of remaining in Area 1. `Populate()` solves for the Area 2 movement probabilities numerically so that the asymptotic distribution matches `UnfishedDist` and the Area 1 staying probability matches `ProbStaying`:

```
# Fixed parameters across all simulations
sp <- Spatial(UnfishedDist = 0.3,   # 30% of unfished biomass in Area 1
              ProbStaying  = 0.6,   # 60% probability of staying in Area 1
              RelativeSize = 0.4)   # Area 1 is 40% of total area

# Stochastic across simulations
sp <- Spatial(UnfishedDist = c(0.2, 0.4),
              ProbStaying  = c(0.5, 0.8),
              RelativeSize = c(0.3, 0.5))
```

### Age-Varying Movement (Two-Area):

Supply `UnfishedDist` as a named `Sim × Area × Age` array. Age-specific movement is fitted independently for each age class:

```
nSim <- 1
ages <- Ages(MaxAge = 5)
nage <- nAge(ages)
```

```
# Fish progressively move to Area 2 with age
frac2 <- seq(0.1, 0.9, length.out = nage)
ud <- array(
  c(1 - frac2, frac2),
  dim      = c(nSim, 2, nage),
  dimnames = list(Sim = 1, Area = 1:2, Age = ages@Classes)
)
sp <- Spatial(UnfishedDist = ud, ProbStaying = 0.95)
```

**Multi-Area Models (Three or More Areas):**

`UnfishedDist` must be a `Sim × Area` (or higher-dimensional) array summing to 1 across areas. `FracOther` is required and specifies the relative probability of movement between each pair of areas. `CVDist` and `CVStay` control the penalty weights in the movement optimisation:

```
nArea <- 3

ud <- matrix(c(0.5, 0.2, 0.3), nrow = 1, ncol = nArea)

# Off-diagonal [i, j] = relative probability of moving from i to j
# Diagonal must be NA (staying probability comes from ProbStaying)
fo <- array(NA, dim = c(1, nArea, nArea))
fo[1, 1, ] <- c(NA, 1,   0.1)
fo[1, 2, ] <- c(1,  NA,  1  )
fo[1, 3, ] <- c(0.1, 1,  NA )

sp <- Spatial(UnfishedDist = ud,
              ProbStaying  = c(0.9, 0.2, 0.9),
              FracOther    = fo,
              RelativeSize = c(0.1, 0.4, 0.5))
```

**Supplying a Movement Matrix Directly:**

When `Movement` is supplied, the asymptotic unfished distribution is derived from it and assigned to `UnfishedDist`, overwriting any existing values. The matrix must have dimensions `Sim × FromArea × ToArea`, with rows summing to 1:

```
mov <- array(
  c(0.7, 0.3,
    0.2, 0.8),
  dim      = c(1, 2, 2),
  dimnames = list(Sim = 1, FromArea = 1:2, ToArea = 1:2)
)
sp <- Spatial(Movement = mov)
```

**Equal Density Across Areas:**

Setting `RelativeSize = "EqualDensity"` assigns `RelativeSize` equal to the mean `UnfishedDist` across ages and years, so that biomass density is constant across areas regardless of their relative size. Useful when areas are defined by biomass share rather than physical extent:

```
sp <- Spatial(UnfishedDist = 0.3,
              ProbStaying  = 0.7,
              RelativeSize = "EqualDensity")
```

**Pass-Through Access from a Stock:**

When `UnfishedDist` is a stock object, `Spatial()` returns the `Spatial` slot directly:

```
Spatial(my_stock)             # returns my_stock@Spatial
Spatial(my_stock) <- my_sp   # replaces my_stock@Spatial
```

**Slot Accessors:**

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
UnfishedDist(sp)  <- 0.4
ProbStaying(sp)   <- 0.7
RelativeSize(sp)  <- 0.5
Movement(sp)      <- my_mov
FracOther(sp)     <- my_fo
CVDist(sp)        <- 0.05
CVStay(sp)        <- 0.5
```

## Value

- `Spatial()` returns a spatial object. If `UnfishedDist` is a stock, returns `x@Spatial`.
- `Spatial<-` returns the stock `x` with the `Spatial` slot replaced and the object re-validated.
- `UnfishedDist()`, `ProbStaying()`, `RelativeSize()`, `Movement()`, `FracOther()`, `Arrangement()`, `CVDist()`, `CVStay()` return the value of the corresponding slot from `x`.
- All replacement variants return `x` with the named slot updated and the object re-validated.

## See Also

- spatial for the class definition and slot-level documentation.
- `FitMovement()` for fitting movement matrix.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.

Other spatial: `spatial-class`

## Examples

```
# ---- Default behaviour (non-spatial models) ----

## For single-area models, leave Spatial empty.
## Populate() creates a default single-area structure automatically.
stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 5))
Spatial(stk)   # empty spatial-class object
```

```
# ---- Two-area model - fixed parameters ----

## UnfishedDist: fraction of unfished biomass in Area 1
## ProbStaying:  probability of remaining in Area 1
## RelativeSize: fraction of total area that is Area 1
## Area 2 values are derived automatically.
sp <- Spatial(UnfishedDist = 0.3,
              ProbStaying  = 0.6,
              RelativeSize = 0.4)
sp <- Populate(sp, Ages = Ages(MaxAge = 5), nSim = 1)
UnfishedDist(sp)   # asymptotic distribution: Sim x Area x Age x Year
Movement(sp)       # fitted movement matrix: Sim x FromArea x ToArea x Age x Year

# ---- Two-area model - stochastic across simulations ----

## Length-2 vectors are sampled from Uniform(lower, upper) once per simulation.
sp_stoch <- Spatial(UnfishedDist = c(0.2, 0.4),
                    ProbStaying  = c(0.5, 0.8),
                    RelativeSize = c(0.3, 0.5))
sp_stoch <- Populate(sp_stoch, Ages = Ages(MaxAge = 5), nSim = 48)
UnfishedDist(sp_stoch)   # nSim x 2 x Age x Year

# ---- Three-area model - fixed parameters ----

## FracOther is required for 3+ areas.
## Diagonal elements must be NA; off-diagonal [i,j] is the relative
## probability of moving from area i to area j (normalised internally).
nArea <- 3
ud <- matrix(c(0.5, 0.2, 0.3), nrow = 1, ncol = nArea)

fo <- array(NA, dim = c(1, nArea, nArea))
fo[1, 1, ] <- c(NA,  1,   0.1)   # from Area 1: 10x more likely to go to Area 2 than 3
fo[1, 2, ] <- c(1,   NA,  1  )   # from Area 2: equal probability to Area 1 and 3
fo[1, 3, ] <- c(0.1, 1,   NA )   # from Area 3: 10x more likely to go to Area 2 than 1

sp3 <- Spatial(UnfishedDist = ud,
               ProbStaying  = c(0.9, 0.2, 0.9),
               FracOther    = fo,
               RelativeSize = c(0.1, 0.4, 0.5))
sp3 <- Populate(sp3, Ages = Ages(MaxAge = 5), nSim = 1)
UnfishedDist(sp3)[1, , , 1]
Movement(sp3)[1, , , 1, 1]

# ---- Age-varying movement (3-area) ----

## Supply a named Sim x Area x Age array for UnfishedDist; movement is
## fitted independently for each age class.
nage <- nAge(Ages(MaxAge = 5))
ud3 <- array(NA, dim = c(1, nArea, nage))
ud3[, , 1] <- c(0.95, 0.045, 0.005)
ud3[, , 2] <- c(0.75, 0.20,  0.05 )
ud3[, , 3] <- c(0.50, 0.40,  0.10 )
ud3[, , 4] <- c(0.30, 0.50,  0.20 )
```

```
ud3[, , 5] <- c(0.10, 0.50,  0.40 )
ud3[, , 6] <- c(0.01, 0.20,  0.79 )

sp3_age <- Spatial(UnfishedDist = ud3, ProbStaying = 0.05, FracOther = fo)
sp3_age <- Populate(sp3_age, Ages = Ages(MaxAge = 5), nSim = 1)

## Asymptotic distribution and movement at youngest and oldest age class
UnfishedDist(sp3_age)[1, , , 1]
Movement(sp3_age)[1, , , 1,    1]   # youngest
Movement(sp3_age)[1, , , nage, 1]   # oldest

# ---- Supplying a movement matrix directly ----

## When Movement is supplied, UnfishedDist is derived from it.
## Each row (FromArea) must sum to 1 across ToAreas.
mov <- array(NA, dim = c(1, 2, 2),
             dimnames = list(Sim = 1, FromArea = 1:2, ToArea = 1:2))
mov[1, 1, ] <- c(0.7, 0.3)   # from Area 1: 70% stay, 30% move to Area 2
mov[1, 2, ] <- c(0.2, 0.8)   # from Area 2: 20% move to Area 1, 80% stay
sp_mov <- Spatial(Movement = mov)
sp_mov <- Populate(sp_mov, Ages = Ages(MaxAge = 5), nSim = 1)
UnfishedDist(sp_mov)   # derived from supplied Movement matrix

# ---- Equal density across areas ----

## RelativeSize = "EqualDensity" sets RelativeSize equal to mean
## UnfishedDist, producing constant density across areas.
sp_ed <- Spatial(UnfishedDist = 0.3,
                 ProbStaying  = 0.7,
                 RelativeSize = "EqualDensity")
sp_ed <- Populate(sp_ed, Ages = Ages(MaxAge = 5), nSim = 1)
RelativeSize(sp_ed)   # equals mean UnfishedDist

# ---- Slot accessors ----

## Read slots
CVDist(sp)
CVStay(sp)

## Replace slots
UnfishedDist(sp) <- 0.4
ProbStaying(sp)  <- 0.7
CVDist(sp)       <- 0.05

# ---- Attaching to a Stock ----

Spatial(stk) <- Spatial(UnfishedDist = 0.3,
                        ProbStaying  = 0.6,
                        RelativeSize = 0.4)
Spatial(stk)
```

`spatial-class` *The* `spatial` *S4 Class*

## Description

Defines the spatial structure and movement dynamics for a stock object. Objects are typically created via `Spatial()`, which documents all parameters and validates inputs. This object is optional — leave it empty for non-spatial (single-area) models; see *Default Behaviour* in `Spatial()`.

## Details

Direct construction via `methods::new()` is not recommended; use `Spatial()` instead, which validates inputs and handles dimension expansion.

For non-spatial (single-area) models, leave the `Spatial` slot of stock empty. `Populate()` creates a default single-area structure automatically.

The movement matrix is fitted by numerical optimisation to simultaneously reproduce the target `UnfishedDist` and `ProbStaying`. For multi-area models, `CVDist` and `CVStay` control the relative weight given to each target in the objective function.

## Slots

`UnfishedDist` `numeric`, `array`, or `NULL`. Relative unfished biomass distribution across areas. After `Populate()`, a named array with dimensions `Sim × Area × Age × Year`. For two-area models, specifies the fraction in Area 1; the Area 2 fraction is derived as `1 - UnfishedDist`. For multi-area models, values must sum to 1 across areas. See `Spatial()` for accepted input formats.

`ProbStaying` `numeric` or `array`. Probability of remaining in an area between time steps. After `Populate()`, a named array with dimensions `Sim × Area × Age × Year`. For two-area models, specifies Area 1 only; the Area 2 value is solved internally. See `Spatial()` for accepted input formats.

`RelativeSize` `numeric`, `array`, or `character(1)`. Relative size of each area, used to compute biomass density. After `Populate()`, a named `Sim × Area` array. The special value `"EqualDensity"` sets `RelativeSize` equal to the mean `UnfishedDist` across ages and years, producing constant density across areas. When `NULL`, all areas are assumed equal in size (`1/nArea` each).

`Movement` `array` or `NULL`. Movement probability matrix with named dimensions `Sim × FromArea × ToArea × Age`. Row `[i, ]` gives the probabilities of moving from area `i` to each other area; rows sum to 1. Populated automatically during `Populate()` from `UnfishedDist`, `ProbStaying`, and optionally `FracOther`. May also be supplied directly, in which case the asymptotic unfished distribution is derived from `Movement` and assigned to `UnfishedDist`.

`FracOther` `array` or `NULL`. Required for models with more than two areas. Named array with dimensions `Sim × FromArea × ToArea × Age × Year`. Diagonal elements must be `NA` (probability of staying is taken from `ProbStaying`). Off-diagonal element `[i, j]` gives the relative probability of moving from area `i` to area `j`. Used as a penalty-weight matrix during movement optimisation.

`Arrangement array` or `NULL`. Spatial layout of areas for plotting purposes. Reserved for future use; currently stored but not applied.

`CVDist numeric(1)`. Standard deviation of the log-scale penalty on the difference between the optimised asymptotic distribution and the target `UnfishedDist` during multi-area movement optimisation. Larger values allow the fitted movement matrix to deviate more from the target distribution. Has no effect for two-area models. Default `0.1`.

`CVStay numeric(1)`. Standard deviation of the logit-scale penalty on the difference between the optimised diagonal (staying probability) and the target `ProbStaying` during multi-area movement optimisation. Larger values allow more deviation from the target staying probability. Has no effect for two-area models. Default `1`.

`Misc list`. Used internally.

### See Also

- `Spatial()` for the constructor and accessor functions.
- `Populate()` for array population and movement matrix fitting.
- `Stock()` for the enclosing stock constructor.

Other spatial: `Spatial()`

---

## SPControl *Numerical Settings for the Internal Surplus Production Model*

---

### Description

Controls for `FitSP()` and `SurplusProduction()` with `Model = 'internal'`.

### Usage

```
SPControl(
  nSubStep = 4,
  nItF = 5,
  Fmax = 3,
  FPenalty = 1000,
  FoxTol = 0.001,
  CatchScale = c("mean", "none"),
  WarmStart = TRUE,
  nGrid = 12,
  GridFMSY = c(0.02, 1),
  GridMSY = c(0.2, 5),
  nRestart = 2,
  MinSD = 0.05,
  GradTol = 0.001,
  nNewton = 10,
  BoundTol = 0.01,
  Bounds = list(FMSY = c(0.005, 3), MSY = c(0.01, 100), Depletion = c(0.05, 1.5), Shape =
```

```
    c(0.2, 10)),
  nlminb = list(iter.max = 500, eval.max = 1000),
  Hessian = c("auto", "always", "never")
)
```

## Arguments

| | |
|---|---|
| `nSubStep` | Positive integer. Sub-steps per year used to integrate the biomass dynamics. Default `4`. |
| `nItF` | Positive integer. Maximum Newton iterations used to solve the annual fishing mortality that produces the observed catch. Default `5`. |
| `Fmax` | Positive number. Maximum annual fishing mortality. Default `3`. |
| `FPenalty` | Non-negative number. Weight of the penalty on the squared log difference between observed and predicted catch, added to the negative log-likelihood. The difference is non-zero when the catch cannot be taken with `F <= Fmax`. Default `1000`. |
| `FoxTol` | Positive number. `Shape` within `FoxTol` of `1` uses the Fox model. Default `1e-3`. |
| `CatchScale` | Character. `'mean'` (default) rescales catch to a mean of `1` during fitting; `'none'` fits on the original scale. |
| `WarmStart` | Logical. In `SurplusProduction()`, start each fit from the estimates of the previous management cycle? Default `TRUE`. |
| `nGrid` | Positive integer. Number of `FMSY` and `MSY` values in the grid of starting values (`nGrid x nGrid` cells), used when there is no `Start` (or warm start) or the fit from it does not converge. The first fit starts from the cell with the lowest negative log-likelihood. Default `12`. |
| `GridFMSY` | Numeric vector, length 2. Range of `FMSY` in the grid. Default `c(0.02, 1)`. |
| `GridMSY` | Numeric vector, length 2. Range of `MSY` in the grid, as multiples of the mean catch. Default `c(0.2, 5)`. |
| `nRestart` | Non-negative integer. Maximum number of additional fits, started from the next-best grid cells, when the fit from the best cell does not converge. Default `2`. |
| `MinSD` | Positive number. Lower bound on an estimated index observation standard deviation (`IndexSD = 'estimate'`). Default `0.05`. |
| `GradTol` | Positive number. A fit is converged when the Hessian of the negative log-likelihood is positive definite and, for each estimated (log-scale) parameter, the Newton step (the inverse Hessian times the gradient, the estimated distance to the optimum) is below `GradTol`. The gradient and Hessian are calculated exactly. Default `1e-3`. |
| `nNewton` | Non-negative integer. Maximum number of Newton steps, with the exact gradient and Hessian, taken after `stats::nlminb()` stops, used when the estimated distance to the optimum exceeds `GradTol`. Default `10`. |
| `BoundTol` | Non-negative number. A fit with a log-scale parameter within `BoundTol` of a bound is not converged. Default `0.01`. |

| | |
|---|---|
| Bounds | Named list of c(lower, upper) bounds on the estimated FMSY, MSY (as multiples of the mean catch), Depletion, and Shape. Elements not given keep their defaults: FMSY = c(0.005, 3), MSY = c(0.01, 100), Depletion = c(0.05, 1.5), Shape = c(0.2, 10). |
| nlminb | List of control settings passed to stats::nlminb(). |
| Hessian | Character. When to calculate the Hessian used for the standard errors: 'auto' (default; when Uncertainty = TRUE in FitSP(), or when SurplusProduction() uses FractileB, FractileF, or Diagnostics = 'full'), 'always', or 'never'. |

### Value

A list of class spcontrol.

### See Also

FitSP(), SurplusProduction(), SpictControl()

---

## SPEstimates *Extract Surplus Production Model Estimates from an MSE*

---

### Description

Collects the estimates made by SurplusProduction() in every management cycle of a projection, alongside the operating model values for comparison.

### Usage

```
SPEstimates(MSE, MPs = NULL, Type = c("Summary", "Indices"))
```

### Arguments

| | |
|---|---|
| MSE | An mse object. |
| MPs | Character vector of MP names. NULL (default) uses every MP that stored SurplusProduction() estimates. |
| Type | Character. 'Summary' (default) or 'Indices'. |

### Value

For Type = 'Indices', a data.frame with one row per MP, simulation, stock (complex), management cycle (AdviceYear), and index, with the columns of Indices described in SurplusProduction() (Details, Diagnostics).

For Type = 'Summary', a data.frame with one row per MP, simulation, stock (complex), and management cycle: AdviceYear, LastDataYear, Converged, Fallback, the estimated B_BMSY (biomass at the start of the year after the last data year relative to BMSY) and F_FMSY (in the last data year), BAdvice_BMSY, MSY, FMSY, FAdvice, Advice (the TAC or relative effort before allocation), Message, and the operating model values OM_B_BMSY and OM_F_FMSY for the same years (NA where not available, e.g. for a complex of several stocks).

Estimates are only available for MPs run with Diagnostics = 'min' or 'full'.

### See Also

SurplusProduction(), Log(), B_BMSY(), F_FMSY()

### Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
MSE  <- Project(Hist, MPs = list(SP = SurplusProduction))
Est  <- SPEstimates(MSE)

# fraction of management cycles with advice set by `OnFail`, by MP
aggregate(Fallback ~ MP, data = Est, FUN = mean)

# reasons the fit was not used
table(Est$Message[Est$Fallback])

# estimated vs operating model stock status
plot(Est$OM_B_BMSY, Est$B_BMSY, xlab = 'OM B/BMSY', ylab = 'Estimated B/BMSY')
abline(0, 1)

## End(Not run)
```

---

## SpictControl *Settings for the SPiCT Surplus Production Model*

---

### Description

Controls for FitSP() and SurplusProduction() with Model = 'spict', which fit the stochastic surplus production model in continuous time (SPiCT) with the spict package. spict is not on CRAN; install it with pak::pkg_install('DTUAqua/spict/spict').

### Usage

```
SpictControl(
  dteuler = 1,
  SeasonalCatch = FALSE,
  Robust = FALSE,
  Priors = list(),
  FixedDepletionCV = 0.01,
  stabilise = TRUE,
  optimiser = c("nlminb", "optim"),
  optimiser.control = list(iter.max = 500, eval.max = 1000),
  GradTol = 0.001,
  RestartK = c(10, 2),
  SDReport = c("auto", "always", "never"),
  getReportCovariance = FALSE,
  WarmStart = TRUE,
```

```
  Quiet = TRUE,
  Inp = list()
)
```

## Arguments

| | |
|---|---|
| `dteuler` | Positive number. Euler time step (years) of the model. Default `1`. With sub-annual catch (`SeasonalCatch = TRUE`), the time step is at most the catch interval. |
| `SeasonalCatch` | Logical. Fit to the seasonal catch of seasonal `Data` (`Data@Seasons > 1`) rather than the annual total? Default `FALSE`. |
| `Robust` | Logical. Use robust (mixture) observation error for the catch and indices (`robflagc`, `robflagi`)? Default `FALSE`. |
| `Priors` | Named list of `spict` priors (e.g. `logalpha`, `logbeta`), each `c(mean, sd, use)` on the log scale. These are applied after, and replace, any priors set from the `FitSP()` arguments. |
| `FixedDepletionCV` | Positive number. CV of the prior on the initial depletion when `EstDepletion = FALSE`. Default `0.01`. |
| `stabilise` | Logical. Use the `spict` vague stabilising priors? Default `TRUE`. |
| `optimiser` | Character. `'nlminb'` (default) or `'optim'`. |
| `optimiser.control` | List of optimiser controls. Default `list(iter.max = 500, eval.max = 1000)`. |
| `GradTol` | Positive number. Convergence tolerance on the Newton step; see Details. Default `1e-3`. |
| `RestartK` | Numeric vector. Starting values of `K`, as multiples of the maximum catch, used for additional fits when a fit does not converge. `NULL` for no additional fits. Default `c(10, 2)`. |
| `SDReport` | Character. When to run `TMB::sdreport()`, which is needed for the standard errors: `'auto'` (default; when `Uncertainty = TRUE` in `FitSP()`, or when `SurplusProduction()` uses `FractileB`, `FractileF`, or `Diagnostics = 'full'`), `'always'`, or `'never'`. |
| `getReportCovariance` | Logical. Passed to `TMB::sdreport()`. Default `FALSE`. |
| `WarmStart` | Logical. In `SurplusProduction()`, start each fit from the estimates of the previous management cycle? Default `TRUE`. |
| `Quiet` | Logical. Suppress `spict` messages and warnings, recording warnings in the fit's `Log`? Default `TRUE`. |
| `Inp` | Named list of any other `spict` `inp` elements. Applied last, so these replace any values set by `FitSP()` or the other arguments. |

### Details

The model shape, initial depletion, index weighting, and priors are set with the `FitSP()` arguments (`Shape`, `EstShape`, `Depletion`, `EstDepletion`, `IndexSD`, `IndexWeight`, `Priors`), which are converted to the equivalent `spict` inputs:

- `EstShape = FALSE` fixes `n` at `Shape`; `EstShape = TRUE` estimates `n` with a lognormal prior, `Priors$Shape` if given, otherwise with median `Shape` and CV `ShapeCV`.
- `Depletion` is the starting value of the initial depletion (`logbkfrac`). With `EstDepletion = FALSE` it is held close to `Depletion` by a lognormal prior with CV `FixedDepletionCV`; with `EstDepletion = TRUE` it has the prior `Priors$Depletion`, if given.
- `IndexWeight` and, with `IndexSD = 'data'`, the relative index `CV`s set the `stdevfacI` scaling of each index observation's standard deviation. The standard deviation of each index is still estimated.
- `Priors$FMSY` sets a prior on the intrinsic growth rate, `FMSY * n` (with `n = Shape`), and `Priors$MSY` a prior on `m` (MSY).
- `Start` (or the warm start in `SurplusProduction()`) sets the starting values of `m` and `K`.
- Catch observations cannot be zero, so a year (or season) with zero catch is combined with the preceding catch observation (or the following one, before the first positive catch) into one observation of the total catch over the longer interval.

A fit is converged when the Hessian of the objective function is positive definite and, for each estimated parameter, the Newton step (the inverse Hessian times the gradient) is below `GradTol`. If the fit does not converge, it is repeated from the default `spict` starting values (when the first fit used `Start`) and then with the starting value of `K` at each multiple of the maximum catch in `RestartK`, and the converged fit with the lowest objective function value is used.

### Value

A list of class `spictcontrol`.

### References

Pedersen, M. W. and Berg, C. W. 2017. A stochastic surplus production model in continuous time. Fish and Fisheries 18: 226-243.

### See Also

`FitSP()`, `SurplusProduction()`, `SPControl()`

## Splot *Standard MSE projection plot*

### Description

Plots projections of F/FMSY, SB/SBMSY and Yield

### Usage

```
Splot(MSEobj, MPs = 5, p = c(0.05, 0.25, 0.5, 0.75, 0.95))
```

### Arguments

| | |
|---|---|
| MSEobj | Object of class 'MSE' from runMSE() or Project() |
| MPs | Either a positive integer (the first MPs number of MPs to plot), a character vector (the names of the MPs to plot), or an integer vector (the index of the MPs to plot) |
| p | Vector of quantiles five positions long. Defaults to c(0.05,0.25,0.5,0.75,0.95) so the 90% and 50% intervals with the median plotted in white. |

### Author(s)

T. Carruthers

## SRR *SRR Constructor and Accessors*

### Description

Construct a srr object defining the stock-recruitment relationship and recruitment variability for a stock, or access and replace the SRR slot of a stock and its individual slots. An SRR object is required for all stock objects.

### Usage

```
SRR(
  Pars = list(h = NA),
  Model = "BevertonHolt",
  R0 = NULL,
  SD = NULL,
  AC = NULL,
  SPFrom = NULL,
  TruncSD = 3,
  RecDevInit = NULL,
  RecDevHist = NULL,
```

```
  RecDevProj = NULL,
  SpawnTimeFrac = 0,
  SpawnLag = NULL,
  RelRecFun = NULL,
  Units = 1,
  Misc = list()
)

R0(x)

R0(x) <- value

SPFrom(x)

SPFrom(x) <- value

RecDevInit(x)

RecDevInit(x) <- value

RecDevHist(x)

RecDevHist(x) <- value

RecDevProj(x)

RecDevProj(x) <- value

SpawnTimeFrac(x)

SpawnTimeFrac(x) <- value

SpawnLag(x)

SpawnLag(x) <- value

RelRecFun(x)

RelRecFun(x) <- value

SRR(x) <- value
```

## Arguments

| | |
|---|---|
| `Pars` | `list`. Named list containing the steepness parameter for the chosen SRR model. `R0` is never placed in `Pars` — it has its own dedicated slot. The required name depends on `Model`:<br>• `BevertonHolt`: `list(h = ...)` where `0.2 < h < 1`. |

- Ricker: list(hR = ...).
- HockeyStick: list(Shinge = ...) where 0 < Shinge <= 1. BevertonHolt and Ricker also accept list(alpha = ..., beta = ...) (see SRRSteepness()), in which case R0 must be NULL. During Populate() these are converted to steepness and R0 using the first-year unfished spawning production per recruit, and the supplied values are kept in Misc$AlphaBeta. alpha is recruits per unit of spawning production and beta is in units of 1/spawning production, on the scale set by Units. Not available for seasonal stocks, stocks in a Herm() complex, or stocks that recruit from another stock's spawning production (SPFrom). When Pars is a non-list S4 object with an SRR slot (e.g., a stock), SRR() acts as a pass-through accessor and returns that slot. See Specifying Biological and Fleet Schedules for accepted input formats (scalar, bounds vector, nSim-length vector). Default list(h = NA).

Model — character(1) or function. SRR model identifier. Must match one of the models listed in SRRModels(), or be a custom R function. Default "BevertonHolt".

R0 — numeric, array, or NULL. Unfished equilibrium recruitment. May be:

- A scalar: same value for all simulations and years.
- A length-2 bounds vector: sampled from Uniform(lower, upper) once per simulation.
- A length-nSim vector: one value per simulation, constant over years.
- A Sim x Year array: deliberately time-varying, e.g. to model a regime shift in carrying capacity. See srr for how this differs from a constant R0. Interpreted in units of Units. Must be NULL when Pars contains alpha and beta. Default NULL.

SD — numeric or NULL. Log-space standard deviation of recruitment deviations. Follows the same length conventions as R0. Currently fixed across years (time-varying SD is not yet supported). When NULL, recruitment is deterministic (no process error). Default NULL.

AC — numeric or NULL. Lag-1 autocorrelation of log-space recruitment deviations. Follows the same length conventions as R0. Currently fixed across years. Defaults to 0 (no autocorrelation) when NULL. Default NULL.

SPFrom — character(1), numeric(1), or a named numeric vector. Stock(s) whose spawning production drives this stock's recruitment. Specify a single source by stock name (character) or 1-based integer index (numeric), or a weighted blend of multiple sources with a named numeric vector, e.g. c(Stock1 = 0.6, Stock2 = 0.4). Defaults to self (the stock recruits from its own spawning production). Relevant only in multi-stock operating models. See srr for details and constraints on the weighted form. Default NULL.

TruncSD — numeric(1). Number of standard deviations at which the log-normal recruitment deviation distribution is truncated. Default 2.

RecDevInit — matrix or NULL. Pre-specified recruitment deviations (Sim × Age) for initialising the historical age structure, covering all age classes except the minimum age (whose deviation is the first column of RecDevHist). If

NULL (default), generated internally from SD and AC during Populate(). The trailing dimension must have length length(Ages@Classes) - 1; PopulateSRR() aborts otherwise.

RecDevHist — matrix or NULL. Pre-specified recruitment deviations (Sim × nHistTS) for historical time steps. The first column corresponds to the minimum age class in the initial year. If NULL (default), generated internally from SD and AC. The trailing dimension must match the number of historical years; PopulateSRR() aborts otherwise.

RecDevProj — matrix or NULL. Pre-specified recruitment deviations (Sim × nProjTS) for projection time steps. If NULL (default), generated internally from SD and AC. The trailing dimension must match the number of projection years; PopulateSRR() aborts otherwise.

SpawnTimeFrac — numeric(1). Fraction of the time step elapsed before spawning occurs. Determines how much mortality (exp(-Z × SpawnTimeFrac)) is applied to the population before spawning numbers are counted:

- 0 (default): spawning at the start of the step; no mortality applied before spawning.
- 0.5: spawning mid-step; half of annual Z applied before spawning.
- 1: spawning at the end of the step; full within-step Z applied before spawning. Default 0.

SpawnLag — numeric(1) or NULL. Number of timesteps between the spawning production evaluation and recruitment. When NULL (default), the lag is derived automatically from min(Ages@Classes) * Seasons. Set explicitly when the spawn season differs from what min(Ages@Classes) implies (e.g. SS3 models where Spawn_month and birthseas are decoupled). Default NULL.

RelRecFun — function, character(1), or NULL. Relative recruitment function giving equilibrium recruitment relative to R0 as a function of spawning-per-recruit (SPR). For built-in models, set automatically during Populate() (e.g., "BevertonHolt_RelRec" when Model = "BevertonHolt"). For custom SRR models, must be supplied with signature function(Pars, SPR). Default NULL.

Units — numeric(1). Scaling factor for recruitment. 1 (default) means R0 is in absolute numbers of fish; 1000 means R0 is in thousands of fish. Does not affect internal calculations — used only to set the interpretation of numbers in hist output. Default 1.

Misc — list. Used internally. Default list().

x — A srr object for slot accessors, or a stock object for SRR<-.

value — For SRR<-: a srr object. For slot replacement functions: the replacement value for the corresponding slot.

## Details

### Specifying the SRR:

The SRR is defined by three components:

1. **The deterministic curve**: Model and Pars (steepness only; R0 goes in its own slot). See SRRModels() for available models:

```
# Beverton-Holt with fixed steepness
srr <- SRR(Pars = list(h = 0.7), R0 = 1000)

# Beverton-Holt with stochastic steepness across simulations
srr <- SRR(Pars = list(h = c(0.6, 0.9)), R0 = 1000)

# Ricker
srr <- SRR(Pars = list(hR = 0.7), Model = "Ricker", R0 = 1000)

# Hockey-stick
srr <- SRR(Pars = list(Shinge = 0.3), Model = "HockeyStick", R0 = 1000)
```

1. **Process error**: `SD` and `AC` control the magnitude and autocorrelation of log-space recruitment deviations. Both are currently fixed across years:

```
# Stochastic recruitment: SD = 0.4, no autocorrelation
srr <- SRR(Pars = list(h = 0.7), R0 = 1000, SD = 0.4)

# Stochastic recruitment: SD drawn from Uniform(0.3, 0.6), AC = 0.4
srr <- SRR(Pars = list(h = 0.7), R0 = 1000,
           SD = c(0.3, 0.6), AC = 0.4)
```

1. **Recruitment deviations** (`RecDevInit`, `RecDevHist`, `RecDevProj`): generated automatically from `SD` and `AC` during `Populate()` when `NULL`. Supply directly to condition the operating model on observed recruitment indices or to reproduce a specific stochastic trajectory.

### Custom SRR Models:

A custom SRR function may be passed to `Model`. It must accept `S`, `S0`, `R0`, and any named parameters in `Pars` as arguments. A matching `RelRecFun` with signature `function(Pars, SPR)` must also be supplied:

```
my_srr <- function(S, S0, R0, h) {
  # custom Beverton-Holt variant
  4 * h * R0 * S / (S0 * (1 - h) + S * (5 * h - 1))
}
my_rrf <- function(Pars, SPR) {
  h <- Pars$h
  CR <- 4 * h / (1 - h)
  pmax((CR * SPR - 1) / ((CR - 1) * SPR), 0)
}
srr <- SRR(Pars = list(h = 0.7), Model = my_srr,
           R0 = 1000, RelRecFun = my_rrf)
```

### Spawning Production Source (`SPFrom`):

In multi-stock models, one stock's recruitment may be driven by another stock's spawning production. Specify by stock name or 1-based index:

```
# Stock 2 recruits based on stock 1's spawning production
srr2 <- SRR(Pars = list(h = 0.7), R0 = 500, SPFrom = 1)
# equivalently by name:
```

```
srr2 <- SRR(Pars = list(h = 0.7), R0 = 500, SPFrom = "Stock1")

# Stock 3 recruits from a 60/40 weighted blend of stocks 1 and 2
srr3 <- SRR(Pars = list(h = 0.7), R0 = 500,
            SPFrom = c(Stock1 = 0.6, Stock2 = 0.4))
```

**Pass-Through Access from a Stock:**
When `Pars` is a non-list S4 object with an `SRR` slot, `SRR()` returns that slot directly:

```
SRR(my_stock)           # returns my_stock@SRR
SRR(my_stock) <- my_srr # replaces my_stock@SRR
```

**Slot Accessors:**
Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Pars(srr)          <- list(h = 0.8)
Model(srr)         <- "Ricker"
R0(srr)            <- 5000
SD(srr)            <- 0.4
AC(srr)            <- 0.3
TruncSD(srr)       <- 2
SpawnTimeFrac(srr) <- 0.5
SPFrom(srr)        <- 1
RelRecFun(srr)     <- "Ricker_RelRec"
RecDevHist(srr)    <- my_hist_matrix
RecDevProj(srr)    <- my_proj_matrix
RecDevInit(srr)    <- my_init_matrix
```

## Value

- `SRR()` returns a srr object. If `Pars` is a non-list S4 object with an `SRR` slot, returns that slot.
- `SRR<-` returns the stock `x` with the `SRR` slot replaced and the object re-validated.
- `R0()`, `SPFrom()`, `RecDevInit()`, `RecDevHist()`, `RecDevProj()`, `SpawnTimeFrac()`, `RelRecFun()` return the value of the corresponding slot from `x`.
- All replacement variants return `x` with the named slot updated and the object re-validated.

## See Also

- srr for the class definition and slot-level documentation.
- `SRRModels()` for available stock-recruitment models and their required parameters.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.
- `Fecundity()` for the spawning production used as input to the SRR.
- Specifying Biological and Fleet Schedules for accepted `Pars`, `R0`, `SD`, and `AC` input formats.

Other srr: `srr-class`

## Examples

```
# See SRRModels() for all available stock-recruitment models and their
# required parameters. Note: R0 always goes in its own slot, not in Pars.
SRRModels()

# ---- Beverton-Holt (default) ----

## Fixed steepness, deterministic recruitment
srr <- SRR(Pars = list(h = 0.7), R0 = 1000)

## Stochastic steepness - h drawn from Uniform(0.6, 0.9) once per simulation
srr <- SRR(Pars = list(h = c(0.6, 0.9)), R0 = 1000)

## Stochastic R0 across simulations
srr <- SRR(Pars = list(h = 0.7), R0 = c(500, 2000))

## Add recruitment process error: SD = 0.4, no autocorrelation
srr <- SRR(Pars = list(h = 0.7), R0 = 1000, SD = 0.4)

## SD drawn from Uniform(0.3, 0.6), with lag-1 autocorrelation
srr <- SRR(Pars = list(h = 0.7), R0 = 1000,
           SD = c(0.3, 0.6), AC = 0.4)

# ---- Ricker ----

srr_r <- SRR(Pars  = list(hR = 0.7),
             Model = "Ricker",
             R0    = 1000,
             SD    = 0.4)

# ---- Hockey-stick ----

## Shinge is the hinge point relative to S0 (0 < Shinge <= 1)
srr_hs <- SRR(Pars  = list(Shinge = 0.3),
              Model = "HockeyStick",
              R0    = 1000,
              SD    = 0.4)

# ---- Spawning timing ----

## SpawnTimeFrac = 0 (default): spawning at start of step, no pre-spawn Z
## SpawnTimeFrac = 0.5:         mid-step spawning, exp(-0.5 * Z) applied first
## SpawnTimeFrac = 1:           end-of-step spawning, full exp(-Z) applied first
srr_mid <- SRR(Pars = list(h = 0.7), R0 = 1000, SpawnTimeFrac = 0.5)

# ---- Pre-specified recruitment deviations ----

## Supply RecDevHist directly to condition on observed recruitment.
## Dimensions: nSim x nHistTS. Values are multiplicative deviations
## in log-space (mean-zero lognormal deviates).
nSim    <- 48
nHistTS <- 50
```

```
hist_devs <- matrix(rnorm(nSim * nHistTS, 0, 0.4),
                    nrow = nSim, ncol = nHistTS)
dimnames(hist_devs) <- list(
  Sim  = seq_len(nSim),
  Year = seq(1970, by = 1, length.out = nHistTS)
)
srr_cond <- SRR(Pars       = list(h = 0.7),
                R0         = 1000,
                SD         = 0.4,
                RecDevHist = hist_devs)

# ---- Multi-stock: SPFrom ----

## Stock 2 recruits based on stock 1's spawning production.
## Specify by 1-based index or by stock name.
srr_s2 <- SRR(Pars   = list(h = 0.7),
              R0     = 500,
              SPFrom = 1)

# ---- Custom SRR model ----

## A custom model must accept S, S0, R0, and any named Pars.
## A matching RelRecFun with signature function(Pars, SPR) must also be given.
my_srr <- function(S, S0, R0, h) {
  4 * h * R0 * S / (S0 * (1 - h) + S * (5 * h - 1))
}
my_rrf <- function(Pars, SPR) {
  h  <- Pars$h
  CR <- 4 * h / (1 - h)
  pmax((CR * SPR - 1) / ((CR - 1) * SPR), 0)
}
srr_custom <- SRR(Pars       = list(h = 0.7),
                  Model      = my_srr,
                  R0         = 1000,
                  SD         = 0.4,
                  RelRecFun  = my_rrf)

# ---- Slot accessors ----

srr <- SRR(Pars = list(h = 0.7), R0 = 1000, SD = 0.4, AC = 0.3)

## Read slots
Pars(srr)
R0(srr)
SD(srr)
AC(srr)
SpawnTimeFrac(srr)
RelRecFun(srr)      # NULL until Populate() is called for built-in models

## Replace slots
R0(srr)            <- 2000
SD(srr)            <- 0.5
SpawnTimeFrac(srr) <- 0.5
```

```
# ---- Attaching to a Stock ----

stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
SRR(stk) <- SRR(Pars = list(h = 0.7), R0 = 1000, SD = 0.4)
SRR(stk)

# ---- Populate ----

## The final populated object (typically done internally).
## RecDevHist, RecDevProj, and RecDevInit are generated from SD and AC.
pop_srr <- Populate(SRR(stk), Ages = Ages(MaxAge = 20))
pop_srr
```

---

`srr-class` *The* `srr` *S4 Class*

---

## Description

Defines the stock-recruitment relationship (SRR) and recruitment variability for a stock object. Objects are typically created via `SRR()`, which documents all parameters and validates inputs. An `srr` object is required for all stock objects.

## Details

Direct construction via `methods::new()` is not recommended; use `SRR()` instead, which validates inputs and sets sensible defaults.

Recruitment deviations are generated in log-space with standard deviation `SD` and lag-1 autocorrelation `AC`, truncated at `TruncSD` standard deviations. `SD` and `AC` are currently fixed across years; time-varying process variance is not yet supported.

`RecDevInit`, `RecDevHist`, and `RecDevProj` may be supplied directly to override the internally generated deviations — for example, to condition the operating model on observed recruitment indices.

## Slots

`Pars list`. Named list containing the steepness parameter for the chosen SRR model. For `BevertonHolt`: `list(h = ...)`. For `Ricker`: `list(hR = ...)`. For `HockeyStick`: `list(Shinge = ...)`. `R0` is never placed in `Pars`; it has its own dedicated slot. `BevertonHolt` and `Ricker` also accept `list(alpha = ..., beta = ...)` with `R0 = NULL`; `Populate()` converts these to steepness and `R0` and stores the supplied values in `Misc$AlphaBeta` (see `SRRSteepness()`). See `SRRModels()` and Specifying Biological and Fleet Schedules for accepted input formats.

`Model function` or `character(1)`. SRR model identifier. Defaults to `"BevertonHolt"`. See `SRRModels()` for available models.

`R0` `array` or `numeric`. Unfished equilibrium recruitment. May be a scalar (same across all simulations), a length-2 bounds vector (sampled from `Uniform(lower, upper)` once per simulation), a length-`nSim` vector, or a `Sim x Year` array for a deliberately time-varying R0 (e.g. to model a regime shift in carrying capacity). Interpreted in units of `Units` (e.g., `Units = 1000` means R0 is in thousands of fish). A constant R0 only sets the absolute scale of numbers in hist output and does not otherwise affect population dynamics. A time-varying R0 does affect dynamics: the spawning production per recruit implied by steepness is fixed at its initial-year value (so time-varying life-history parameters like `M` don't cause it to drift), but `R0` itself is read for the current year, so a change in `R0` immediately rescales the stock-recruitment curve from that year on.

`SD` `array` or `numeric`. Log-space standard deviation of recruitment deviations. May be a scalar, length-2 bounds vector, or length-`nSim` vector. Fixed across years (currently not time-varying). When `NULL`, recruitment is deterministic.

`AC` `array` or `numeric`. Lag-1 autocorrelation of log-space recruitment deviations. Follows the same length conventions as `SD`. Fixed across years (currently not time-varying). Defaults to `0` (no autocorrelation) when `NULL`.

`SPFrom` `character(1)`, `numeric(1)`, or a named `numeric` vector. Identifies which stock's spawning production drives this stock's recruitment. Defaults to self (stock recruits from its own spawning production). For multi-stock models, may be:

- An unnamed stock name (`character(1)`) or 1-based integer index (`numeric(1)`): single source, 100% of this stock's spawning production is replaced by the named stock's.
- A **named** `numeric` vector, e.g. `c(Stock1 = 0.6, Stock2 = 0.4)`: weighted blend of spawning production from multiple stocks. Names may be stock names or numeric-string indices. Weights need not sum to `1` (a warning is logged to `Log` if they don't); negative or non-finite weights are not allowed. Every referenced source stock must itself be self-recruiting.

`TruncSD` `numeric(1)`. Number of standard deviations at which the log-normal recruitment deviation distribution is truncated. Default `2`.

`RecDevInit` `matrix`. Recruitment deviations (`Sim × Age`) used to initialise the historical age structure. Covers all age classes except the minimum age, whose deviation appears as the first entry of `RecDevHist`. Generated automatically during `Populate()` from `SD` and `AC` if `NULL`.

`RecDevHist` `matrix`. Recruitment deviations (`Sim × nHistTS`) for historical time steps. The first column corresponds to the minimum age class in the initial year. Generated automatically if `NULL`.

`RecDevProj` `matrix`. Recruitment deviations (`Sim × nProjTS`) for projection time steps. Generated automatically if `NULL`.

`SpawnTimeFrac` `numeric(1)`. Fraction of the time step elapsed before spawning occurs. Controls how much mortality is applied to the population before spawning numbers are counted: numbers at spawning are `N × exp(-Z × SpawnTimeFrac)`. `0` = spawning at the start of the step (no mortality applied first); `1` = spawning at the end of the step (full within-step Z applied first). Default `0`.

`SpawnLag` `numeric(1)` or `NULL`. Number of timesteps between the spawning production evaluation and recruitment (first appearance of the youngest age class). When `NULL`

(default), the lag is derived automatically from `min(Ages@Classes)` and `Seasons` as `round(min(Ages@Classes) * Seasons)`. Set explicitly when the spawn season differs from what `min(Ages@Classes)` implies — for example, in SS3 models where `Spawn_month` and `birthseas` are decoupled.

`RelRecFun function` or `character(1)`. Relative recruitment function giving equilibrium recruitment relative to `R0` as a function of spawning-per-recruit (SPR). For built-in models, set automatically to `paste0(Model, "_RelRec")` (e.g., `"BevertonHolt_RelRec"`) during `Populate()`. For custom SRR models, must be supplied by the user with signature `function(Pars, SPR)`.

`Units numeric(1)`. Scaling factor for recruitment. `Units = 1` (default) means `R0` is in absolute numbers of fish. `Units = 1000` means `R0` is in thousands of fish. Does not affect internal calculations — used only to set the interpretation of numbers in hist output.

`Misc list`. Used internally.

### See Also

- `SRR()` for the constructor and accessor functions.
- `SRRModels()` for available stock-recruitment models and their required parameters.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.
- `Fecundity()` for the spawning production used as input to the SRR.
- Specifying Biological and Fleet Schedules for accepted `Pars` input formats.

Other srr: `SRR()`

---

`SRRModels` *Stock–Recruitment Relationships (SRR)*

---

### Description

Deterministic stock–recruitment and relative-recruitment models evaluated for a single simulation and a single year.

### Usage

```
BevertonHolt(S, S0, R0, h)

BevertonHolt_RelRec(Pars, SPR)

Ricker(S, S0, R0, hR)

Ricker_RelRec(Pars, SPR)

HockeyStick(S, S0, R0, Shinge)
```

```
HockeyStick_RelRec(Pars, SPR)

SRRModels(full = TRUE, print = TRUE)
```

## Arguments

| | |
|---|---|
| `S` | Spawning production (biomass or eggs). Numeric length >= 1 |
| `S0` | Unfished equilibrium spawning production. Numeric scalar. |
| `R0` | Unfished equilibrium recruitment. Numeric scalar. |
| `h` | Beverton–Holt steepness. A named argument in `Pars`. $0.2 < h < 1$ |
| `Pars` | Named list of parameters required by a specific SRR model. |
| `SPR` | Spawning-per-recruit multiplier. Numeric scalar. |
| `hR` | Ricker steepness. A named argument in `Pars`. |
| `Shinge` | Hockey-stick hinge point **relative to** `S0`. A named argument in `Pars`.$0 <$ Shinge $<= 1$ |
| `full` | Logical. Provide a complete table (TRUE) or just the model names (FALSE)? |
| `print` | Logical. Print out the results (TRUE) or just return the data.frame (FALSE)? |

## Details

This family of functions provides:

- **Stock–recruitment functions** that return expected recruitment given spawning production.
- **Relative-recruitment functions** that return equilibrium recruitment relative to `R0` as a function of spawning-per-recruit (SPR).

### Model listing:

`SRRModels()` prints or returns the set of stock–recruitment models available in the package.

### Parameterisation:

In `SRR()`, `Pars` for `BevertonHolt` and `Ricker` may be given either as steepness (`list(h = ...)` or `list(hR = ...)`, with `R0`) or as `list(alpha = ..., beta = ...)` with `R0 = NULL`. Alpha-beta values are converted to steepness and `R0` during `Populate()` using the first-year unfished spawning production per recruit $\phi_0$. See `SRRSteepness()` for the conversion.

### Beverton–Holt:

- `BevertonHolt()` evaluates expected recruitment:

$$R = \frac{\alpha S}{1 + \beta S}$$

where $\phi_0 = S_0/R_0$, $\alpha = 4h/((1-h)\phi_0)$, and $\beta = (5h-1)/((1-h)\phi_0 R_0)$.

- `BevertonHolt_RelRec()` evaluates equilibrium recruitment relative to unfished recruitment as a function of SPR.

**Ricker:**

- `Ricker()` evaluates expected recruitment:

$$R = \alpha S e^{-\beta S}$$

  where $\alpha = (5h_R)^{1.25}/\phi_0$ and $\beta = \log((5h_R)^{1.25})/(\phi_0 R_0)$.
- `Ricker_RelRec()` evaluates equilibrium relative recruitment as a function of SPR.

**Hockey Stick:**

- `HockeyStick()` evaluates recruitment assuming linear increase up to a hinge point followed by saturation at `R0`:

$$R = \begin{cases} \frac{R0}{Shinge} S, & S < Shinge \\ R0, & S \geq Shinge \end{cases}$$

- `HockeyStick_RelRec()` evaluates relative equilibrium recruitment as a function of SPR.

## Value

For stock–recruitment functions (`BevertonHolt()`, `Ricker()`, `HockeyStick()`): A numeric scalar giving expected recruitment for the specified spawning production.

For relative-recruitment functions (`*_RelRec()`): A numeric scalar giving equilibrium recruitment relative to unfished recruitment (`R / R0`) as a function of spawning-per-recruit (SPR).

For `SRRModels()`: If `print = TRUE`, prints the available stock–recruitment models to the console and invisibly returns the result. If `print = FALSE`, returns a character vector or data frame describing the available models.

## See Also

`SRR()`, `SRRSteepness()`

## Examples

```
## Compare stock-recruitment relationships

R0 <- 1000
S0 <- 1000
h <- 0.8
Shinge <- 0.2
S <- seq(0, S0, length.out = 200)

R_bh <- BevertonHolt(S, S0, R0, h)
R_rk <- Ricker(S, S0, R0, hR = h)
R_hs <- HockeyStick(S, S0, R0, Shinge)
```

```
plot(S, R_bh, type = "l", lwd = 2,
     ylim = c(0, 1200), xlab = "Spawning biomass (S)",
     ylab = "Recruitment")
lines(S, R_rk, lwd = 2, lty = 2)
lines(S, R_hs, lwd = 2, lty = 3)
abline(h=h*R0, lty=3)
abline(v=0.2*S0, lty=3)
legend("bottomright",
       legend = c("Beverton-Holt", "Ricker", "Hockey Stick"),
       lty = c(1, 2, 3), lwd = 2, bty = "n")


## Compare relative equilibrium recruitment vs SPR
SPR <- seq(0, 1, length.out = 200)

Pars_bh <- list(h = h)
Pars_rk <- list(hR = h)
Pars_hs <- list(Shinge = Shinge)

RR_bh <- sapply(SPR, BevertonHolt_RelRec, Pars = Pars_bh)
RR_rk <- sapply(SPR, Ricker_RelRec, Pars = Pars_rk)
RR_hs <- sapply(SPR, HockeyStick_RelRec, Pars = Pars_hs)

plot(SPR, RR_bh, type = "l", lwd = 2,
     ylim = c(0, 1.05), xlab = "SPR",
     ylab = "Relative recruitment (R / R0)")
lines(SPR, RR_rk, lwd = 2, lty = 2)
lines(SPR, RR_hs, lwd = 2, lty = 3)
legend("bottomright",
       legend = c("Beverton-Holt", "Ricker", "Hockey Stick"),
       lty = c(1, 2, 3), lwd = 2, bty = "n")
```

---

`SRRSteepness` *Convert Between Steepness and Alpha-Beta SRR Parameters*

---

## Description

Converts `BevertonHolt` and `Ricker` stock-recruitment parameters between the steepness parameterisation (steepness and `R0`) and the alpha-beta parameterisation, given the unfished spawning production per recruit `phi0`.

## Usage

```
SRRSteepness(alpha, beta, phi0, Model = c("BevertonHolt", "Ricker"))

SRRAlphaBeta(h, R0, phi0, Model = c("BevertonHolt", "Ricker"))
```

### Arguments

| | |
|---|---|
| alpha | Recruits per unit of spawning production at low spawning production (slope at the origin). |
| beta | Density-dependence parameter, in units of 1/spawning production. |
| phi0 | Unfished spawning production per recruit, e.g., from CalcSPR0(). |
| Model | "BevertonHolt" or "Ricker". |
| h | Steepness: h for BevertonHolt, hR for Ricker. |
| R0 | Unfished equilibrium recruitment. |

### Details

Beverton-Holt: $R = \alpha S/(1 + \beta S)$, with $h = \alpha\phi_0/(4 + \alpha\phi_0)$ and $R_0 = (\alpha\phi_0 - 1)/(\beta\phi_0)$.

Ricker: $R = \alpha S e^{-\beta S}$, with $h_R = (\alpha\phi_0)^{0.8}/5$ and $R_0 = \log(\alpha\phi_0)/(\beta\phi_0)$.

Both require $\alpha\phi_0 > 1$; otherwise the stock cannot persist in the absence of fishing.

Arguments are vectorised and recycled.

### Value

SRRSteepness(): a named list with elements h (hR for Ricker) and R0.

SRRAlphaBeta(): a named list with elements alpha and beta.

### See Also

SRR(), SRRModels(), CalcSPR0()

### Examples

```
ab <- SRRAlphaBeta(h = 0.7, R0 = 1000, phi0 = 2.5)
ab
SRRSteepness(ab$alpha, ab$beta, phi0 = 2.5)

# Specify a Beverton-Holt SRR directly with alpha and beta
SRR(Pars = list(alpha = ab$alpha, beta = ab$beta), Model = "BevertonHolt")
```

---

## SS2Data — *Reads data Stock Synthesis file structure into a Data object using package r4ss*

---

### Description

A function that uses the file location of a fitted SS3 model including input files to population the various slots of an Data object.

### Usage

```
SS2Data(
  SSdir,
  Name = "Imported by SS2Data",
  Common_Name = "",
  Species = "",
  Region = "",
  min_age_M = 1,
  gender = 1,
  comp_fleet = "all",
  comp_season = "sum",
  comp_partition = "all",
  comp_gender = "all",
  index_season = "mean",
  silent = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| `SSdir` | A folder with Stock Synthesis input and output files in it |
| `Name` | The name for the Data object |
| `Common_Name` | Character string for the common name of the stock. |
| `Species` | Scientific name of the species |
| `Region` | Geographic region of the stock or fishery. |
| `min_age_M` | Currently, the Data object supports a single value of M for all ages. The argument selects the minimum age for calculating the mean of age-dependent M from the SS assessment. |
| `gender` | An integer index for the sex for importing biological parameters (1 = female, 2 = male). |
| `comp_fleet` | A vector of indices corresponding to fleets in the assessment over which to aggregate the composition (catch-at-length and catch-at-age) data. By default, character string `"all"` will aggregate across all fleets. |
| `comp_season` | Integer, for seasonal models, the season for which the value of the index will be used. By default, `"mean"` will take the average across seasons. |
| `comp_partition` | Integer vector for selecting length/age observations that are retained (2), discarded (1), or both (0). By default, `"all"` sums over all available partitions. |
| `comp_gender` | Integer vector for selecting length/age observations that are female (1), male (2), or both (0), or both scaled to sum to one (3). By default, `"all"` sums over all gender codes. |
| `index_season` | Integer, for seasonal models, the season for which the value of the index will be used. By default, `"mean"` will take the average across seasons. |
| `silent` | Logical. Suppress all messages? |
| `...` | Arguments to pass to SS_output |

### Value

An object of class Data.

### Note

Currently supports the version of r4ss on CRAN (v.1.24) and Github (v.1.34-40). Function may be incompatible with other versions of r4ss.

### Author(s)

T. Carruthers and Q. Huynh

### See Also

SS2OM

---

`SS2DataMOM` *Reads data Stock Synthesis file structure into a nested Data object analogous with multiMSE*

---

### Description

A function that uses the file location of a fitted SS3 model including input files to population the various slots of an Data object.

### Usage

```
SS2DataMOM(SSdir, age_M = NULL, comp_partition = 2, silent = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `SSdir` | A folder with Stock Synthesis input and output files in it. Alternatively, |
| `age_M` | A vector of ages to average across to calculate a single value of natural mortality. Currently, the Data object supports a single value of M for all ages. By default, `NULL` averages over all ages. |
| `comp_partition` | Integer vector for selecting length/age observations that are retained (2), discarded (1), or both (0). By default, only retained comps are used. If multiple codes are used, then comp matrix is the sum over all codes. |
| `silent` | Logical. Suppress messages? |
| `...` | Arguments to pass to SS_output |

### Value

A nested list of Data objects, with the first index by stock/sex and the second index by fleet.

### Note

Currently tested on r4ss version 1.38.1-41 and SS 3.30.14.

Catches in `Data@Cat` are the predicted sex-specific catch calculated from the SS output.

### Author(s)

Q. Huynh

### See Also

SS2MOM

---

`SS2MOM` *Import Stock Synthesis to MOM (2-sex multi-fleet) or OM (single-sex, single-fleet)*

---

### Description

Functions that uses the file location or the r4ss output list of a fitted SS3 model including input files to populate the various slots of an `MOM` or `OM` object. `SS2MOM` and `SS2OM` mainly populates the Stock and Fleet components components of the operating model. `SS2MOM` creates a 2-sex model and multiple fleets with discarding behavior. `SS2OM` returns a single sex (either male, female, or averaged biological parameters) and single fleet (aggregate selectivity and mortality, no explicit discarding modeled). For either, the user still needs to parameterize most of the observation and implementation portions. `SSMOM2OM` is the internal function that simplifies the MOM object to an OM object. `plot_SS2OM` generates a markdown report to compare the OM and SS output.

### Usage

```
SS2MOM(
  SSdir,
  nsim = 48,
  proyears = 50,
  reps = 1,
  maxF = 3,
  seed = 1,
  interval = 1,
  pstar = 0.5,
  comp_partition = 2,
  Obs = MSEtool::Generic_Obs,
  Imp = MSEtool::Perfect_Imp,
  silent = FALSE,
  Name = "MOM generated by SS2MOM",
  Source = "No Source provided",
  ...
)
```

```
plot_SS2MOM(
  x,
  SSdir,
  gender = 1:2,
  filename = "SS2MOM",
  dir = tempdir(),
  open_file = TRUE,
  silent = FALSE,
  ...
)

SS2OM(
  SSdir,
  nsim = 48,
  proyears = 50,
  reps = 1,
  maxF = 3,
  seed = 1,
  interval = 1,
  pstar = 0.5,
  comp_partition = 2,
  Obs = MSEtool::Generic_Obs,
  Imp = MSEtool::Perfect_Imp,
  import_mov = TRUE,
  gender = 1:2,
  seasons_to_years = TRUE,
  model_discards = TRUE,
  silent = FALSE,
  Name = "OM generated by SS2OM function",
  Source = "No source provided",
  Author = "No author provided",
  report = FALSE,
  filename = "SS2OM",
  dir = tempdir(),
  open_file = TRUE,
  ...
)

SSMOM2OM(
  MOM,
  SSdir,
  gender = 1:2,
  import_mov = TRUE,
  seed = 1,
  silent = FALSE,
  model_discards = TRUE
)
```

```
plot_SS2OM(
  x,
  SSdir,
  gender = 1:2,
  filename = "SS2OM",
  dir = tempdir(),
  open_file = TRUE,
  silent = FALSE,
  ...
)

MOM_agg_fleets(MOM)
```

## Arguments

| | |
|---|---|
| SSdir | A folder with Stock Synthesis input and output files in it. |
| nsim | The number of simulations to take for parameters with uncertainty (for OM@cpars custom parameters). |
| proyears | The number of projection years for MSE |
| reps | The number of stochastic replicates within each simulation in the operating model. |
| maxF | The maximum allowable F in the operating model. |
| seed | The random seed for the operating model. |
| interval | The interval at which management procedures will update the management advice in multiMSE, e.g., 1 = annual updates. |
| pstar | The percentile of the sample of the management recommendation for the MP/MMP. |
| comp_partition | Integer vector for selecting length/age observations that are retained (2), discarded (1), or both (0). By default, only retained comps are used. If multiple codes are used, then comp matrix is the sum over all codes. |
| Obs | The observation model (class Obs). These functions do not update implementation parameters. |
| Imp | The implementation model (class Imp). These functions do not update implementation parameters. |
| silent | Whether to silence messages to the console. |
| Name | The name of the operating model |
| Source | Reference to assessment documentation e.g. a url |
| ... | Arguments to pass to SS_output. |
| x | For plot_SS2OM, an object of either class OM or Hist. For plot_SS2MOM, an object of either class MOM or multiHist. |
| gender | An integer that indexes the sex for importing life history parameters (1 = usually female, 2 = usually male, 1:2 = mean across both sexes). Only used for SS2OM only in a 2-sex model. |

`filename` If `report = TRUE`, character string for the name of the markdown and HTML files.

`dir` If `report = TRUE`, the directory in which the markdown and HTML files will be saved.

`open_file` If `report = TRUE`, whether the HTML document is opened after it is rendered.

`import_mov` Logical. Import movement matrix?

`seasons_to_years` Logical, when season is the time step, whether to convert OM from a seasonal model to annual model.

`model_discards` Logical, how to simplify a multi-fleet SS model to an OM object. If TRUE, OM will still model discards using the mean retention across fleets (weighted by fleet F). Otherwise, no discards are modeled and all fishing removals are calculated in the OM from the SS F-at-age matrix.

`Author` Who did the assessment

`report` Logical, if TRUE, the function will run runMSE to generate the Hist object from the operating model to compare against SS output. A markdown report will be generated.

`MOM` MOM object

## Value

SS2MOM returns an object of class `MOM`. SS2OM returns an object of class `OM`.

## Functions

- `MOM_agg_fleets()`: Aggregate all fleets in an MOM object.

## Note

Currently tested on r4ss version 1.38.1-40.0 and SS 3.30.14.

## Author(s)

Q. Huynh

## See Also

SS2Data SS2DataMOM

---

`SSBrefplot` *Plot Spawning stock biomass and reference points for both historical and projected period*

---

## Description

Plot Spawning stock biomass and reference points for both historical and projected period

## Usage

```
SSBrefplot(MSE, simno = 1, ystart = 1, log = F, leg = T)
```

## Arguments

| | |
|---|---|
| `MSE` | An object of class 'MSE' produced by from runMSE() |
| `simno` | Positive integer, the simulation number you wish to plot |
| `ystart` | Positive integer, the calendar year corresponding with the first historical year |
| `log` | Boolean, whether log SSB and reference points should be plotted |
| `leg` | Boolean, should a legend be included in the plot? |

## Author(s)

T. Carruthers

---

`StandardizeEffort` *Standardize fishing effort across stocks for each fleet*

---

## Description

For a multi-stock operating model, effort for a given fleet must be identical across stocks. When stocks have been specified with differing effort trajectories (e.g. derived from stock-specific estimated Fs), this function standardizes effort to the geometric mean across **active stocks** (those with positive effort) at each time step, and back-calculates stock-fleet targeting weights that preserve effective fishing mortality.

## Usage

```
StandardizeEffort(
  OM,
  populate = TRUE,
  fit_stock_targeting = TRUE,
  generate_stock_targeting = TRUE,
  record_assumption = TRUE
)
```

### Arguments

| | |
|---|---|
| `OM` | An operating model object (om). |
| `populate` | `logical(1)`. Populate `OM` at beginning of call? Used internally. Default `TRUE`. |
| `fit_stock_targeting` | `logical(1)`. Run `FitStockTargeting()` after standardizing? Default `TRUE`. |
| `generate_stock_targeting` | `logical(1)`. Run `GenerateStockTargeting()` after fitting? Default `TRUE`. Ignored (with a warning) if `fit_stock_targeting = FALSE`, since `GenerateStockTargeting()` requires fitted targeting parameters. |
| `record_assumption` | `logical(1)`. Record the effort-standardization as an `"assumption"` in `OM@Log`? Default `TRUE`. Set `FALSE` when the differing per-stock effort is structural to how `OM` was built (e.g. a sex-structured `ImportSS()` model, where per-sex effort naturally differs due to sex-specific selectivity) rather than an unexpected input. |

### Details

For each fleet `fl`, year `t`, and simulation `i`, the standardized effort is the geometric mean of stock-specific effort over stocks with positive effort at that time step:

$$E_{f,t} = \exp\left(\frac{1}{|\mathcal{A}_{f,t}|}\sum_{s\in\mathcal{A}_{f,t}} \log E_{s,f,t}\right)$$

where $\mathcal{A}_{f,t} = \{s : E_{s,f,t} > 0\}$ is the set of active stocks for fleet $f$ at time $t$.

The targeting weight for each stock is then:

$$\delta_{s,f,t} = \frac{E_{s,f,t}}{E_{f,t}}$$

so that effective fishing mortality is conserved:

$$E_{f,t} \times \delta_{s,f,t} \times q_{s,f} = E_{s,f,t} \times q_{s,f} = F_{s,f,t}$$

By construction, the targeting weights have geometric mean 1 over active stocks at each time step — equivalently, the log-targeting deviations sum to zero:

$$\sum_{s\in\mathcal{A}_{f,t}} \log \delta_{s,f,t} = 0$$

Inactive stocks ($E_{s,f,t} = 0$) receive targeting weight $\delta_{s,f,t} = 0$.

These targeting weights are stored in `OM@StockTargeting@Targeting` and are subsequently used by `FitStockTargeting()` to estimate the historical covariance of log-targeting deviations.

If populated, the effort `Distribution` array must be identical across stocks for each fleet. The function returns `OM` unchanged if `nStock == 1` (any pre-existing `StockTargeting` data is also cleared in this case, as it is not meaningful for single-stock models).

### Value

The input `OM` object with:

- `Fleet[[st]][[fl]]@Effort@Effort` set to the geometric mean effort over active stocks for **all stocks** in each fleet where effort differed across any stock.
- `OM@StockTargeting@Targeting` populated with back-calculated targeting weights $\delta_{s,f,t}$. Over active stocks at each time step, the weights have geometric mean 1 (equivalently, their logs sum to zero). Inactive stocks receive weight 0.

### See Also

`FitStockTargeting()`, `GenerateStockTargeting()`

### Examples

```
## Not run:
OM_standardized <- StandardizeEffort(OM)
# Inspect back-calculated targeting
OM_standardized@StockTargeting@Targeting

## End(Not run)
```

---

## `Stock` *Stock Constructor and Accessors*

---

### Description

Construct a stock object defining the biological and population-dynamics properties of a stock, or access and replace the `Stock` slot of an enclosing S4 object (e.g., an om object).

### Usage

```
Stock(
  Name = "New Stock Object",
  CommonName = NULL,
  Species = NULL,
  Ages = NULL,
  Length = NULL,
  Weight = NULL,
  NaturalMortality = NULL,
  Maturity = NULL,
  Fecundity = NULL,
  SRR = NULL,
  Spatial = NULL,
```

```
  Depletion = NULL,
  Seasons = 1,
  Misc = list()
)

Stock(x) <- value

CommonName(x)

CommonName(x) <- value

Species(x)

Species(x) <- value
```

### Arguments

| | |
|---|---|
| `Name` | `character(1)`. Unique stock identifier used throughout model output. Alternatively, an S4 object that contains a `Stock` slot (e.g., an om object), in which case `Stock()` acts as a pass-through accessor — see *Pass-Through Access*. Default `"New Stock Object"`. |
| `CommonName` | `character(1)`. Common name of the species (e.g., `"Atlantic cod"`). When `Name` is an S4 object and `CommonName` is `numeric`, it is treated as a list index for multi-stock `Stock` slots — see *Pass-Through Access*. Default `NULL`. |
| `Species` | `character(1)`. Scientific name of the species (e.g., `"Gadus morhua"`). Default `NULL`. |
| `Ages` | An ages object defining the age structure and plus-group age. If `NULL` (default), an empty ages is created via `Ages()`. |
| `Length` | A length object specifying the length-at-age growth schedule. If `NULL` (default), created via `Length()`. |
| `Weight` | A weight object specifying the weight-at-age and weight-at-length schedules. If `NULL` (default), created via `Weight()`. |
| `NaturalMortality` | A naturalmortality object specifying natural mortality rate(s), optionally age- or length-varying. If `NULL` (default), created via `NaturalMortality()`. |
| `Maturity` | A maturity object specifying the maturity schedule as a function of age, length, or weight. If `NULL` (default), created via `Maturity()`. |
| `Fecundity` | A fecundity object specifying egg production as a function of age or length. Optional. When `NULL` (default), spawning production (`SProduction`) equals spawning biomass (`SBiomass`, i.e. mature weight-at-age). When populated, `SProduction` is in the units of the `Fecundity` object (e.g. eggs). If `NULL` (default), created via `Fecundity()`. |
| `SRR` | A srr object specifying the stock-recruitment relationship and recruitment variability. Created via `SRR()` |

| | |
|---|---|
| `Spatial` | A spatial object specifying spatial structure and movement dynamics. Optional — leave empty for single-area (non-spatial) models. If `NULL` (default), created via `Spatial()`. |
| `Depletion` | A depletion object specifying depletion assumptions at the start (`Initial`) and/or end (`Final`) of the historical period, relative to a reference biomass (default `"B0"`). When `Final` is specified, `Efficiency` in `Catchability()` is optimised to achieve the target terminal depletion, overwriting any existing values. If `NULL` (default), created via `Depletion()`. |
| `Seasons` | `integer(1)`. Number of seasons per calendar year. Use values greater than `1` for within-year dynamics (e.g., `4` for quarterly seasons). Default `1`. Must match the time units set in `Ages()`. |
| `Misc` | `list`. Used internally. Default `list()`. |
| `x` | A stock object, or an S4 object with a `Stock` slot, for use with accessor and replacement functions. |
| `value` | For `Stock<-`: a stock object, or a list of stock objects for multi-stock operating models. For `CommonName<-` and `Species<-`: a `character(1)` string. |

## Details

### Object Structure:

A stock object aggregates all biological components needed to simulate stock dynamics:

- age structure (`Ages()`);
- somatic growth (`Length()`, `Weight()`);
- survival (`NaturalMortality()`);
- reproduction (`Maturity()`, `Fecundity()`, `SRR()`);
- initial conditions (`Depletion()`);
- and optional spatial structure (`Spatial()`).

### Required vs Optional Components:

`Ages()`, `Length()`, `Weight()`, `NaturalMortality()`, `Maturity()`, and `SRR()` are required. If any are omitted, empty objects with default slots are created automatically, but the model will not run until the required parameters within each sub-object are populated.

`Fecundity()` and `Spatial()` are optional. Omitting `Fecundity()` causes `SProduction` to equal `SBiomass`. Omitting `Spatial()` implies a single well-mixed area.

`Depletion()` is also optional. Omitting it (or leaving both `Initial` and `Final` as `NULL`) means the stock starts unfished and terminal depletion is determined by the `Fleet()` and `Catchability()` parameters.

### Pass-Through Access from an Enclosing Object:

When `Name` is an S4 object with a `Stock` slot (e.g., an om), `Stock()` returns the slot rather than constructing a new object:

- `Stock(om)` returns `om@Stock`.
- `Stock(om, i)` returns `om@Stock[[i]]` when `om@Stock` is a list (multi-stock model), or `om@Stock` when it is a single stock.

The index `i` is passed via the `CommonName` argument.

**Bookkeeping Slots:**

The slots `nYear`, `pYear`, `nSim`, `CurrentYear`, and `Years` are initialised to working defaults (`20`, `30`, `48`, the current calendar year, and the corresponding year vector). These are overwritten automatically when the stock is attached to an `OM()`, so they need not be set manually.

## Value

- `Stock()` returns a stock object. If `Name` is an S4 object with a `Stock` slot, returns that slot (or an indexed element for list-valued slots).
- `Stock<-` returns `x` with the `Stock` slot replaced by `value`.
- `CommonName()` and `Species()` return a `character(1)` string from the corresponding slot of `x`.
- `CommonName<-` and `Species<-` return `x` with the named slot updated.

## See Also

- stock for the class definition and slot-level documentation.
- `OM()` for the operating model constructor.
- `Ages()`, `Length()`, `Weight()`, `NaturalMortality()`, `Maturity()`, `Fecundity()`, `SRR()`, `Spatial()`, `Depletion()` for the sub-object constructors.
- Specifying Biological and Fleet Schedules for how `Pars`, `Model`, and `MeanAt*` arrays are specified across sub-objects.

Other stock: `stock-class`

## Examples

```
# ---- Minimal stock (required sub-objects only) ----

## Sub-objects with empty Pars are created automatically.
## The model will not run until each sub-object is fully specified.
## Nominally Gadus morhua but with made-up paramater values!!
stk <- Stock(Name = "Atlantic cod", CommonName = "Atlantic cod",
             Species = "Gadus morhua")
stk

# ---- Fully specified stock ----

stk <- Stock(
  Name             = "Atlantic cod",
  CommonName       = "Atlantic cod",
  Species          = "Gadus morhua",
  Ages             = Ages(MaxAge = 20),
  Length           = Length(Pars = list(Linf = 120, K = 0.2, t0 = -0.1)),
  Weight           = Weight(Pars = list(a = 0.008, b = 3.0)),
  NaturalMortality = NaturalMortality(Pars = list(M = 0.2)),
  Maturity         = Maturity(Pars = list(L50 = 50, L50_95 = 10)),
```

```
  SRR             = SRR(Pars = list(h = 0.7), R0 = 1000, SD = 0.4),
  Depletion       = Depletion(Final = 0.4)
)
stk

# ---- Accessing sub-objects ----

Ages(stk)
Length(stk)
NaturalMortality(stk)
SRR(stk)
Depletion(stk)

# ---- Replacing sub-objects ----

Length(stk) <- Length(Pars = list(Linf = c(100, 140), K = 0.2, t0 = -0.1))
Maturity(stk) <- Maturity(Pars = list(A50 = 4, A50_95 = 2))

# ---- Species name accessors ----

CommonName(stk)
Species(stk)
CommonName(stk) <- "Cod"
Species(stk)    <- "Gadus morhua"

# ---- Seasonal model ----

## Seasons = 4 creates quarterly time steps within each calendar year.
stk_seasonal <- Stock(
  Name    = "Seasonal stock",
  Ages    = Ages(MaxAge = 40, Units = "quarter"),
  Seasons = 4
)

Years(stk_seasonal, 'H')

# ---- Multi-stock pass-through access ----

## When Name is an OM or other S4 object with a Stock slot, Stock() is an
## accessor. For a list-valued Stock slot, pass an integer via CommonName.
## om <- OM(Stock = list(stk1, stk2))
## Stock(om)      # returns om@Stock (the list)
## Stock(om, 1)   # returns om@Stock[[1]]

# ---- Attaching to an OM ----

## om <- OM()
## Stock(om) <- stk
## Stock(om)
```

`stock-class` *The* `stock` *S4 Class*

## Description

Defines the biological and population-dynamics properties of a stock for use in an operating model. Objects are typically created via `Stock()`, which documents all parameters in detail and initialises sub-objects automatically.

## Details

This class is the central biological object in the operating model framework. The bookkeeping slots `nYear`, `pYear`, `nSim`, `CurrentYear`, and `Years` are populated with working defaults by `Stock()` and overridden automatically when the stock is attached to an `OM()`; they need not be set manually.

Direct construction via `methods::new()` is not recommended; use `Stock()` instead, which validates inputs and initialises all sub-objects.

## Slots

`Name character`. Unique stock identifier.

`CommonName character`. Common name of the species.

`Species character`. Scientific name of the species.

`Ages` An ages object defining the age structure. See `Ages()`.

`Length` A length object defining the length-at-age schedule. See `Length()`.

`Weight` A weight object defining the weight-at-age and weight-at-length schedules. See `Weight()`.

`NaturalMortality` A naturalmortality object defining the natural mortality schedule. See `NaturalMortality()`.

`Maturity` A maturity object defining the maturity schedule. See `Maturity()`.

`Fecundity` A fecundity object defining egg production as a function of age or length. Optional — when empty, spawning production (`SProduction`) equals spawning biomass (`SBiomass`). See `Fecundity()`.

`SRR` A srr object defining the stock-recruitment relationship and recruitment variability. See `SRR()` and `SRRModels()`.

`Spatial` A spatial object defining spatial structure and movement dynamics. Optional — when empty, a single well-mixed area is assumed. See `Spatial()`.

`Depletion` A depletion object defining depletion assumptions at the start (`Initial`) and/or end (`Final`) of the historical period. Optional — when empty, the stock starts unfished and terminal depletion is determined by the `Fleet()` and `Catchability()` parameters. See `Depletion()`.

`nYear numeric(1)`. Number of historical years. Initialised to 20 by `Stock()` and overridden when attached to an `OM()`.

`pYear numeric(1)`. Number of projection years. Initialised to 30 by `Stock()` and overridden when attached to an `OM()`.

`nSim numeric(1)`. Number of stochastic simulations. Initialised to `48` by `Stock()` and overridden when attached to an `OM()`.

`CurrentYear numeric(1)`. Final calendar year of the historical period. Initialised to the current system year by `Stock()` and overridden when attached to an `OM()`.

`Years numeric`. Vector of all model years (historical + projection), derived from `nYear`, `pYear`, `CurrentYear`, and `Seasons`. Overridden when attached to an `OM()`.

`Seasons numeric(1)`. Number of seasons per calendar year.

`Misc list`. Used internally.

`Log list`. Internal named list storing diagnostics, warnings, and assumptions recorded during processing. See `Log()`. Not intended for direct user access.

### See Also

- `Stock()` for the constructor and accessor functions.
- `Ages()`, `Length()`, `Weight()`, `NaturalMortality()`, `Maturity()`, `Fecundity()`, `SRR()`, `Spatial()`, `Depletion()` for the sub-object constructors.
- `OM()` for the operating model constructor.
- Specifying Biological and Fleet Schedules for how `Pars`, `Model`, and `MeanAt*` arrays are specified across sub-objects.

Other stock: `Stock()`

---

## `Stock-legacy-class` *Class* `'Stock'`

---

### Description

An operating model component that specifies the parameters of the population dynamics model

### Slots

`Name` An identifying name for the Stock object. Single value. Character string.

`Common_Name` Common name of the species. Character string.

`Species` Scientific name of the species. Genus and species name. Character string.

`maxage` The maximum age of individuals that is simulated. There are `maxage+1` (recruitment to age-0) age classes in the storage matrices. `maxage` is the 'plus group' where all age-classes > `maxage` are grouped, unless option switched off with `OM@cpars$plusgroup=0`. Single value. Positive integer.

`R0` Initial number of unfished recruits to age-0. This number is used to scale the size of the population to match catch or data, but does not affect any of the population dynamics unless the OM has been conditioned with data. As a result, for a data-limited fishery any number can be used for `R0`. In data-rich stocks `R0` may be estimated as part of a stock assessment, but for data limited stocks users can choose either an arbitrary number (say, 1000) or choose a number that produces simulated catches in recent historical years that are similar to real world catch data. Single value. Positive real number.

`M` The instantaneous rate of natural mortality. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Uniform distribution lower and upper bounds. Non-negative real numbers.

`Msd` Inter-annual variation in `M` expressed as a coefficient of variation of a log-normal distribution. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. If this parameter is positive, yearly `M` is drawn from a log-normal distribution with a mean specified by `log(M)` drawn for that simulation and a standard deviation in log space specified by the value of `Msd` drawn for that simulation. Uniform distribution lower and upper bounds. Non-negative real numbers

`h` Steepness of the stock recruit relationship. Steepness governs the proportion of unfished recruits produced when the stock is at 20% of the unfished population size. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years of a given simulation. Uniform distribution lower and upper bounds. Values from 1/5 to 1.

`SRrel` Type of stock-recruit relationship. Use 1 to select a Beverton Holt relationship, 2 to select a Ricker relationship. Single value. Integer

`Perr` Recruitment process error, which is defined as the standard deviation of the recruitment deviations in log space. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Uniform distribution lower and upper bounds. Non-negative real numbers.

`AC` Autocorrelation in the recruitment deviations in log space. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided, and used to add lag-1 auto-correlation to the log recruitment deviations. Uniform distribution lower and upper bounds. Non-negative real numbers.

`Linf` The von Bertalanffy growth parameter Linf, which specifies the average maximum size that would reached by adult fish if they lived indefinitely. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years unless `Linfsd` is a positive number. Uniform distribution lower and upper bounds. Positive real numbers.

`Linfsd` Inter-annual variation in Linf. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. If this parameter has a positive value, yearly Linf is drawn from a log-normal distribution with a mean specified by the value of `Linf` drawn for that simulation and a standard deviation (in log space) specified by the value of `Linfsd` drawn for that simulation. Uniform distribution lower and upper bounds. Non-negative real numbers.

`K` The von Bertalanffy growth parameter k, which specifies the average rate of growth. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This value is the same in all years unless `Ksd` is a positive number. Uniform distribution lower and upper bounds. Positive real numbers.

`Ksd` Inter-annual variation in K. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. If this parameter has a positive value, yearly K is drawn from a log-normal distribution with a mean specified by the value of `K` drawn for that simulation and a standard deviation (in log space) specified by the value of `Ksd` drawn for that simulation. Uniform distribution lower and upper bounds. Non-negative real numbers.

`t0` The von Bertalanffy growth parameter t0, which specifies the theoretical age at a size 0. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Uniform distribution lower and upper bounds. Non-positive real numbers.

`LenCV` The coefficient of variation (defined as the standard deviation divided by mean) of the length-at-age. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided to specify the distribution of observed length-at-age, and the CV of this distribution is constant for all age classes (i.e, standard deviation increases proportionally with the mean). Uniform distribution lower and upper bounds. Positive real numbers.

`L50` Length at 50% maturity. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. The `L50` and `L50_95` parameters are converted to ages using the growth parameters provided and used to construct a logistic curve to determine the proportion of the population that is mature in each age class. Uniform distribution lower and upper bounds. Positive real numbers.

`L50_95` Difference in lengths between 50% and 95% maturity. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. The value drawn is then added to the length at 50% maturity to determine the length at 95% maturity. This parameterization is used `instead` of specifying the size at 95 percent maturity to avoid situations where the value drawn for the size at 95% maturity is smaller than that at 50% maturity. The `L50` and `L50_95` parameters are converted to ages using the growth parameters provided and used to construct a logistic curve to determine the proportion of the population that is mature in each age class. Uniform distribution lower and upper bounds. Positive real numbers.

`D` Estimated current level of stock depletion, which is defined as the current spawning stock biomass divided by the unfished spawning stock biomass. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. This parameter is used during model initialization to select a series of yearly historical recruitment values and fishing mortality rates that, based on the information provided, could have resulted in the specified depletion level in the simulated last historical year. Uniform distribution lower and upper bounds. Positive real numbers (typically $< 1$)

`a` The alpha parameter in allometric length-weight relationship. Single value. Weight parameters are used to determine catch-at-age and population-at-age from the number of individuals in each age class and the length of each individual, which is drawn from a normal distribution determined by the `Linf`, `K`, `t0`, and `LenCV` parameters. As a result, they function as a way to scale between numbers at age and biomass, and are not stochastic parameters. Single value. Positive real number.

`b` The beta parameter in allometric length-weight relationship. Single value. Weight parameters are used to determine catch-at-age and population-at-age from the number of individuals in each age class and the length of each individual, which is drawn from a normal distribution determine by the `Linf`, `K`, `t0`, and `LenCV` parameters. As a result, they function as a way to scale between numbers at age and biomass, and are not stochastic parameters. Single value. Positive real number.

`Size_area_1` The size of area 1 relative to area 2. The fraction of the unfished biomass in area 1. Please specify numbers between 0 and 1. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided.

For example, if Size_area_1 is 0.2, then 20% of the total area is allocated to area 1. Fishing can occur in both areas, or can be turned off in one area to simulate the effects of a no take marine reserve. Uniform distribution lower and upper bounds. Positive real numbers.

`Frac_area_1` The fraction of the unfished biomass in area 1. Please specify numbers between 0 and 1. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. For example, if Frac_area_1 is 0.5, then 50% of the unfished biomass is allocated to area 1, regardless of the size of area 1 (i.e, size and fraction in each area determine the density of fish, which may impact fishing spatial targeting). In each time step recruits are allocated to each area based on the proportion specified in Frac_area_1. Uniform distribution lower and upper bounds. Positive real numbers.

`Prob_staying` The probability of individuals in area 1 remaining in area 1 over the course of one year. Please specify numbers between 0 and 1. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. For example, in an area with a Prob_staying value of 0.95 each fish has a 95% probability of staying in that area in each time step, and a 5% probability of moving to the other area. Uniform distribution lower and upper bounds. Positive fraction.

`Fdisc` The instantaneous discard mortality rate the stock experiences when fished using the gear type specified in the corresponding fleet object and discarded. For each simulation a single value is drawn from a uniform distribution specified by the upper and lower bounds provided. Uniform distribution lower and upper bounds. Non-negative real numbers.

`Source` A reference to a website or article from which parameters were taken to define the stock object. Single value. Character string.

### Objects from the Class

Objects can be created by calls of the form `new('Stock')`

### Author(s)

T. Carruthers and A. Hordyk

---

## `StockTargeting` *Stock Targeting Constructor*

---

### Description

Initialises a `stocktargeting` object with `NA`-filled arrays sized to the supplied operating model. All arrays are dimensioned and named but left unpopulated.

### Usage

```
StockTargeting(OM)
```

### Arguments

| | |
|---|---|
| `OM` | An om object. Must have at least one stock and one fleet. |

### Value

A `stocktargeting` object with:

- `Mean`: `NA` array of dimension `[nSim, nStock, nFleet]`
- `Covariance`: `NA` array of dimension `[nSim, nStock, nStock, nFleet]`
- `AC`: `NA` array of dimension `[nSim, nStock, nFleet]`
- `Targeting`: `NA` array of dimension `[nSim, nStock, nFleet, nHistYear]`
- `Misc`: empty list

If `nFleet < 1` or `nStock < 1` an empty (default) `stocktargeting` object is returned with a warning.

### See Also

stocktargeting

---

## `stocktargeting-class` *The* `stocktargeting` *S4 Class*

---

### Description

The `stocktargeting` object defines the properties of a fleet's ability to selectively target individual stocks in a mixed-stock fishery. Targeting weights scale the effective fishing effort of each fleet toward or away from each stock relative to the fleet mean.

### Slots

`Mean array[sim, stock, fleet]`. Mean targeting weight for each simulation, stock, and fleet. Values of 1 indicate neutral (equal) targeting across stocks.

`Covariance array[sim, stock_i, stock_j, fleet]`. Log-space residual covariance matrix of targeting weights for each fleet and simulation. Diagonal entries represent per-stock variance; off-diagonal entries represent co-variation in targeting between pairs of stocks.

`AC array[sim, stock, fleet]`. Lag-1 autocorrelation of log targeting deviations for each simulation, stock, and fleet. Currently not used but retained for future implementation.

`Targeting array[sim, stock, fleet, year]`. Realised targeting deviations in normal space. For each fleet and year, values represent each stock's share of effective fishing effort relative to the fleet mean. Values greater than 1 indicate preferential targeting; values less than 1 indicate avoidance.

`Misc list`. Reserved for additional information or user-defined extensions. Not used internally.

---

`stocktransition-class`

*The* `stocktransition` *S4 Class*

---

## Description

The `stocktransition` object defines an age-dependent reclassification of individuals from one stock's numbers-at-age bookkeeping into another's, e.g. sequential hermaphroditism (protogynous/protandrous sex change), or any other ontogenetic-stage or stock-of-origin transition modelled as separate stock objects.

## Details

Only raw numbers-at-age move between stocks; weight, maturity, natural mortality, and selectivity are all functions of (age, year, area, stock) applied fresh to whatever `Number` is present, so a transitioned individual is indistinguishable from one that started life in the `To` stock. Constructed via `Herm()`.

## Slots

`From` `character(1)` or `numeric(1)`. Source stock, specified by stock name (`character`) or 1-based integer index (`numeric`).

`To` `character(1)` or `numeric(1)`. Destination stock, specified by stock name (`character`) or 1-based integer index (`numeric`).

`Frac` `array` or `NULL`. Cumulative "fraction remaining in `From` by age" curve - `[sim, age]` or `[sim, age, year]`, values in `[0, 1]`. This is *not* transitioned/mover fraction: it starts near `1` (almost no one has transitioned yet) and decreases toward `0` as age increases (nearly everyone has transitioned out of `From` by the oldest ages). Converted internally to a per-age hazard rate applied each year.

`Misc` `list`. Reserved for additional information or user-defined extensions. Not used internally.

## See Also

`Herm()`

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `TwoFleetOM`, `om-class`

`Sub` *Subset MSE object by management procedure (MP) or simulation.*

## Description

Subset the MSE object by particular MPs (either MP number or name), or particular simulations, or a subset of the projection years (e.g., 1: < projection years).

## Usage

```
Sub(MSEobj, MPs = NULL, sims = NULL, years = NULL)
```

## Arguments

| | |
|---|---|
| `MSEobj` | A MSE object. |
| `MPs` | A vector MPs names or MP numbers to subset the MSE object. Defaults to all MPs. |
| `sims` | A vector of simulation numbers to subset the MSE object. Can also be a logical vector. Defaults to all simulations. |
| `years` | A numeric vector of projection years. Should start at 1 and increase by one to some value equal or less than the total number of projection years. |

## Author(s)

A. Hordyk

## See Also

SubOM for OM components and SubCpars for subsetting by simulation and projection years.

## Examples

```
## Not run:
MSE <- runMSE()
MSE_1 <- Sub(MSE, MPs=1:2)
MSE_1@MPs
MSE_2 <- Sub(MSE, sims=1:10)
MSE_2@nsim

## End(Not run)
```

SubCpars *Subset the cpars slot in an operating model*

## Description

Subset the custom parameters of an operating model by simulation and projection years

## Usage

```
SubCpars(x, ...)

## S4 method for signature 'OM'
SubCpars(x, sims = 1:x@nsim, proyears = x@proyears, silent = FALSE)

## S4 method for signature 'MOM'
SubCpars(x, sims = 1:x@nsim, proyears = x@proyears, silent = FALSE)
```

## Arguments

| | |
|---|---|
| x | An object of class OM or MOM |
| ... | Arguments for method. |
| sims | A logical vector of length x@nsim to either retain (TRUE) or remove (FALSE). Alternatively, a numeric vector indicating which simulations (from 1 to nsim) to keep. |
| proyears | If provided, a numeric to reduce the number of projection years (must be less than x@proyears). |
| silent | Logical to indicate if messages will be reported to console. |

## Details

Useful function for running multiMSE in batches if running into memory constraints.

## Value

An object of class OM or MOM (same class as x).

## Author(s)

T. Carruthers, Q. Huynh

## See Also

Sub for MSE objects, SubOM for OM components.

---

`SubOM` *Subset a Stock, Fleet, Obs, or Imp object from an OM object*

---

## Description

A function that strips out a Stock, Fleet, Obs, or Imp object from a complete OM object. Mainly used for internal functions.

## Usage

```
SubOM(OM, Sub = c("Stock", "Fleet", "Obs", "Imp"))
```

## Arguments

| | |
|---|---|
| `OM` | An operating model object (class OM) |
| `Sub` | A character string specifying what object type to strip out "Stock", "Fleet", "Obs", or "Imp" |

## Value

An object of class Stock, Fleet, Obs, or Imp

## Author(s)

A. Hordyk

## See Also

Sub for subsetting MSE output and SubCpars for subsetting by simulation and projection years.

## Examples

```
Stock <- SubOM(testOM, "Stock")
class(Stock)
```

---

`Subset` *Subset an object by simulation, year, age, MP, and/or fleet*

---

## Description

Recursively subsets an object along the `Sim`, `Year`, `Age`, `MP`, and/or `Fleet` & `Stock` dimensions. The function operates on arbitrarily nested structures including S4 objects, lists, arrays, and named numeric vectors.

### Usage

```
Subset(
  object,
  Sims = NULL,
  Years = NULL,
  Ages = NULL,
  MPs = NULL,
  Fleets = NULL,
  Stocks = NULL,
  Impute = TRUE
)
```

### Arguments

| | |
|---|---|
| `object` | An object to be subset. Supported types include:<br>• S4 objects (all slots are recursively subset)<br>• Lists (including `StockList`, `FleetList`, etc.)<br>• Arrays with a `Sim`, `Year`, `Age`, `MP`, and/or `Fleet` dimension<br>• Named numeric vectors (for `Sim` and `Age` dimensions)<br>• Data frames with a `Sim` column |
| `Sims` | Numeric vector of simulation indices to retain, in the order they are returned. `NULL` skips. An `om` is populated with `Populate()` before it is subset. Subsetting an `om`, `hist`, or `mse` object records the original simulation numbers, so projections of the subset reproduce those simulations of the full object. |
| `Years` | Numeric vector of calendar years to retain. `NULL` skips. |
| `Ages` | Numeric vector of age classes to retain. `NULL` skips. |
| `MPs` | Integer or character vector of MPs to retain. `NULL` skips. |
| `Fleets` | Character vector of fleets to retain. `NULL` skips. |
| `Stocks` | Character vector of stocks to retain. `NULL` skips. |
| `Impute` | Logical; only relevant when subsetting by `Years`. If `TRUE` (default), years not present in the object are imputed from the nearest available past year. If `FALSE`, years earlier than the earliest available year are dropped rather than imputed.<br>Subsetting is applied sequentially when multiple dimensions are supplied, allowing consistent extraction across simulations, years, and age classes in a single call. |

### Value

An object of the same class as `object`, subset according to the supplied dimension arguments.

### Examples

```
nSim(SingleStockOM) <- 10
OM <- Populate(SingleStockOM)

# Subset Sims and Years
RecDevs <- OM |>
  Stock(1) |>
  SRR() |>
  RecDevHist()

Subset(RecDevs, Sims=1:2, Years = 1997:2000)


# Subset Ages
MeanLength <- OM |>
  Stock(1) |>
  Length() |>
  MeanAtAge()

Subset(MeanLength, Sim=1:2, Ages=0:3)
```

---

## SummarizeConvergence *Summarize Effort/TAC Solver Convergence*

---

### Description

Reports how often the effort/TAC solver failed to converge within tolerance during a `Project()`/`runMSE()` run, and — where recorded — the per-fleet or per-complex TAC vs. achieved-catch shortfall behind each failure. TAC saturation (TAC unachievable within the effort ceiling, e.g. because the stock is too depleted) and expected multi-complex choke trade-offs are never logged, since they reflect stock dynamics or fleet allocation rather than a solver problem — every row returned here is a genuine solver failure.

### Usage

```
SummarizeConvergence(MSE, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `MSE` | An mse object. |
| `silent` | Logical. Suppress the console summary? Default `FALSE`. |

### Details

The TAC values reported are the *effective* TAC the solver actually targeted, i.e. after any implementation error (OM@Imp) has already been applied — the same target realised landings/removals should be compared against.

### Value

Invisibly, a data.frame with one row per fleet or complex involved in a logged failure: Sim, Year, MP, Level ("Fleet" or "Complex"), Name, TAC, Catch, PctAchieved (100 * Catch / TAC). Zero rows if nothing was logged. Level/Name/TAC/Catch/ PctAchieved are NA for log entries recorded before this per- fleet/complex detail was captured.

### See Also

Project(), runMSE(), MPFailureRate()

---

## summary,Data-method *Summary of Data object*

---

### Description

Summary of Data object

### Usage

```
## S4 method for signature 'Data'
summary(
  object,
  wait = TRUE,
  x = 1,
  plots = "all",
  rmd = FALSE,
  head = "##",
  tplot = 25
)
```

### Arguments

| | |
|---|---|
| object | An object of class Data |
| wait | Logical. Wait for key press before next plot? |
| x | iteration number for the Data object. |
| plots | Character. What plots to show? all, TS, CAA, CAL, PD for all plots, time-series, catch-at-age, catch-at-length, and probability distributions respectively |
| rmd | Logical. Used in a rmd file? |
| head | Character. Heading for rmd file. Default is '##' (second level heading) |
| tplot | Integer. Number of plots per page. Default 25 |

---

`summary,MMSE-method` *Summary of MMSE object*

---

## Description

Summary of MMSE object

## Usage

```
## S4 method for signature 'MMSE'
summary(object, ..., silent = FALSE, Refs = NULL)
```

## Arguments

| | |
|---|---|
| `object` | object of class `MMSE` |
| `...` | a list of names of PM methods |
| `silent` | Should summary be printed to console? Logical. |
| `Refs` | An optional named list (matching the PM names) with numeric values to override the default `Ref` values. See examples. |

---

`summary,MSE-method` *Summary of MSE object*

---

## Description

Summary of MSE object

## Usage

```
## S4 method for signature 'MSE'
summary(object, ..., silent = FALSE, Refs = NULL)
```

## Arguments

| | |
|---|---|
| `object` | object of class MSE |
| `...` | a list of names of PM methods |
| `silent` | Should summary be printed to console? Logical. |
| `Refs` | An optional named list (matching the PM names) with numeric values to override the default `Ref` values. See examples. |

`SumOverAge` *Sum over named array dimensions*

## Description

Convenience wrappers for summing an array over a specific named dimension. These functions provide fast, explicit alternatives to `apply()` when working with multi-dimensional arrays that use named dimensions.

## Usage

```
SumOverAge(x)

SumOverArea(x)

SumOverClass(x)

SumOverFleet(x)

SumOverStock(x)

SumOverYear(x)
```

## Arguments

| | |
|---|---|
| `x` | An array. If the requested dimension is named, it will be summed out; otherwise the last dimension is used as a fallback. |

## Details

If the requested dimension name is present in `names(dimnames(x))`, that dimension is summed out. If the dimension is not named (or `x` has no dimension names), the **last dimension** of the array is summed by default.

**Exported functions:**

- `SumOverAge()` – sum over the `"Age"` dimension
- `SumOverArea()` – sum over the `"Area"` dimension
- `SumOverClass()` – sum over the `"Class"` dimension
- `SumOverFleet()` – sum over the `"Fleet"` dimension
- `SumOverStock()` – sum over the `"Stock"` dimension
- `SumOverYear()` – sum over the `"Year"` dimension

## Value

An array with the specified dimension removed. Dimension names are preserved where possible.

`SurplusProduction` *Surplus Production Model Management Procedure*

## Description

A model-based management procedure that fits a surplus production model to the catch and one or more relative abundance indices with `FitSP()`, sets a fishing mortality from the estimated stock status with a harvest control rule, and projects the fitted model to calculate the TAC (or effort) for the next management interval.

## Usage

```
SurplusProduction(
  Data,
  Indices = NULL,
  IndexSource = "auto",
  IndexFreq = NULL,
  IndexWeight = NULL,
  IndexSeasons = NULL,
  IndexUnits = c("Biomass", "Number"),
  CatchType = c("Removals", "Landings"),
  FitYears = NULL,
  MinIndexYears = 5,
  Model = c("internal", "spict"),
  Shape = 2,
  EstShape = FALSE,
  ShapeCV = 0.5,
  Depletion = 1,
  EstDepletion = FALSE,
  IndexSD = c("estimate", "data"),
  Priors = list(FMSY = c(0.2, 2)),
  Control = NULL,
  FTarget = 1,
  HCRControlPointsBiomass = c(0, 0.5),
  HCRControlPointsRate = c(0, 1),
  RampType = c("linear", "smooth"),
  FractileB = NULL,
  FractileF = NULL,
  tunepar = 1,
  AdviceYear = NULL,
  Interval = NULL,
  GapCatch = c("TAC", "LastCatch"),
  TACYears = c("mean", "first"),
  AdviceType = c("TAC", "Effort"),
  TACType = c("Removals", "Landings"),
  LandedFracYears = 3,
  Responsiveness = 1,
```

```
  DeltaDown = c(0.01, 0.5),
  DeltaUp = c(0.01, 0.5),
  TACRange = NULL,
  Allocation = NULL,
  OnFail = c("hold", "trend"),
  FallbackYears = c(2, 3),
  Diagnostics = c("min", "none", "full")
)
```

## Arguments

| | |
|---|---|
| `Data` | A data object. |
| `Indices`, `IndexSource`, `IndexFreq`, `IndexWeight`, `IndexSeasons` | Index selection, as in `IndexRate()`. `IndexWeight` is normalised to a mean of `1`. |
| `IndexUnits` | Character vector of index `Units` to use: one or both of `'Biomass'` and `'Number'`. Indices with other units (including `'Recruitment'`) are excluded. Indices with no units are assumed to be in biomass. |
| `CatchType` | Character. The catch the model is fitted to: `'Removals'` (landings plus discards, default) or `'Landings'`. |
| `FitYears` | `NULL` (default) fits from the first year with catch data to the last data year. Otherwise a single number (fit to that many most recent years) or a vector of consecutive years. Missing catch after the first fitted year is set to `0`. |
| `MinIndexYears` | Positive integer. Minimum number of years with index observations required to fit the model. Default `5`. |
| `Model` | Character. The surplus production model: `'internal'` (default) or `'spict'`. See `FitSP()`. |
| `Shape` | Positive number. Shape parameter `n` (fixed value, or the median of its prior when `EstShape = TRUE`). Default `2` (Schaefer). |
| `EstShape` | Logical. Estimate `Shape`? Default `FALSE`. |
| `ShapeCV` | Positive number. CV of the lognormal prior on `Shape` when `EstShape = TRUE` and `Priors$Shape` is not given. Default `0.5`. |
| `Depletion` | Positive number. Biomass in the first fitted year relative to `K` (fixed value, or starting value when `EstDepletion = TRUE`). Default `1`. |
| `EstDepletion` | Logical. Estimate `Depletion`? Default `FALSE`. |
| `IndexSD` | Character. `'estimate'` (default) estimates the observation standard deviation of each index; `'data'` calculates it from the index `CV` in `Data` (with `Model = 'spict'`, the `CV`s scale the estimated standard deviation; see `SpictControl()`). See `FitSP()`. |
| `Priors` | Named list of lognormal priors, each `c(median, CV)`, on any of `FMSY`, `MSY` (in catch units), `Depletion`, and `Shape`. A prior on a fixed parameter is ignored. Default `list(FMSY = c(0.2, 2))`, a vague prior that keeps the `FMSY` estimate away from zero when the data contain little information on productivity. `list()` uses no priors. |

`Control` An `SPControl()` (`Model = 'internal'`) or `SpictControl()` (`Model = 'spict'`) list. `NULL` (default) uses the defaults of the selected model.

`FTarget` Positive number. Target fishing mortality as a fraction of the estimated `FMSY`. Default `1`.

`HCRControlPointsBiomass`
Numeric vector (length `>= 2`, non-decreasing). Harvest control rule control points in units of `B/BMSY`. Default `c(0, 0.5)`: fishing mortality declines linearly from the target at `0.5 BMSY` to zero at `B = 0`.

`HCRControlPointsRate`
Numeric vector, the same length as `HCRControlPointsBiomass`. Multipliers of the target fishing mortality at each control point. Default `c(0, 1)`.

`RampType` Character. `'linear'` (default) or `'smooth'`; see `HockeyStickHCR()`.

`FractileB`, `FractileF`
`NULL` (default) uses the point estimates of `B/BMSY` and `FMSY`. Otherwise a probability in `(0, 1)`; e.g. `0.35` uses the 35th percentile. See Details, step 3.

`tunepar` Positive number. Multiplier of the target fishing mortality used to tune the MP. Default `1`.

`AdviceYear` `NULL` (default) or the calendar year the advice first applies in. `NULL` uses `Data@Misc$AdviceYear` (set during a projection), else the year after the last data year.

`Interval` `NULL` (default) or the number of years the advice applies for. `NULL` uses `Data@Misc$Interval` (set during a projection), else `1`.

`GapCatch` Character. Catch assumed in the years between the last data year and `AdviceYear`: `'TAC'` (default; the TAC in force, see `LastTAC()`, converted to `CatchType`) or `'LastCatch'` (the catch in the last fitted year).

`TACYears` Character. `'mean'` (default; mean catch over the `Interval` years) or `'first'` (catch in the first year).

`AdviceType` Character. `'TAC'` (default) or `'Effort'`.

`TACType` Character. `'Removals'` (default) or `'Landings'`. See `Advice()`.

`LandedFracYears`
Positive integer. Number of most recent data years used to calculate the landed fraction of the removals, used when `TACType` differs from `CatchType`. Default `3`.

`Responsiveness`
Positive number. The change in advice is `exp(log(new / previous) * Responsiveness)`. Default `1`.

`DeltaDown`, `DeltaUp`
Numeric vectors, length 2 (`c(min, max)`). Minimum and maximum fractional decrease (`DeltaDown`) and increase (`DeltaUp`) in the advice between management cycles. Changes smaller than the minimum are set to zero; see `ConstrainTAC()`.

`TACRange` Numeric vector, length 2. Absolute bounds on the TAC. `NULL` (default) uses `c(0, 100 * max(catch))`. Ignored for effort.

| | |
|---|---|
| `Allocation` | `NULL` (default), or a non-negative numeric vector of length `nFleet` giving the fraction of the TAC allocated to each fleet (normalised to sum to `1`). See `IndexRate()`. Ignored for effort. |
| `OnFail` | Character. `'hold'` (default) or `'trend'`; see Details. |
| `FallbackYears` | Positive integer vector, length 2. Recent and preceding window lengths (years) for `OnFail = 'trend'`. Default `c(2, 3)`. |
| `Diagnostics` | Character. Information stored in `Advice@Misc$SurplusProduction`: `'min'` (default; the parameter estimates, a one-row summary, and a table of the indices used, used by `SPEstimates()`), `'none'` (the parameter estimates only, used to start the next fit), or `'full'` (also the `spfit` object, without `Prep` and `Spict`; see `FitSP()`). When the advice is set by `OnFail`, `'min'` is used. See Details, Diagnostics. |

## Details

### Model parameters:

The leading parameters of the surplus production model are:

- `FMSY`: the fishing mortality that produces MSY. Always estimated.
- `MSY`: the maximum sustainable yield, in catch units. Always estimated.
- `Depletion`: biomass at the start of the first fitted year relative to `K`. Fixed at `Depletion` (default `1`, an unfished stock at the start of the fitted years), or estimated with `EstDepletion = TRUE`.
- `Shape`: the shape parameter `n` of the production function, which sets `BMSY/K = n^(1/(1-n))`. `n = 2` (the default) is the Schaefer model (`BMSY/K = 0.5`), and `n = 1` the Fox model (`BMSY/K = exp(-1)`, the limit as `n` approaches `1`). Fixed at `Shape`, or estimated with `EstShape = TRUE` with a lognormal prior (median `Shape`, CV `ShapeCV`, or `Priors$Shape`).

The derived quantities are `BMSY = MSY / FMSY` and `K = BMSY / (BMSY/K)`. Lognormal priors can be set on any estimated leading parameter with `Priors`; by default `FMSY` has a vague prior (median `0.2`, CV `2`). With `Model = 'internal'`, the catchability of each index is calculated at its maximum likelihood value given the leading parameters, and the observation standard deviation of each index is either calculated the same way (`IndexSD = 'estimate'`) or set from the index `CV` (`IndexSD = 'data'`). See `FitSP()` for the model equations, and `SpictControl()` for how the leading parameters map to the `spict` parameterisation with `Model = 'spict'`, which also estimates process error and catch observation error.

### Procedure:

1. The model is fitted to the calendar-year data (see `FitSP()` for the models and the index selection arguments, and `AnnualData()` for seasonal data). Each fit starts from the estimates of the previous management cycle, if these were made with the same `Model` (`WarmStart` in `SPControl()`/`SpictControl()`).
2. Biomass at the start of `AdviceYear` is calculated by projecting the fitted model through any years between the last fitted year and `AdviceYear` (when there is a data lag, or `FitYears` ends before the last data year), with the catch set by `GapCatch`.

3. The fishing mortality is `HockeyStickHCR(tunepar * FTarget * FMSY, Est = B/BMSY, Ref = 1)`, where `B/BMSY` is the projected status at the start of `AdviceYear` and the control points are `HCRControlPointsBiomass`/`HCRControlPointsRate` (see `HockeyStickHCR()`). With `FractileB` (`FractileF`), the `FractileB` (`FractileF`) quantile of `B/BMSY` (`FMSY`) is used instead of the point estimate, assuming a lognormal distribution with the standard error of the estimate (for `B/BMSY`, the standard error at the start of the year after the last fitted year).
4. The model is projected at that fishing mortality for the `Interval` years of the management cycle. The TAC is the mean (`TACYears = 'mean'`) or first-year (`'first'`) catch. With `AdviceType = 'Effort'`, the advice is the effort relative to the last historical year, `F / F(YearLH)`.
5. The TAC is converted to `TACType` using the landed fraction of the removals over the last `LandedFracYears` data years (when `TACType` differs from `CatchType`). The change from the previous TAC (or effort) is scaled by `Responsiveness` and constrained by `DeltaDown` and `DeltaUp`, and the TAC is bounded by `TACRange` (see `ConstrainTAC()`). If the previous TAC is zero, the TAC is only bounded by `TACRange`.

If the data cannot be prepared, fewer than `MinIndexYears` years have index observations, the model fit fails or does not converge, the standard errors needed by `FractileB`/`FractileF` are not available, or (with `AdviceType = 'Effort'`) the estimated fishing mortality in the last historical year is zero, the advice is set by `OnFail`: `'hold'` keeps the previous TAC (or effort), and `'trend'` changes it by the ratio of the mean index over the last `FallbackYears[1]` fitted years to the mean over the `FallbackYears[2]` years before them (the geometric mean over indices, weighted by `IndexWeight`, and raised to `Responsiveness`). If the data cannot be prepared, `'trend'` keeps the previous advice. The fallback advice is constrained as in step 5. Each such management cycle is recorded as a warning in the `Log` of the returned advice object.

**Diagnostics:**

The estimates of each management cycle are stored in the `Misc` slot of the returned advice object, as the list `Advice@Misc$SurplusProduction`. Its contents are set by the `Diagnostics` argument:

- `Model` and `par`: the model and the parameter estimates, used as starting values for the next fit. Always stored.
- `Summary` (`'min'` and `'full'`): a one-row `data.frame` with `AdviceYear`, `LastDataYear`, `Converged`, `Fallback`, the estimated `B_BMSY` and `F_FMSY`, `BAdvice_BMSY` (the projected status at the start of `AdviceYear`), `MSY`, `FMSY`, `FAdvice` (the fishing mortality from the harvest control rule), `Advice` (the TAC or relative effort before allocation), and `Message` (`'Converged'`, or the reason the advice was set by `OnFail`).
- `Indices` (`'min'` and `'full'`): a `data.frame` with one row per index used in the fit: `Index`, `Source` (`'Survey'` or `'CPUE'`), `Units`, `Timing`, `Weight` (the normalised `IndexWeight`), `nObs` (number of positive observations in the fitted years), and the estimated `q` and `Sigma` (`NA` when the advice is set by `OnFail`). The relative weight of each index in the fit is `Weight / Sigma^2`.
- `Fit` (`'full'` only): the `spfit` object returned by `FitSP()`, including the estimated biomass and fishing mortality time series, the index catchabilities and standard deviations, the standard errors, and the `Log` of the fit, without the `Prep` and `Spict` elements. Not stored when the advice is set by `OnFail`.

In a projection, the advice objects are kept in `MSE@Misc$Advice`, indexed by MP, management year, simulation, and stock (complex). `SPEstimates()` combines the `Summary` of every management cycle into one `data.frame` and adds the operating model values of `B/BMSY` and `F/FMSY` (from `B_BMSY()` and `F_FMSY()`) for the same years, to compare the estimated and true stock status. `SPEstimates(Type = 'Indices')` combines the `Indices` of every management cycle. The fit for a single data set can be examined directly with `FitSP()`, using the same arguments.

Each management cycle with advice set by `OnFail` is also recorded as a `SurplusProduction` warning in `MSE@Log`, with the reason the fit was not used; `Log(MSE, type = 'warning')` lists these by MP, year, and simulation. The `Converged` and `Fallback` columns of `SPEstimates()` give the same information as a `data.frame`.

**Tuning:**

`tunepar = 1` applies the MP as specified; larger values increase the target fishing mortality proportionally. See `TuneMP()`.

## Value

An advice object.

## See Also

`FitSP()`, `SPControl()`, `SpictControl()`, `AnnualData()`, `SPEstimates()`, `Log()`, `B_BMSY()`, `F_FMSY()`, `HockeyStickHCR()`, `ConstrainTAC()`, `IndexRate()`, `TuneMP()`

## Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)
SurplusProduction(Hist@Data[[1]][[1]])

SP_Fox <- SetMPArgs(SurplusProduction, Shape = 1, FTarget = 0.8)
MSE <- Project(Hist, MPs = list(SP = SurplusProduction, SP_Fox = SP_Fox))
SPEstimates(MSE)

## End(Not run)
```

---

# TAC *Calculate TAC recommendations for more than one MP*

---

## Description

A function that returns the stochastic TAC recommendations from a vector of output control MPs given a data object Data

## Usage

```
TAC(Data, MPs = NA, reps = 100, timelimit = 1, checkMP = TRUE, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `Data` | A data-limited methods data object |
| `MPs` | optional vector of MP names |
| `reps` | Number of repetitions |
| `timelimit` | The maximum time (seconds) taken to complete 10 reps |
| `checkMP` | Logical. Check if the MP can be run first? |
| `silent` | Logical. Suppress messages? |

### Author(s)

T. Carruthers

### Examples

```
## Not run:
library(MSEtool)
Data <- TAC(MSEtool::Cobia)
plot(Data)

## End(Not run)
```

---

## `TACfilter` *TAC Filter*

---

### Description

Filters vector of TAC recommendations by replacing negatives with NA and and values beyond five standard deviations from the mean as NA

### Usage

```
TACfilter(TAC)
```

### Arguments

| | |
|---|---|
| `TAC` | A numeric vector of TAC recommendations |

### Author(s)

T. Carruthers

`TACs` *Extract TAC Recommendations from an MSE Object*

## Description

Extracts the total allowable catch (TAC) recommendations issued by each management procedure (MP) in each simulation, year, stock complex, and fleet from the `PPD` slot of an mse object. Returns a tidy data frame suitable for plotting and performance indicator calculations.

## Usage

```
TACs(MSE)
```

## Arguments

| | |
|---|---|
| `MSE` | An mse object. |

## Details

An MP's `Advice(TAC = ...)` only gets recorded in the year it actually changes (see `Advice()`: a TAC applies from the year it's issued "until a new Advice object is returned"), so the raw recommendation history is sparse. `TACs()` forward-fills each gap (via `ExtendYears()`) so the returned series has one row per year the TAC was actually in effect, not just the years it changed. Years before an MP's first recommendation (i.e. the historical period, where no MP has run yet) are absent rather than filled with a placeholder.

## Value

A data frame with columns:

- `Sim`: simulation index.
- `Stock`: stock complex name (`names(OM@Complexes)`). TAC advice is set per complex; for single-stock complexes this is the stock name.
- `MP`: management procedure name.
- `Year`: calendar year (or decimal date for sub-annual time steps).
- `Period`: always `"Projection"` for the reasons above.
- `Fleet`: fleet name.
- `Value`: TAC recommendation in the units of the operating model.
- `Variable`: always `"TAC"`.

## See Also

mse, `Advice()`, `PPD()`

---

`Taxa_Table` *Taxa_Table*

---

## Description

Database from rfishbase

## Usage

```
Taxa_Table
```

## Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 34721 rows and 8 columns.

## Source

doi:10.1111/j.10958649.2012.03464.x

## References

Carl Boettiger and Duncan Temple Lang and Peter Wainwright 2012. Journal of Fish Biology

---

`TEG` *Tom's expand grid*

---

## Description

Create an indexing grid from just a vector of maximum dimension sizes

## Usage

```
TEG(vec)
```

## Arguments

| | |
|---|---|
| `vec` | A vector of maximum array sizes |

## Author(s)

T. Carruthers

---

`testOM` *OM class objects*

---

## Description

Example objects of class OM

## Usage

```
testOM
```

## Format

An object of class `OM` of length 1.

## Examples

```
avail("OM")
```

---

`Thresh_tab` *Current default thresholds for DFO satisficing*

---

## Description

Crit_S is the probability of being in the critical zone in the first 10 projected years Caut_S is the probability of being in the cautious zone in the first 10 projected years Health_S is the probability of being in the healthy zone in the first 10 projected years OvFish_S is the probability of overfishing in the first 10 projected years Yield_S is the mean yield relative to FMSY management over the first 10 projected years Crit is the probability of being in the critical zone in the last 10 projected years Caut is the probability of being in the cautious zone in the last 10 projected years Health is the probability of being in the healthy zone in the last 10 projected years OvFish is the probability of overfishing in the last 10 projected years Yield is the mean yield relative to FMSY management over the last 10 projected years AAVY is the average annual variability in yield over the whole projection phrased as a CV percentage Reb is the probability the stock has rebuilt to over BMSY in 2 mean generation times

## Usage

```
Thresh_tab(Ptab1)
```

## Arguments

| | |
|---|---|
| `Ptab1` | A DFO performance table made by DFO_tab() |

## Author(s)

T. Carruthers

`timeseries-class` `Timeseries` *Class*

### Description

Stores detailed simulated time-series outputs including population, catch, effort, and fishing mortality components.

### Slots

`Number` List of number-at-age arrays Stock: Sim × Age × Year × Area (x MP for `MSE` objects).

`Biomass` Total biomass array(Sim × Stock × Year (x MP for `MSE` objects).

`SBiomass` Spawning biomass array Sim × Stock × Year (x MP for `MSE` objects).

`SProduction` Spawning production array Sim × Stock × Year (x MP for `MSE` objects).

`Interactions` Total interaction by Stock and Fleet (biomass). Array: Sim x Stock x Year x Fleet (x MP for `MSE` objects).

`Landings` Total landings by Stock and Fleet (biomass). Array: Sim x Stock x Year x Fleet (x MP for `MSE` objects).

`Discards` Total *dead* discards by Stock and Fleet (biomass). Array: Sim x Stock x Year x Fleet (x MP for `MSE` objects). Fish that are released and survive are counted in `Interactions` but not in `Discards`, so `Interactions` exceeds `Landings + Discards` whenever discard mortality is below 1.

`InteractAtAge` List of interactions-at-age (numbers) arrays. Stock: Sim × Age × Year × Fleet × Area (x MP for `MSE` objects).

`LandingsAtAge` List of landings-at-age (numbers) arrays. Stock: Sim × Age × Year × Fleet × Area (x MP for `MSE` objects).

`DiscardsAtAge` List of dead discards-at-age (numbers) arrays. Stock: Sim × Age × Year × Fleet × Area (x MP for `MSE` objects).

`LandingsAtSize` Nested list of landings-at-size (numbers) arrays Stock, Fleet: Sim × Class × Year × Area (x MP for `MSE` objects).

`DiscardsAtSize` Nested list of discards-at-size (numbers) arrays Stock, Fleet: Sim × Class × Year × Area (x MP for `MSE` objects).

`Effort` Effort array Sim × Year × Fleet (x MP for `MSE` objects).

`Distribution` Fleet effort distribution by area: Sim × Year × Fleet × Area (x MP for `MSE` objects).

`FInteract` Apical fishing mortality for fish that interact with the fishing gear. Array: Sim x Stock x Year x Fleet (x MP for `MSE` objects).'

`FDead` Apical fishing mortality for fish that are killed by fishing gear. Array: Sim x Stock x Year x Fleet (x MP for `MSE` objects).

`FRetain` Apical fishing mortality for fish that are retained by fishers. Array: Sim x Stock x Year x Fleet (x MP for `MSE` objects).

`FInteractArea` Area-specific fishing mortality-at-age for fish that interact with the fishing gear (list by stock). Stock: Sim x Age x Year x Fleet x Area (x MP for `MSE` objects).

`FDeadArea` Area-specific fishing mortality-at-age of fish killed by fishing gear (list by stock). Stock: Sim x Age x Year x Fleet x Area (x MP for `MSE` objects).

`FRetainArea` Area-specific fishing mortality-at-age of retained fish (list by stock). Stock: Sim x Age x Year x Fleet x Area (x MP for `MSE` objects).

`Misc` Miscellaneous list.

---

`tinyErr` *Remove observation, implementation, and process error*

---

## Description

Takes an existing OM object and converts it to one without any observation error, implementation error, very little process error, and/or gradients in life history parameters and catchability.

## Usage

```
tinyErr(x, ...)

## S4 method for signature 'OM'
tinyErr(x, obs = TRUE, imp = TRUE, proc = TRUE, grad = TRUE, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `OM` |
| `...` | Arguments to generic function |
| `obs` | Logical. Remove observation error? `Obs` is replaced with `Perfect_Info` |
| `imp` | Logical. Remove implementation error? `Imp` is replaced with `Perfect_Imp` |
| `proc` | Logical. Remove process error? All `sd` and `cv` slots in `Stock` and `Fleet` object are set to 0. |
| `grad` | Logical. Remove gradients? All `grad` slots in `Stock` and `qinc` in `Fleet` are set to 0. |
| `silent` | Logical. Display messages? |

## Details

Useful for debugging and testing that MPs perform as expected under perfect conditions.

## Value

An updated object of class `OM`

## Examples

```
OM_noErr <- tinyErr(MSEtool::testOM)
```

`TradePlot` *Generic Trade-Plot Function*

## Description

Generic Trade-Plot Function

## Usage

```
TradePlot(
  MSEobj,
  ...,
  Lims = c(0.2, 0.2, 0.8, 0.8),
  Title = NULL,
  Labels = NULL,
  Satisficed = FALSE,
  Show = "both",
  point.size = 2,
  lab.size = 4,
  axis.title.size = 12,
  axis.text.size = 10,
  legend = TRUE,
  legend.title.size = 12,
  position = c("right", "bottom"),
  cols = NULL,
  fill = "gray80",
  alpha = 0.4,
  PMlist = NULL,
  Refs = NULL,
  Yrs = NULL
)

Tplot(MSEobj, Lims = c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5), ...)

Tplot2(MSEobj, Lims = c(0.2, 0.2, 0.8, 0.8), ...)

Tplot3(MSEobj, Lims = c(0.5, 0.5, 0.8, 0.5), ...)

NOAA_plot2(MSEobj)
```

## Arguments

| | |
|---|---|
| `MSEobj` | An object of class `MSE` |
| `...` | Names of Performance Metrics (PMs), or other arguments to `TradePlot`. First PM is recycled if number of PMs is not even |
| `Lims` | A numeric vector of acceptable risk/minimum probability thresholds. Recycled if not equal to number of PMs. |

| | |
|---|---|
| `Title` | Optional title for each plot. Character vector of `length(PMs)/2`. Recycled. |
| `Labels` | Optional named list specifying new labels for MPs. For example: `Labels = list(AvC="Average Catch", CC1="Constant Catch")` |
| `Satisficed` | Logical. Show only the MPs that meet minimum acceptable thresholds (specified in `Lims`) |
| `Show` | Character. Show the plots ('plots'), results table ('table'), 'both' (default), or invisibly return objects only ('none') |
| `point.size` | Numeric. Size of the MP points |
| `lab.size` | Numeric. Size of MP label. Set to NULL to remove MP labels. |
| `axis.title.size` | Numeric. Size of axis titles |
| `axis.text.size` | Numeric. Size of axis text |
| `legend` | Logical. Include legend? |
| `legend.title.size` | Numeric. Size of legend title text |
| `position` | Character. Position of legend - 'right' or 'bottom' |
| `cols` | Optional character vector of colors for the legend (MP Types) or if `cols` is a character vector of length `MSEobj@nMPs`, then the MP labels are colored (no color legend). |
| `fill` | Character. Color of the fill |
| `alpha` | Numeric. Transparency of fill |
| `PMlist` | Optional list of PM names. Overrides any supplied in ... above |
| `Refs` | An optional named list (matching the PM names) with numeric values to override the default `Ref` values. See examples. |
| `Yrs` | An optional named list (matching the PM names) with numeric values to override the default `Yrs` values. See examples. |

## Value

Invisibly returns a list with summary table of MP performance and the ggplot objects for the plots

## Functions

- `Tplot()`: A trade-off plot showing probabilities that:
  - not overfishing (PNOF) against long-term yield is $> 50\backslash$
  - spawning biomass is below BMSY (P100) against LTY
  - spawning biomass is below 0.5BMSY (P50) against LTY
  - spawning biomass is below 0.1BMSY (P10) against LTY
- `Tplot2()`: A trade-off plot showing probabilities that:
  - short-term yield is $> 50\backslash$

  - spawning biomass is below 0.1BMSY (P10) against average annual variability in yield is < 20\
- `Tplot3()`: A trade-off plot showing probabilities that:
  - not overfishing (PNOF) against long-term yield is > 50\
  - spawning biomass is below 0.1BMSY (P10) against average annual variability in yield is < 20\
- `NOAA_plot2()`: A trade-off plot developed for NOAA showing probabilities that:
  - not overfishing (PNOF) against long-term yield is > 50\
  - spawning biomass is below 0.5BMSY (P50) against average annual variability in yield is < 15\

### Author(s)

A. Hordyk

---

`tune_MP` *Tune MP*

---

### Description

A generic function that uses optimize to tune a single MP parameter to minimize a user-specified function (e.g. squared distance from a mean yield, PGK = 60%, etc.)

### Usage

```
tune_MP(Hist_list, MP, MP_parname, interval, minfunc, tol = 0.01, parallel = F)
```

### Arguments

| | |
|---|---|
| `Hist_list` | A list of objects of class Hist - created by runMSE(..., Hist=T) |
| `MP` | A character string that is the name of the MP to be tuned |
| `MP_parname` | A character string that is the argument (parameter) of the MP to be tuned |
| `interval` | A numeric vector two positions long that is the c(lower.bound, upper.bound) for the parameter to be tuned (MP_parname) |
| `minfunc` | A function to be minimized (e.g. the squared difference between mean yield obtained by the MP and a desired yield) that takes a list of MSE objects as its first argument. |
| `tol` | A positive numerical value that is the tolerance for the optimize procedure (default is 1E-2) |
| `parallel` | Logical: should the MSE projections (over the Hist objects in Hist_list) be calculated in parallel? |

### Value

A function of class MP with argument MP_parname tuned by optim to minimize minfunc

### Author(s)

T. Carruthers

### Examples

```
## Not run:
testOM@cpars$Data = new('Data')
testOM@cpars$Data@MPrec=2000
Hist_1 = runMSE(testOM,Hist=T)
testOM2 = testOM
testOM2@D = testOM@D / 2
Hist_2 = runMSE(testOM2,Hist=T)

myMP = function(x, Data, reps=1, rate = 1){
  CpI = mean(Data@Cat[x,46:50]) / mean(Data@Ind[x,46:50],na.rm=T)
  I = Data@Ind[x,]
  recI = mean(I[length(I)-((5-1):0)])
  Rec=new('Rec')
  Rec@TAC = recI * CpI * rate
  Rec
}
class(myMP) = "MP"

C1000 = function(MSE_list){
  mucat = mean(sapply(MSE_list,function(X){mean(X@Catch)}))
  cat(paste0("mean catch = ",round(mucat,3),"\n"))
  (mucat - 1000)^2 # try to match 1,250t mean yield
}

myMP_t = tune_MP(list(Hist_1,Hist_2), MP = "myMP", MP_parname = "rate",
                 interval = c(1,1.5), minfunc = C1000, tol=1E-3, parallel =F)

formals(myMP_t)$rate

## End(Not run)
```

---

## TuneControl *Settings for Tuning an MP*

---

### Description

Controls for TuneMP(). See the Details of TuneMP() for how they are used.

### Usage

```
TuneControl(
  Interval = c(0.1, 10),
  nGrid = 7,
  Expand = TRUE,
  ExpandFactor = 3,
  MaxExpand = 2,
  nPerRound = NULL,
  MaxIter = 10,
  TolTune = 0.01,
  TolPM = 0.005,
  TolObj = 0.001,
  Direction = c("increasing", "decreasing"),
  nGridConfig = 5,
  nSimConfig = NULL,
  nRefine = 1,
  MaxConfigs = 200,
  ValidateConfigs = TRUE,
  MaxFailRate = 0.05,
  KeepMSE = FALSE,
  DryRun = FALSE
)
```

### Arguments

| | |
|---|---|
| `Interval` | Numeric vector, length 2. Initial range of the tuning argument. Default `c(0.1, 10)`. |
| `nGrid` | Positive integer. Number of log-spaced values in the initial grid. Default `7`. |
| `Expand` | Logical. Extend the range when needed? Default `TRUE`. |
| `ExpandFactor` | Number greater than 1. Factor by which the range is extended. Default `3`. |
| `MaxExpand` | Non-negative integer. Maximum number of extensions. Default `2`. |
| `nPerRound` | `NULL` (default: the number of parallel workers when `parallel = TRUE`, else `1`) or the number of values evaluated per refinement round. |
| `MaxIter` | Positive integer. Maximum number of refinement rounds. Default `10`. |
| `TolTune` | Positive number. Relative width of the tuning-argument bracket at which refinement stops. Default `0.01`. |
| `TolPM` | Non-negative number. Constraint slack tolerance. Default `0.005`. |
| `TolObj` | Positive number. Relative change in the objective at which refinement of an interior optimum stops. Default `0.001`. |
| `Direction` | Character. `'increasing'` (default) if larger values of the tuning argument give higher catches (as for `tunepar`), otherwise `'decreasing'`. Used when `Objective = NULL`. |

| | |
|---|---|
| `nGridConfig` | Positive integer. Number of values of the tuning argument evaluated for each configuration in the first stage. Default `5`. |
| `nSimConfig` | `NULL` (default, all) or the number of simulations used in the first stage. |
| `nRefine` | Positive integer. Number of best configurations tuned fully. Default `1`. |
| `MaxConfigs` | Positive integer. Maximum number of configurations. Default `200`. |
| `ValidateConfigs` | Logical. Apply the MP (with each configuration) to the historical data of the first `Hist` before projecting, to catch invalid arguments? Default `TRUE`. |
| `MaxFailRate` | Number in `[0, 1]`. Candidates whose MP fails in a larger fraction of simulation-management years are infeasible. Default `0.05`. |
| `KeepMSE` | Logical. Keep the mse objects of every evaluation (`@MSE` of the result)? Default `FALSE`. |
| `DryRun` | Logical. Only report the number of projections and an estimate of the run time? Default `FALSE`. |

### Value

A list of class `tunecontrol`.

### See Also

`TuneMP()`

---

## `TuneMP` *Tune a Management Procedure to Meet Management Objectives*

---

### Description

Finds the value of a management procedure's tuning argument (`tunepar` by default) that maximises an objective performance metric (e.g. mean landings) while meeting minimum or maximum thresholds on other performance metrics (e.g. `P(SB > SBMSY) >= 0.6`), evaluated over one or more hist objects. Optionally, first chooses the best values of other MP arguments from a set of candidate configurations.

### Usage

```
TuneMP(
  Hist,
  MP,
  Objective = TuneObjective(PM_Landings),
  Constraints = list(),
  Configs = NULL,
  HistWeights = NULL,
  TuneArg = "tunepar",
  Control = TuneControl(),
```

```
  ValidationHist = NULL,
  parallel = FALSE,
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| `Hist` | A hist object, or a (named) list of them (e.g. a reference set of operating models). |
| `MP` | An MP function (or its name) with a `TuneArg` argument. |
| `Objective` | A `TuneObjective()` object, or `NULL` to maximise the tuning argument subject to the constraints. Default `TuneObjective(PM_Landings)`. |
| `Constraints` | A `TuneConstraint()` object or a list of them. |
| `Configs` | `NULL` (default), or candidate configurations of other MP arguments; see Details. |
| `HistWeights` | `NULL` (default, equal weights) or a positive numeric vector, one per `Hist`, used by metrics with `HistSummary = 'weighted'`. |
| `TuneArg` | Character. Name of the numeric MP argument to tune. Default `'tunepar'`. |
| `Control` | A `TuneControl()` list. |
| `ValidationHist` | `NULL` (default), or a hist object or list of them, used only to report the final MP's performance. |
| `parallel` | Logical. Project candidates in parallel (see `SetupParallel()`)? Default `FALSE`. |
| `silent` | Logical. Suppress progress messages? Default `FALSE`. |

## Details

### Objective and constraints:

The objective and constraints are performance metric functions (see PM, `NewPM()`), specified with `TuneObjective()` and `TuneConstraint()`. When the objective is still increasing at the boundary set by the constraints (e.g. yield increases with `tunepar` until the biomass constraint is breached), the result is the feasible value closest to that boundary. When the objective peaks inside the feasible range, the result is that peak.

With `Objective = NULL`, the tuning argument itself is maximised (or, with `Direction = 'decreasing'` in `TuneControl()`, minimised) subject to the constraints: the classic tuning of an MP to a target.

### Search:

The tuning argument is first evaluated on a grid of `nGrid` log-spaced values over `Interval` (see `TuneControl()`). All grid values are projected in one `Project()` call per `Hist` object, so they run in parallel when `parallel = TRUE`. The range is extended by `ExpandFactor` (up to `MaxExpand` times) when no grid value meets the constraints, or when the objective is still increasing at the end of the range. The best grid value is then refined towards the constraint boundary, evaluating `nPerRound` values per round, until the bracket is

narrower than TolTune (relative), the slack is within TolPM, the objective changes by less than TolObj (relative), or after MaxIter rounds.
The slack of a constraint is (Value - Min) / |Min| and/or (Max - Value) / |Max|. A value is feasible when every slack is at least -TolPM.
Every candidate is evaluated with the same process and observation errors (stored in each Hist), and any random numbers drawn by the MP are seeded from OM@Seed (see Project()), so tuned performance is reproduced exactly by Project().

**Configurations (optional first stage):**
Configs gives candidate values of other MP arguments. It can be a named list of candidate values (all combinations are evaluated; wrap vector-valued candidates in list(), e.g. IndexWeight = list(c(1, 1), c(2, 1))), a list of named lists (each a configuration), or a data frame (one configuration per row; list-columns allowed). Configurations are compared at their own approximately tuned value: each is evaluated on a coarse grid of nGridConfig values of the tuning argument (optionally on the first nSimConfig simulations), its constrained optimum is interpolated, and the configurations are ranked by the objective there. The nRefine best are then tuned fully, and the one with the highest objective is returned. Selecting configurations on the same operating models used to tune them is optimistic; ValidationHist reports the performance of the final MP on other operating models.

**The MP:**
MP must be an MP function (class mp or mmp) with an argument named TuneArg that takes a single positive number; the search is on the log scale. Arguments in Configs must be arguments of MP (unless MP has a ... argument). Candidates are made with SetMPArgs(), so the other arguments of MP keep their defaults.

**Results:**
The tuned MP is @MP, ready for Project(). Its Tuning attribute records the tuned value, the other arguments, the status, and the constraint table. Printing the result shows the tuned value, the configuration, and the value and slack of each metric.
@Status reports how the tuning ended:

- 'boundary': the objective is still increasing where a constraint is breached, and the tuned value is the feasible value closest to that boundary. The binding constraints have Binding = TRUE in the constraint table.
- 'interior': the objective peaks within the feasible range.
- 'at_bound': the objective was still increasing at the end of the searched range after the permitted extensions. Widen Interval or increase MaxExpand in TuneControl().
- 'infeasible': no evaluated value met every constraint. The value closest to meeting them (largest slack) is returned; the constraints may be unattainable for this MP.

TuneTable() returns the constraint table, the ranking of configurations, every evaluated candidate, and the validation results. The full mse objects of every evaluation are kept with KeepMSE = TRUE in TuneControl(). With DryRun = TRUE, TuneMP() instead returns (invisibly) a list with the number of projections in each stage and an estimate of the run time.

## Value

A tunemp object (invisibly). The tuned MP is @MP. See Results in Details.

### See Also

TuneObjective(), TuneConstraint(), TuneControl(), TuneTable(), SetMPArgs(), PM

### Examples

```
## Not run:
Hist <- Simulate(SingleStockOM)

# tune to a target: the largest tunepar with P(SB > SBMSY) >= 0.6
Tuned <- TuneMP(Hist, IndexTarget, Objective = NULL,
                Constraints = TuneConstraint(PM_SBSBMSY, Min = 0.6))

# maximise landings subject to constraints, choosing among configurations
Tuned <- TuneMP(Hist, IndexTarget,
                Objective   = TuneObjective(PM_Landings),
                Constraints = list(TuneConstraint(PM_SBSBMSY, Min = 0.6),
                                   TuneConstraint(PM_Safety, Lim = 0.4, Min = 0.9)),
                Configs     = list(Smooth = c(TRUE, FALSE), RecentYears = 1:2))
Tuned
TuneTable(Tuned, 'Configs')
MSE <- Project(Hist, MPs = list(IT_Tuned = Tuned@MP))

# check the run time first
TuneMP(Hist, IndexTarget, Constraints = TuneConstraint(PM_SBSBMSY, Min = 0.6),
       Control = TuneControl(DryRun = TRUE))

# tune over a reference set, validate on a robustness set
Tuned <- TuneMP(list(Base = HistBase, LowM = HistLowM), IndexTarget,
                Constraints    = TuneConstraint(PM_SBSBMSY, Min = 0.6, HistSummary = 'worst'),
                ValidationHist = list(HighSteep = HistHighSteep))
TuneTable(Tuned, 'Validation')

## End(Not run)
```

---

## tunemp-class *The* tunemp *S4 Class*

---

### Description

The result of TuneMP().

### Usage

```
## S4 method for signature 'tunemp'
show(object)
```

### Arguments

| | |
|---|---|
| object | A tunemp object. |

### Slots

`MP` The tuned management procedure: `MP` with the chosen configuration and tuning argument set (see `SetMPArgs()`), and a `Tuning` attribute summarising the tuning.

`MPName` Character. Name of the MP that was tuned.

`Args` Named list of the argument values set in `MP`.

`Status` Character. How the tuning ended: `'boundary'` (at the constraint boundary), `'interior'` (at a peak of the objective within the constraints), `'at_bound'` (the objective was still increasing at the end of the searched range), or `'infeasible'` (no value met the constraints; the value closest to meeting them is returned).

`Objective` Numeric. Objective value of the tuned MP (`NA` without an objective).

`Constraints` `data.frame` of each metric's value, thresholds, slack, and whether the constraint is binding, for the tuned MP.

`Configs` `data.frame` ranking the configurations of the first stage (empty without `Configs`).

`History` `data.frame` of every evaluated candidate.

`Validation` `data.frame` of the tuned MP's metrics for `ValidationHist` (empty without it). See `TuneTable()` for the columns of these four tables.

`MSE` List of the mse objects of every evaluation, with `KeepMSE = TRUE` in `TuneControl()`.

`Settings` List of the metrics, `TuneControl()` settings, `HistWeights`, and `TuneArg`.

### See Also

`TuneMP()`, `TuneTable()`

---

## `TuneObjective` *Specify the Objective and Constraints for Tuning an MP*

---

### Description

`TuneObjective()` specifies the performance metric maximised by `TuneMP()`; `TuneConstraint()` a performance metric that must be at least `Min` and/or at most `Max`.

### Usage

```
TuneObjective(
  PM,
  ...,
  Value = "Mean",
  Stocks = NULL,
  StockSummary = c("sum", "mean", "min", "max"),
  HistSummary = c("weighted", "pooled", "min", "max"),
  Hists = NULL,
  Name = NULL
)
```

```
TuneConstraint(
  PM,
  ...,
  Min = NULL,
  Max = NULL,
  Value = "Mean",
  Stocks = NULL,
  StockSummary = c("worst", "mean", "sum", "min", "max"),
  HistSummary = c("weighted", "pooled", "worst", "min", "max"),
  Hists = NULL,
  Name = NULL
)
```

### Arguments

| | |
|---|---|
| `PM` | A performance metric function (class `pm`, see PM and `NewPM()`), or its name. |
| `...` | Arguments passed to `PM`, e.g. `Ref`, `Years`, `Lim`. |
| `Value` | `'Mean'` (default; `pm@Mean`), `'Stat'` (mean over simulations of `pm@Stat`), or a function of the `pm` object returning a Stock x MP matrix. |
| `Stocks` | `NULL` (default, all) or the names of the stocks (complexes) to use. |
| `StockSummary` | Character. How the metric is combined across stocks. |
| `HistSummary` | Character. How the metric is combined across `Hist` objects. |
| `Hists` | `NULL` (default, all) or the names or positions of the `Hist` objects the metric applies to. |
| `Name` | Character. Label of the metric. Default is the name of `PM`. |
| `Min`, `Max` | `NULL` or a number: minimum and maximum allowed values. At least one is needed. |

### Details

A metric is calculated for each `Hist` object and stock (or complex, see PM) as `Value` of the pm object returned by `PM`, then combined across stocks (`StockSummary`) and across `Hist` objects (`HistSummary`). `'worst'` requires the constraint to hold for every stock (or `Hist`) separately; `'weighted'` averages with `HistWeights`; `'pooled'` averages with weights proportional to the number of simulations (equivalent to pooling the simulations of all `Hist` objects).

### Value

A list of class `tunemetric`.

### See Also

`TuneMP()`, PM

### Examples

```
TuneObjective(PM_Landings, Years = 2031:2050)
TuneConstraint(PM_SBSBMSY, Min = 0.6)
TuneConstraint(PM_AAVY, Value = 'Mean', Max = 0.2)
```

---

## TuneTable *Tables from a Tuned MP*

---

### Description

Returns one of the tables stored in a tunemp object: the performance of the tuned MP, the ranking of candidate configurations, the record of every evaluated candidate, or the performance of the tuned MP on the validation operating models.

### Usage

```
TuneTable(x, What = c("Constraints", "Configs", "History", "Validation"))
```

### Arguments

| | |
|---|---|
| `x` | A tunemp object. |
| `What` | Character. `'Constraints'` (default; the tuned MP's metric values), `'Configs'` (the ranking of configurations), `'History'` (every evaluated candidate), or `'Validation'` (the tuned MP's metric values for `ValidationHist`). See Details. |

### Details

Metric values are combined across stocks and `Hist` objects as specified in `TuneObjective()` and `TuneConstraint()`. With `'worst'`, the reported value is that of the stock or `Hist` with the smallest slack. The slack of a constraint is `(Value - Min) / |Min|` and/or `(Max - Value) / |Max|` (the smaller of the two); a negative slack means the constraint is not met.

`'Constraints'`:

One row per metric (the objective first, then each constraint), for the tuned MP. Also printed by `show()`.

- `Name`: name of the metric (`Name` in `TuneObjective()` or `TuneConstraint()`).
- `Type`: `'objective'` or `'constraint'`.
- `Value`: value of the metric.
- `Min`, `Max`: thresholds of the constraint (`NA` when not set).
- `Slack`: slack of the constraint (`NA` for the objective).
- `Binding`: `TRUE` for constraints with an absolute slack no greater than `max(TolPM, 0.01)` (see `TuneControl()`), i.e. the constraints that limit the tuned value.

`'Configs'`:

One row per configuration in `Configs` (an empty data frame without `Configs`, or with a single configuration). `Value` to `Slack` are from the first stage, the `Tuned` columns from the second stage.

- `Config`: index of the configuration. Matches `Config` in `'History'`.
- `Label`: the configuration's argument values, e.g. `'Smooth = TRUE, RecentYears = 2'`.
- `Value`: approximately tuned value of the tuning argument. When the constraint boundary lies between two grid values, `Value` is interpolated between them.
- `Objective`: objective at `Value` (interpolated in the same way; `NA` when infeasible).
- `Feasible`: did any grid value meet every constraint?
- `Status`: `'boundary'`, `'interior'`, `'at_bound'`, or `'infeasible'` (see the `Status` slot of tunemp).
- `Slack`: largest slack across the grid values (`Inf` without constraints).
- `TunedValue`, `TunedObjective`, `TunedStatus`: the tuned value, objective, and status of the `nRefine` best configurations that were tuned fully (`NA` for the others).

`'History'`:

One row per evaluated combination of configuration and tuning-argument value, in the order they were evaluated.

- `Config`: index of the configuration (`1` without `Configs`).
- `x`: value of the tuning argument.
- `Objective`: value of the objective (`NA` without an objective, or when the candidate failed).
- `Slack`: smallest slack across the constraints (`Inf` without constraints, `-Inf` when the candidate failed).
- `Feasible`: does the candidate meet every constraint (within `TolPM`)?
- `FailRate`: largest fraction, across the `Hist` objects, of simulation-management years in which the MP returned an error (`1` when the projection failed). Candidates with a `FailRate` greater than `MaxFailRate` (see `TuneControl()`) fail.
- One column per metric, named by the metric, with its value.
- `Stage`: `'Configure'` (first stage, on the first `nSimConfig` simulations when set) or `'Tune'` (second stage, all simulations).

`'Validation'`:

One row per validation `Hist` object and metric (an empty data frame without `ValidationHist` in `TuneMP()`). Metrics are calculated for each `Hist` separately and combined across stocks only.

- `Hist`: name of the validation `Hist` object (`'Validation1'`, ... when unnamed).
- `Name`, `Type`, `Value`, `Min`, `Max`: as for `'Constraints'`.
- `Met`: is the constraint met (without tolerance)? `NA` for the objective.

## Value

A `data.frame`; see Details for the columns.

### See Also

TuneMP(), tunemp

### Examples

```
## Not run:
Tuned <- TuneMP(Hist, IndexTarget,
                Constraints = TuneConstraint(PM_SBSBMSY, Min = 0.6),
                Configs     = list(RecentYears = 1:3))
TuneTable(Tuned)

# configurations, best first
Cfg <- TuneTable(Tuned, 'Configs')
Cfg[order(-Cfg$Objective), ]

# objective against the tuning argument for the chosen configuration
Evals <- TuneTable(Tuned, 'History')
Evals <- Evals[Evals$Stage == 'Tune', ]
plot(Objective ~ x, data = Evals, log = 'x', col = ifelse(Evals$Feasible, 1, 2))

## End(Not run)
```

---

## Turing *Turing Test*

---

### Description

Plots the available data in the Data object together with 5 samples of historical data from the Operating Model (OM) in a random order. The test is used to determine if the data generated by the OM is similar to the fishery data in the Data object. In a well specified OM the user should not be able to visually identify which of the 6 plots is the real fishery data and which are generated by the OM.'

### Usage

```
Turing(OM, Data, wait = TRUE)

TuringMOM(multiHist, Data, wait = TRUE)
```

### Arguments

| | |
|---|---|
| OM | An object of class OM or class multiHist |
| Data | An object of class Data or a nested list of Data objects for each stock and fleet |
| wait | Logical. Wait for key press before next plot? |
| multiHist | An object of class multiHist. The output of SimulateMOM |

### Details

In its current form the Turing function does not interpolate missing data in the Data object. Therefore if there are years with missing data, say in the catch time-series, it will be obvious which are the real data and which have been generated by the model. Future versions of the function may include methods to impute missing data for plotting purposes.

The question to ask when examining the plots produced by `Turing`: do the plots of the 6 data samples look like they are all samples from the same underlying distribution?

### Functions

- `TuringMOM()`: Turing function for multi-stock, multi-fleet MOMs

### Note

The Turing function was suggested by Andre Punt in his review of one of our recent projects. It is named after the Turing test, developed by Alan Turing in 1950, which is designed to see if a human can detect the difference between human and machine generated information.

### Examples

```
## Not run:
Turing(MSEtool::testOM, MSEtool::SimulatedData, wait=FALSE)

## End(Not run)
```

---

## `TwoFleetOM` *Two-Fleet Example Operating Model*

---

### Description

An example om object representing a single stock exploited by two fleets with contrasting selectivity patterns and monitoring intensities. Intended to illustrate multi-fleet OM construction in `MSEtool`. Pairings of stocks, fleets, and observation models are for illustration only and should not be taken as ecologically meaningful. See also SingleStockOM, ComplexOM, and MultiStockOM for structural variants.

### Usage

```
TwoFleetOM
```

### Format

An om object with the following slots populated:

- `Name`: `"Single Stock - Two Fleets"`.
- `nSim`: `8`. Number of stochastic simulations.
- `nYear`: `20`. Number of historical years.

- `pYear`: `30`. Number of projection years.
- `Stock`: AlbacoreExStock. A single stock object.
- `Fleet`: a length-1 list containing a length-2 list of `[[AsympExFleet, DomeExFleet]]`. Structure is `[[stock]][[fleet]]`.
- `Obs`: a length-1 list containing a length-2 list of `[[AgeStructuredObs, CatchAndSurveyObs]]`. Structure is `[[stock]][[fleet]]`. Fleet 1 has a data-moderate age-structured monitoring programme; Fleet 2 has catch and a survey only.
- `Imp`: a length-1 list containing a length-2 list of FullComplianceImp. Structure is `[[stock]][[fleet]]`. Perfect TAC/ effort implementation and full immediate size-regulation compliance.

## Details

**Stock:**
AlbacoreExStock represents a long-lived, slow-growing large pelagic species with low natural mortality. See AlbacoreExStock for full parameter details.

**Fleets:**
Two contrasting fleets are included:

- AsympExFleet: asymptotic length-based selectivity; effort stabilises early in the historical period.
- DomeExFleet: dome-shaped selectivity; effort increases gradually across the historical period.

See AsympExFleet and DomeExFleet for full parameter details.

**Observation Models:**
Each fleet has its own observation model, illustrating a contrast between a well-monitored commercial fleet and a less-monitored one:

- Fleet 1 (AsympExFleet) uses AgeStructuredObs: landings, a spawning-biomass survey, and annual age-composition samples from landed and discarded fish.
- Fleet 2 (DomeExFleet) uses CatchAndSurveyObs: reported landings and a biomass-proportional survey index only, with no composition data.

**Implementation Error:**
FullComplianceImp models perfect implementation: `TAC@Mean = 1/ SD = 0` and `Effort@Mean = 1/SD = 0` mean realised TAC/effort always equal advice exactly, and `Size@Compliance = 1` means any size-regulation advice (a change to `Advice@Retention/Advice@Selectivity`) is adopted immediately by the whole fleet. See `Imp()` for details, and OverageImp, UnderageImp, PartialSizeComplianceImp for contrasting examples.

## See Also

SingleStockOM, ComplexOM, MultiStockOM for structural variants. AlbacoreExStock, AsympExFleet, DomeExFleet, AgeStructuredObs, CatchAndSurveyObs, FullComplianceImp for the component objects. `OM()`, om, `runMSE()`, `PopulateOM()`, `ExampleMPs()`

Other om: `CatchFrac()`, `ComplexOM`, `DataLag()`, `EffortAllocation()`, `Herm()`, `HermOM`, `HistoricalWeight()`, `InterimAdvice()`, `Interval()`, `MPStartYear()`, `MultiStockOM`, `OM()`, `OM-accessors`, `RefSeason()`, `SeasonalSpatialOM`, `SingleStockOM`, `om-class`, `stocktransition-class`

### Examples

```
TwoFleetOM
nFleet(TwoFleetOM)

## Not run:
Hist <- runMSE(TwoFleetOM, MPs = ExampleMPs())

## End(Not run)
```

---

`UnderageImp` *Underage Implementation Error*

---

### Description

An imp object representing a fleet that consistently realises less catch and effort than advised – e.g. quota left unfilled because the fishery isn't profitable enough to fully prosecute, or an effort target that isn't reached due to weather, capacity, or market constraints.

### Usage

```
UnderageImp
```

### Format

An imp object. Populated slots:

- `TAC` (impslot): `Mean` in `[0.7, 0.9]`, `SD` in `[0.05, 0.15]` – realised catch is on average 10-30% below the advised TAC, with additional simulation-to-simulation and year-to-year stochastic variation.
- `Effort` (impslot): same `Mean`/`SD` ranges, applied to effort-based advice.

Empty slots: `Size` (this example isolates TAC/effort underage; see PartialSizeComplianceImp for a size-regulation-focused example).

### See Also

imp, `Imp()`, `ImpSlot()`, FullComplianceImp, OverageImp, PartialSizeComplianceImp

Other imp: `FullComplianceImp`, `Imp()`, `ImpSlot()`, `OverageImp`, `PartialSizeComplianceImp`, `imp-class`, `impslot-class`

### Examples

```
UnderageImp
slot(UnderageImp, "TAC")
```

---

`unfished-class` `unfished` *Class*

---

## Description

Stores unfished equilibrium and dynamic population states, used as the baseline reference for calculating biological reference points.

## Usage

```
Unfished(object)

Equilibrium(unfished)

Dynamic(unfished)
```

## Arguments

| | |
|---|---|
| `object` | A `hist` or `mse` object. |
| `unfished` | An `unfished` object. |

## Details

Three accessor functions are provided:

- `Unfished()` extracts the `unfished` object from a `hist` or `mse` object
- `Equilibrium()` extracts the equilibrium population dynamics
- `Dynamic()` extracts the dynamic population dynamics

## Value

- `Unfished()` returns an `unfished` object
- `Equilibrium()` returns a popdynamics object
- `Dynamic()` returns a popdynamics object

## Slots

`Equilibrium` Unfished equilibrium population dynamics. A popdynamics object.

`Dynamic` Unfished dynamic population dynamics. A popdynamics object.

`Misc` Miscellaneous list for additional outputs.

## See Also

popdynamics

### Examples

```
## Not run:
unf  <- Unfished(Hist)
eq   <- Equilibrium(unf)
dyn  <- Dynamic(unf)

## End(Not run)
```

---

`unit-conversions` *Unit Conversion Functions*

---

### Description

A family of simple unit conversion functions for weight and length measurements commonly used in fisheries stock assessments.

### Usage

```
lb2kg(x)

lb2mt(x)

kg2lb(x)

kg2mt(x)

kg2_1000lb(x)

inch2mm(x)

mm2inch(x)
```

### Arguments

| | |
|---|---|
| `x` | Numeric vector of values to convert. |

### Details

**Weight:**

- `lb2kg(x)`: pounds → kilograms
- `lb2mt(x)`: pounds → metric tonnes
- `kg2lb(x)`: kilograms → pounds
- `kg2mt(x)`: kilograms → metric tonnes
- `kg2_1000lb(x)`: kilograms → thousands of pounds

**Length:**

- `inch2mm(x)`: inches → millimetres
- `mm2inch(x)`: millimetres → inches

### Value

Numeric vector of the same length as `x` in the target units.

---

`UpdateObject` *Update an S4 Object and Its Slots*

---

### Description

Recursively updates an openMSE S4 object to ensure all slots are valid, and optionally replaces specified slots with new values. `UpdateSlots()` is called first to reset any inaccessible or corrupted slots to their class defaults, recursing into nested S4 objects. Named arguments in `...` are then used to replace slots by name.

### Usage

```
UpdateObject(object, ...)

UpdateSlots(object)
```

### Arguments

| | |
|---|---|
| `object` | An S4 object. |
| `...` | Named objects used to replace existing slots. Names must match slot names in `object`. Unrecognised names emit a warning and are ignored. |

### Value

The updated S4 object with all slots valid and any supplied replacements applied.

### Examples

```
## Not run:
obj <- UpdateObject(obj, Fleet=newFleet)

## End(Not run)
```

---

`Uses` *Find the Management Procedures that use a particular data slot*

---

## Description

Find the Management Procedures that use a particular data slot

## Usage

```
Uses(slot, silent = FALSE)
```

## Arguments

| | |
|---|---|
| `slot` | A slot from an object of class `Data`. Character string. |
| `silent` | Logical. Should messages be printed? |

## Value

A character string of MPs that use the slot.

## Author(s)

A. Hordyk

## Examples

```
Uses("Mort")
```

---

`validcpars` *Valid custom parameters (cpars)*

---

## Description

Valid custom parameters (cpars)

## Usage

```
validcpars(
  type = c("all", "Stock", "Fleet", "Obs", "Imp", "internal"),
  valid = TRUE,
  show = TRUE
)
```

### Arguments

| | |
|---|---|
| type | What cpars to show? 'all', 'Stock', 'Fleet', 'Obs', 'Imp', or 'internal' |
| valid | Logical. Show valid cpars? |
| show | Logical. Display the table in the Viewer? |

### Value

a HTML datatable with variable name, description and type of valid cpars

### Control list

A named list for `control`, for example, `OM@cpars$control <- list(TAC = "removals", CAL = "removals")`, can be specified to override default settings in the MSE simulation. Possible names in the `control` list are:

- `TAC` Character, set to `"removals"` so that the TAC is applied to the sum of retained + discarded catch. Default only applies the TAC to the retained catch.
- `CAL` Character, set to `"removals"` to sample the catch-at-length from retained + discarded catch. Default only samples from retained catch.
- `D` Character, set to `"VB"` so that historical depletion `OM@D` corresponds to vulnerable biomass depletion (only used when `OM@cpars$qs = NULL`).
- `optVB` Logical, set to `TRUE` so that historical depletion `OM@D` corresponds to vulnerable biomass depletion. Default sets depletion according to spawning biomasss when `OM@cpars$qs = NULL`.
- `optSBMSY` Logical, set to `TRUE` such that `OM@D` corresponds to the ratio of spawning biomass to MSY. Default uses according to spawning biomass depletion (biomass relative to unfished levels).
- `Depletion` Character, set to `"end"` such that historical depletion `OM@D` corresponds to the biomass at the end of the last projection year. Default corresponds to the value at the beginning of the last projection year.
- `ntrials` Integer, set the number of iterations to sample the operating model to match the depletion to `OM@D`. Default is 50.
- `fracD` Numeric, the maximum allowable proportion of simulations allowed to hit the bounds of the depletion parameter (simulation returns an error if exceeded). Default is 0.05.
- `checks` Logical. If `TRUE`, plots depletion and SB/SBMSY figures and prints values to the R console to diagnose issues with operating model configuration with regards to depletion.
- `unfished` Logical. If `TRUE`, returns historical simulations with F = 0.
- `progress` Logical. If `TRUE`, updates progress bar through `shiny::incProgress`. Used in conjunction with Shiny apps.
- `maxiterF` Integer, the number of iterations to solve for F in the projections from the specified TAC. Default is 300.
- `tolF` Numeric, the tolerance for the catch relative to the TAC when solving for F in the projections. Default is 1e-4.

- `HZN` Integer, the number of generations to solve for B_low. Default is 2. See `getBlow()`.
- `Bfrac` Numeric, proportion of SBMSY to solve for B_low. Default is 0.5. See `getBlow()`.
- `skipdata` Logical. If `TRUE`, skips conditioning on data in `MOM@cpars[[p]][[f]]$Data`. Only used in `multiMSE()`.
- `HermEq` Logical, whether the equilibrium population age structures in the multi-OM is generated from the hermaphroditism vector (intended for use in salmonMSE). Default is TRUE. Only used in `multiMSE()`.
- `HistRel` Logical, whether to perform the historical reconstruction with inter-stock relationships in `MOM@Rel`. Default is TRUE. Only used in `multiMSE()`.

### Examples

```
## Not run:
validcpars() # all valid cpars

validcpars("Obs", FALSE) # invalid Obs cpars

## End(Not run)
```

---

## `ValidUnits` *Return Valid Units*

---

### Description

Returns the valid units for `Ages()`, `Length()`, and `Weight()` objects.

### Usage

```
ValidUnits(x = "ages")
```

### Arguments

| | |
|---|---|
| `x` | An ages, length, or weight object, or a character string specifying the class: one of `"ages"`, `"length"`, or `"weight"` (case-insensitive). If `NULL`, returns a named list of valid units for all three classes. Default `"ages"`. |

### Value

- If `x` is `NULL`: a named list with elements `Age`, `Length`, and `Weight`, each containing a character vector of valid unit strings.
- If `x` is an ages object or `"ages"`: a character vector of valid age units: `"year"`, `"half-year"`, `"quarter"`, `"month"`, `"week"`.
- If `x` is a length object or `"length"`: a character vector of valid length units: `"mm"`, `"cm"`, `"inch"`.
- If `x` is a weight object or `"weight"`: a character vector of valid weight units: `"g"`, `"kg"`, `"lb"`.
- If no match is found: the string `"No units found"`.

## See Also

Ages(), Length(), Weight()

## Examples

```
ValidUnits()           # age units (default)
ValidUnits("length")   # length units
ValidUnits("weight")   # weight units
ValidUnits(NULL)       # all units as a named list
ValidUnits(Ages())     # age units from an ages-class object
```

---

# VOI *Calculate Value Of Information*

---

## Description

A function that relates operating model parameters and parameters of the observation model to yield (by default). A user can also specific their own utility values (Ut) which is arranged in a matrix of nsim rows and nMP columns.

## Usage

```
VOI(
  MSEobj,
  ncomp = 6,
  nbins = 8,
  maxrow = 8,
  Ut = NA,
  Utnam = "Utility",
  plot = TRUE
)
```

## Arguments

| | |
|---|---|
| MSEobj | An object of class MSE |
| ncomp | Maximum number of variables to examine per MP |
| nbins | Number of percentile bins for sampled parameters of the operating model or observation model, which is used for calculating variability in utility across the sampled range of each parameter |
| maxrow | maximum number of MPs per plot |
| Ut | A matrix of user-specified utility values of nsim rows and nMPs columns |
| Utnam | The name of the utility measure for plotting |
| plot | Logical. Show the plot? |

## Author(s)

T. Carruthers

---

`VOI2` *Calculate Value Of Information 2*

---

## Description

A function that relates operating model parameters and parameters of the observation model to relative yield (yield over last 5 years of projection relative to a 'best F' scenario that maximizes yield).

## Usage

```
VOI2(MSEobj, ncomp = 6, nbins = 4, Ut = NA, Utnam = "yield", lay = F)
```

## Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `ncomp` | Maximum number of observation variables to examine per MP |
| `nbins` | Number of bins for sampled observation variables used for calculating variability in utility across the sampled range of each parameter |
| `Ut` | A matrix of user-specified utility values of nsim rows and nMPs columns |
| `Utnam` | The name of the utility measure for plotting |
| `lay` | Controls whether labels are in lay terms or not |

## Note

VOI2 assumes that relative cost for each type of improvement in data is linearly related to the number of samples (e.g. nCAAobs) or square function of improved precision and bias e.g.: relative cost= 1/(newCV/oldCV)^2

## Author(s)

T. Carruthers

---

`VOIplot` *Yet another Value of Information Plot*

---

## Description

A function that relates parameters of the observation model and the operating model parameters to yield.

### Usage

```
VOIplot(
  MSEobj,
  MPs = NA,
  nvars = 5,
  nMP = 4,
  Par = c("Obs", "OM"),
  YVar = c("Y", "B"),
  doPlot = TRUE,
  incStat = FALSE,
  availMP = NULL,
  acceptMP = NULL,
  incNames = TRUE,
  labcex = 0.8,
  quants = c(0.05, 0.95)
)
```

### Arguments

| | |
|---|---|
| `MSEobj` | An object of class MSE |
| `MPs` | The MPs to plot. If NA it will plot the first nMP from MSEobj |
| `nvars` | The number of observation or operating model parameters to plot (number of columns) |
| `nMP` | The maximum number of MPs to plot (number of rows) |
| `Par` | Plot Operating Model (OM) or Observation (Obs) parameters? |
| `YVar` | Variable for Y-Axis: Yield (Y) or Biomass (B) (relative to BMSY) |
| `doPlot` | Output the plot? |
| `incStat` | Include a print out of statistic describing the curviness of the line? |
| `availMP` | Optional character string of MPs that are available. These names are colored black |
| `acceptMP` | Optional character string of MPs that are acceptable. These names are colored green if they are also in availMP |
| `incNames` | Include the names? |
| `labcex` | Character size of the label |
| `quants` | Quantiles to calculate |

### Value

A list of all the information included in the plot

### Author(s)

A. Hordyk

`Weight` *Weight Constructor and Accessors*

## Description

Construct a weight object defining the weight-at-age and weight-at-length schedules for a stock, or access and replace the `Weight` slot of a stock and its individual slots. A `Weight` object is required for all stock objects.

## Usage

```
Weight(
  Pars = list(),
  Model = NULL,
  Units = "kg",
  MeanAtAge = NULL,
  MeanAtLength = NULL,
  CVatAge = NULL,
  Dist = "lognormal",
  TruncSD = 2,
  Timing = 0,
  Random = NULL,
  AWK = NULL,
  Classes = NULL,
  Misc = list()
)

Weight(x) <- value

AWK(x)

AWK(x) <- value
```

## Arguments

| | |
|---|---|
| `Pars` | `list`. Named list of allometric parameters. Element names must match the arguments of a built-in weight model (see `WeightModels()`). When `Pars` is non-empty and `Model` is `NULL`, the model is inferred automatically from the parameter names. When `Pars` is a stock object, `Weight()` acts as a pass-through accessor and returns `x@Weight`. See *Specifying Weight-at-Age* and Specifying Biological and Fleet Schedules for the full set of accepted input formats. Default `list()`. |
| `Model` | `character(1)` or `function`. Weight model identifier. When `NULL` (default), the model is inferred from `Pars`. May be set to a character string naming a built-in model or to a custom R function — see Specifying Biological and Fleet Schedules. |

`Units` — `character(1)`. Physical unit of weight measurements. Must be one of the strings returned by `ValidUnits()` (e.g., `"g"`, `"kg"`). Default `"kg"`.

`MeanAtAge` — `array` or `NULL`. Mean weight at age with named dimensions `Sim × Age × Year`. Supply directly when bypassing the model-based approach (`Pars = list()`). Only the years at which values *change* need to be included; `Extend()` fills the remainder. When `Pars` contains a matched model, any values here are **overwritten** during `Populate()`. A numeric vector of length `nAge` is also accepted and is converted to a `1 × nAge × 1` array automatically. Default `NULL`.

`MeanAtLength` — `array` or `NULL`. Mean weight at length with named dimensions `Sim × Class × Year`. Populated automatically during `Populate()` when an at-length weight model is used (see *At-Length Models*); may also be supplied directly. When `MeanAtLength` is populated and `MeanAtAge` is not, `Populate()` converts it to `MeanAtAge` via the age-length key (`ALK`) from a length object. A numeric vector is also accepted and converted to a `1 × nClass × 1` array automatically, using `Classes` if supplied or the length object's own `Classes` otherwise; if `Classes` differs from the length object's, the `ALK` is recalculated on `Classes`. Default `NULL`.

`CVatAge` — `numeric(1)`, length-2 bounds vector, `array`, or `NULL`. Coefficient of variation of weight-at-age, used to parameterise the within-age-class weight distribution and to build the age-weight key (`AWK`). A length-2 vector is treated as `Uniform(lower, upper)` bounds sampled once per simulation. When `NULL` (default), no `AWK` is constructed; for most models weight variability is propagated through the `ALK` and a length-weight relationship rather than through `CVatAge` directly.

`Dist` — `character(1)`. Parametric distribution for weight-at-age variability. Options are `"lognormal"` (default) and `"normal"`.

`TruncSD` — `numeric(1)`. Number of standard deviations at which the weight distribution is truncated. Default `2`.

`Timing` — `numeric(1)`. Fractional position within the time step at which weight is measured: `0` = start of step, `1` = end of step. Default `0`.

`Random` — `array` or `NULL`. Simulation- and year-specific random multipliers applied to weight parameters. Currently stored but not applied. Default `NULL`.

`AWK` — `array` or `NULL`. Age-weight key giving the probability that an individual of a given age belongs to each weight class (`Sim × Age × Class × Year`). Only populated during `Populate()` when `CVatAge` is non-`NULL`; supply directly only when overriding the computed key. Default `NULL`.

`Classes` — `numeric` or `NULL`. Weight class midpoints in units of `Units`. Populated automatically during `Populate()` when `CVatAge` is non-`NULL` and `Classes` is `NULL`. Default `NULL`.

`Misc` — `list`. Used internally. Default `list()`.

`x` — A weight object for slot accessors, or a stock object for `Weight<-`.

`value` — For `Weight<-`: a weight object. For `AWK<-`: a replacement age-weight key array.

## Details

### Specifying Weight-at-Age:

There are two ways to define the weight schedule:

**Model-based** (recommended): supply `Pars` with named parameters matching a built-in model (see `WeightModels()`). If `Model = NULL` and the parameter names uniquely match a model, the model is resolved automatically. `MeanAtAge` is then populated by `Populate()` when the stock is added to an `OM()`:

```
# Power-law length-weight - model inferred from Pars names
wt <- Weight(Pars = list(a = 0.01, b = 3.0))

# Stochastic a across simulations
wt <- Weight(Pars = list(a = c(0.008, 0.012), b = 3.0))

# Time-varying a (increases in 2010; Extend() fills the rest)
a_arr <- array(c(0.01, 0.012), dim = c(1, 2),
               dimnames = list(Sim = 1, Year = c(1990, 2010)))
wt <- Weight(Pars = list(a = a_arr, b = 3.0))

# Inter-annual random walk on a (log-normal, SD = 0.1)
wt <- Weight(Pars = list(a = 0.01, aSD = 0.1, b = 3.0))
```

**Direct array**: supply `MeanAtAge` (or `MeanAtLength`) with `Pars = list()`. Values are preserved during `Populate()`:

```
ages  <- 0:20
wt_aa <- Weight(MeanAtAge = 0.01 * (100 * (1 - exp(-0.2 * ages)))^3)
```

See Specifying Biological and Fleet Schedules for full details on input formats and the rules that govern how `Pars`, `Model`, and `MeanAt*` arrays interact.

### At-Length Weight Models:

Some built-in weight models (those whose class contains `"at-Length"`) take a length object as input and produce `MeanAtLength` rather than `MeanAtAge` directly. In this case a populated `Length()` object must be passed to `Populate()`. The resulting `MeanAtLength` is then converted to `MeanAtAge` automatically via the age-length key (`ALK`). The same conversion applies when `MeanAtLength` is supplied directly by the user.

### The Age-Weight Key:

The `AWK` is only constructed when `CVatAge` is non-`NULL`. For most operating models weight variability is captured through the `ALK` and a length-weight relationship, so `CVatAge` is typically left at its default of `NULL`.

### Pass-Through Access from a Stock:

When `Pars` is a stock object, `Weight()` returns the `Weight` slot directly:

```
Weight(my_stock)           # returns my_stock@Weight
Weight(my_stock) <- my_wt  # replaces my_stock@Weight
```

**Slot Accessors:**

Individual slots can be read or replaced using generic functions matching their names. All replacement functions re-validate the object:

```
Pars(wt)          <- list(a = 0.01, b = 3.0)
Model(wt)         <- "LengthWeight"
Units(wt)         <- "kg"
MeanAtAge(wt)     <- my_array
MeanAtLength(wt)  <- my_length_array
CVatAge(wt)       <- 0.1
Dist(wt)          <- "lognormal"
TruncSD(wt)       <- 3
Timing(wt)        <- 0
AWK(wt)           <- my_awk
Classes(wt)       <- seq(0, 5, by = 0.5)
```

## Value

- `Weight()` returns a weight object. If `Pars` is a stock, returns `x@Weight`.
- `Weight<-` returns the stock `x` with the `Weight` slot replaced and the object re-validated.
- `AWK()` returns the age-weight key array from `x`.
- `AWK<-` returns `x` with the `AWK` slot updated and the object re-validated.

## See Also

- weight for the class definition and slot-level documentation.
- `WeightModels()` for available weight models and required parameter sets.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `Stock()` for the enclosing stock constructor.
- `Length()` for the companion length schedule, required when using at-length weight models.

Other weight: `weight-class`

## Examples

```
# See WeightModels() for all available built-in weight models and the
# parameter names required for each.
WeightModels()

# ---- Model-based specification ----

## Constant parameters (scalar) - same value for every simulation and year
wt <- Weight(Pars = list(a = 0.01, b = 3.0))

## Stochastic across simulations - a drawn from Uniform(lower, upper)
## once per simulation; b fixed in all simulations
wt <- Weight(Pars = list(a = c(0.008, 0.012), b = 3.0))
```

```
## Inter-annual random walk on a - log-normal with 10% CV
## aSD is removed from Pars after use
wt <- Weight(Pars = list(a    = c(0.008, 0.012),
                         aSD  = 0.1,
                         b    = 3.0))

## Time-varying parameters with named change-point arrays
## Only the years where values change need to be supplied; Extend() fills
## all intermediate and future years by forward-filling automatically.
nSim <- 48

## a increases in 2010 (two change points: 1990 and 2010)
a_arr <- array(
  c(rep(0.01, nSim), rep(0.012, nSim)),
  dim      = c(nSim, 2),
  dimnames = list(Sim = seq_len(nSim), Year = c(1990, 2010))
)
wt <- Weight(Pars = list(a = a_arr, b = 3.0))

# ---- Direct array specification ----

## Supply MeanAtAge directly (Pars left empty).
## A plain numeric vector of length nAge is accepted; it is treated as a
## single simulation, single year and replicated by Extend() as needed.
ages  <- 0:20
Linf  <- 100; K <- 0.2; t0 <- -0.1; a <- 0.01; b <- 3
wt_aa <- Weight(MeanAtAge = a * (Linf * (1 - exp(-K * (ages - t0))))^b)

## Named Sim x Age x Year array - two change-point years, single simulation.
## Extend() replicates Sim = 1 to all nSim simulations automatically.
maa <- array(
  c(a * (Linf       * (1 - exp(-K * (ages - t0))))^b,   # 1990 schedule
    a * (Linf * 1.1 * (1 - exp(-K * (ages - t0))))^b),  # 2010 (Linf up 10%)
  dim      = c(1, length(ages), 2),
  dimnames = list(Sim = 1, Age = ages, Year = c(1990, 2010))
)
wt_arr <- Weight(MeanAtAge = maa)

## Supply MeanAtLength directly - converted to MeanAtAge via the ALK
## during Populate() when a Length object is provided.
lens      <- seq(5, 120, by = 5)
wt_al <- Weight(
  MeanAtLength = array(
    a * lens^b,
    dim      = c(1, length(lens), 1),
    dimnames = list(Sim = 1, Length = lens, Year = 1990)
  )
)

# ---- Slot accessors ----

wt <- Weight(Pars = list(a = 0.01, b = 3.0))
```

```
## Read slots
MeanAtAge(wt)     # NULL until Populate() is called
MeanAtLength(wt)  # NULL until Populate() is called (at-length models only)
CVatAge(wt)       # NULL by default; AWK is only built when non-NULL
Units(wt)

## Replace slots
Units(wt)   <- "kg"
TruncSD(wt) <- 3

# ---- Attaching to a Stock ----

stk <- Stock(Name = "Example stock", Ages = Ages(MaxAge = 20))
Weight(stk) <- Weight(Pars = list(a = 0.01, b = 3.0))
Weight(stk)

# ---- Populate ----

## The final populated object (typically done internally).
## An at-length model requires a populated Length object.
pop_wt <- Populate(Weight(stk), Ages = Ages(MaxAge = 20))
pop_wt
```

---

`weight-class` *The* `weight` *S4 Class*

---

## Description

Defines the weight-at-age and weight-at-length schedules for a stock object. Objects are typically created via `Weight()`, which documents all parameters, validates inputs, and infers the model automatically when possible.

## Details

Direct construction via `methods::new()` is not recommended; use `Weight()` instead, which handles model inference and object validation.

The `AWK` slot is only populated during `Populate()` when `CVatAge` is non-`NULL`. For most applications weight variability is propagated through the age-length key (`ALK`, see `Length()`) and a length-weight relationship rather than directly through an age-weight distribution.

## Slots

`Pars list`. Named list of parameters whose names correspond to those expected by `Model`. See `WeightModels()` for parameter sets and Specifying Biological and Fleet Schedules for accepted input formats.

`Model function` or `character(1)`. Weight model identifier, matched to one of `WeightModels()`. Set automatically from the parameter names when `Pars` is supplied without an explicit model.

`Units character(1)`. Physical unit of the weight measurements (e.g., `"g"`, `"kg"`). Must be accepted by `ValidUnits()`.

`MeanAtAge array`. Mean weight at age with named dimensions `Sim`, `Age`, and `Year`. Populated automatically by `Populate()` when `Pars` and `Model` are set; may also be supplied directly when `Pars` is empty. When `Pars` contains a matched model, any existing values are overwritten. See Specifying Biological and Fleet Schedules.

`MeanAtLength array`. Mean weight at length with named dimensions `Sim`, `Length`, and `Year`. Populated automatically when an at-length weight model is used, or may be supplied directly. Values are evaluated at the midpoint of each bin (halfway between consecutive lower bounds); the `Class` dimension is labelled by bin lower bounds (see `Classes`). Converted to `MeanAtAge` via the `ALK` during `Populate()` when `MeanAtAge` is not already populated. See Specifying Biological and Fleet Schedules.

`CVatAge array` or `numeric`. Coefficient of variation of weight-at-age, used to generate the within-age-class weight distribution and to build the age-weight key (`AWK`). `NULL` by default; the `AWK` is only constructed when this slot is populated.

`Dist character(1)`. Parametric distribution used for weight-at-age variability (e.g., `"lognormal"`).

`TruncSD array` or `numeric`. Number of standard deviations at which the weight distribution is truncated.

`Timing array` or `numeric`. Fractional position within the time step at which weight is measured (`0` = start, `1` = end).

`Random array`. Random effects on weight parameters across simulations or years.

`AWK array`. Age-weight key: the probability of belonging to each weight class given age, derived from `MeanAtAge`, `CVatAge`, `Dist`, and `TruncSD`. Only populated when `CVatAge` is non-`NULL`; use `AWK()` to retrieve.

`Classes numeric`. Lower bounds of weight bins in units of `Units`. Bin `k` spans `[Classes[k], Classes[k+1])`; the final bin is open-ended. Values in `MeanAtLength` are evaluated at bin midpoints, not at these lower bounds.

`Misc list`. Used internally.

## See Also

- `Weight()` for the constructor and accessor functions.
- `WeightModels()` for available models and their required parameters.
- `ValidUnits()` for accepted unit strings.
- `Populate()` for array population.
- `AWK()` to retrieve the age-weight key.
- `Length()` for the companion length schedule, required when using at-length weight models.
- Specifying Biological and Fleet Schedules for the full description of how `Pars`, `Model`, and `MeanAt*` arrays interact.

Other weight: `Weight()`

---

`Weight-Models` *Weight-at-Age and Weight-at-Length Models*

---

## Description

Allometric weight models used to compute weight at age or at length.

## Usage

```
WeightatAge(Ages, a, b)

WeightatLength(Length, Alpha, Beta)

WeightatMeanLength(Length, alpha, beta)

WeightModels(full = TRUE, print = TRUE)
```

## Arguments

| | |
|---|---|
| `Ages` | numeric vector of age classes (for Weight-at-Age) |
| `a` | scaling parameter for Weight-at-Age in units of `Weight()`. |
| `b` | exponent for Weight-at-Age. Typically around 3. |
| `Length` | numeric vector of length classes for `WeightatLength` or a mean length-at-age schedule for `WeightatMeanLength` |
| `Alpha` | scaling parameter for Weight-at-Length in units of `Weight()`. |
| `Beta` | exponent for Weight-at-Length. Typically around 3. |
| `alpha` | scaling parameter for Weight-at-Mean-Length in units of `Weight()`. |
| `beta` | exponent for Weight-at-Mean-Length. Typically around 3. |
| `full` | logical; provide a complete table of models (TRUE) or just model names (FALSE) |
| `print` | logical; print results (TRUE) or return data frame invisibly (FALSE) |
| | `WeightModels()` prints the list of available weight models. |

## Details

Models are identified internally by matching the names in `Pars`. `WeightatMeanLength` returns weight-at-age corresponding to the mean length at each age, while `WeightatLength` returns weight-at-length for the supplied length classes.

- **Weight-at-Age**:
$$W(a) = a \cdot Age^{b}$$
- **Weight-at-Length**:
$$W(L) = \alpha \cdot L^{\beta}$$

- **Weight-at-Mean-Length**:

$$W(L_{mean}) = \alpha_{mean} \cdot L_{mean}^{\beta_{mean}}$$

, returning weight-at-age

at-length schedules (except from `WeightatMeanLength`) are converted internally to at-age using the age-length key respectively.

### Value

Each function returns a numeric vector of expected weights for each age or length. `WeightModels()` invisibly returns a data frame describing available models.

### See Also

`Ages()`, `Length()`, `Stock()`, `Weight()`

### Examples

```
Ages <- 0:10
MeanLength <- vonBert(Ages, 100, 0.2)
Length <- seq(10, 100, 10)

W_age <- WeightatAge(Ages, a = 59.56, b = 2)
W_mean <- WeightatMeanLength(MeanLength, alpha = 0.01, beta = 3)
W_length <- WeightatLength(Length, Alpha = 0.0001, Beta = 3)

df <- data.frame(
  Age = Ages,
  W_age = W_age,
  W_mean = W_mean
)

# Plot Weight-at-Age and Weight-at-Mean-Length
matplot(df$Age, df[,-1], type = "l", lty = 1, lwd = 2,
        col = c("blue", "purple"), xlab = "Age (years)", ylab = "Weight",
        main = "Weight-at-Age Models")
legend("topleft", legend = colnames(df)[-1], col = c("blue", "purple"), lty = 1, lwd = 2)

# Plot Weight-at-Length
plot(Length, W_length, type = "l", lwd = 2, col = "red",
     xlab = "Length", ylab = "Weight", main = "Weight-at-Length Model")
```

---

`WHAM2OM` *Takes a fitted SAM model and samples historical population and fishing dynamics from the MLE fit and variance-covariance matrix.*

---

## Description

Takes a fitted SAM model and samples historical population and fishing dynamics from the MLE fit and variance-covariance matrix. Maturity-at-age-year, Mortality-at-age-year and weight-at-age-year are identical among simulations and are a direct copy of the matrices in the WHAM fitting object.

## Usage

```
WHAM2OM(
  obj,
  nsim = 3,
  proyears = 30,
  interval = 2,
  Name = NULL,
  WLa = 1,
  WLb = 3,
  WAAind = 1,
  Obs = MSEtool::Imprecise_Unbiased,
  Imp = MSEtool::Perfect_Imp,
  nyr_par_mu = 3,
  LowerTri = 2,
  plusgroup = T,
  altinit = 0,
  fixq1 = T,
  report = FALSE,
  silent = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `obj` | a SAM output object |
| `nsim` | Positive integer. The number of simulations. |
| `proyears` | Positive integer. The number of projection years for MSE. |
| `interval` | Positive integer. The interval at which management procedures will update the management advice in runMSE, e.g., 1 = annual updates. |
| `Name` | Character string. The name of the operating model. |
| `WLa` | positive real number or array `[sim, ages, year]`. The default weight-length parameter a (W=aL^b) |
| `WLb` | positive real number or array `[sim, ages, year]`. The default weight-length parameter b (W=aL^b) |
| `WAAind` | positive integer. The index of the WHAM weight-at-age array input$data$waa to be assumed as the weight-at-age for the operating model |
| `Obs` | The observation model (class Obs). This function only updates the catch and index observation error. |

| | |
|---|---|
| `Imp` | The implementation model (class Imp). This function does not update implementation parameters. |
| `nyr_par_mu` | Positive integer. The number of recent years that natural mortality, age vulnerability, weight, length and maturity parameters are averaged over for defining future projection conditions. |
| `LowerTri` | Integer. The number of recent years for which model estimates of recruitment are ignored (not reliably estimated by the assessment) |
| `plusgroup` | Logical. Does the assessment assume that the oldest age class is a plusgroup? |
| `altinit` | Integer. Various assumptions for how to set up the initial numbers. 0: standard, 1: no plus group, 2: temporary fix for MSEtool plus group initialization |
| `fixq1` | Logical. Should q be fixed (ie assume the F-at-age array faa is accurate? |
| `report` | Logical, if TRUE, a diagnostic will be reported showing the matching of the OM reconstructed numbers at age vs the assessment. |
| `silent` | Whether to silence messages to the console. |
| `...` | Additional arguments, including R0 (unfished recruitment), phi0 (unfished spawners per recruit associated with R0 and h for calculating stock recruit parameters), |

### Details

Use a seed for the random number generator to sample future recruitment.

### Value

An object of class `OM`.

### Author(s)

T. Carruthers

### See Also

Assess2OM

---

## `wormplot` *Biomass wormplot*

---

### Description

A worm plot for plotting the likelihood of meeting biomass targets in future years.

### Usage

```
wormplot(MSEobj, Bref = 0.5, LB = 0.25, UB = 0.75)
```

### Arguments

| | |
|---|---|
| MSEobj | Object of class MSE, output of the runMSE function |
| Bref | The reference fraction of BMSY (to evaluate the probability of exceeding this level) |
| LB | The lower bound probability that separates red (bad) and yellow (O.K.) colored segments |
| UB | The upper bound probability that separates yellow (O.K.) and green (good) colored segments |

### Details

Returns a matrix of nMPs rows and proyears columns which is the fraction of simulations for which biomass was above Bref.

### Author(s)

T. Carruthers

---

## WrappedNormal *Wrapped Normal Distribution*

---

### Description

Creates a wrapped normal distribution to represent a seasonal recruitment patterns in models with `Seasons > 1`.

### Usage

```
WrappedNormal(n_seasons, mu, sigma, n_wraps = 5)
```

### Arguments

| | |
|---|---|
| n_seasons | Integer. Number of seasons per year (e.g. `12` for monthly, `4` for quarterly). |
| mu | Numeric. Peak season (need not be an integer; e.g. `3.5` for a peak between seasons 3 and 4). |
| sigma | Numeric. Standard deviation in units of seasons. Smaller values concentrate recruitment around the peak; larger values spread it across the year. |
| n_wraps | Integer. Number of periods summed on each side to approximate wrapping. Default `5` is sufficient for any reasonable `sigma`. |

### Value

A numeric vector of normalised density values of length `n_seasons`.

### Examples

```
# Monthly model: peak recruitment March/April, moderate spread
pi_m <- WrappedNormal(n_seasons = 12, mu = 3.5, sigma = 1.5)
names(pi_m) <- month.abb

# Narrow spread - recruitment concentrated in 1-2 seasons
pi_m <- WrappedNormal(n_seasons = 12, mu = 3.5, sigma = 0.5)

# Wide spread - recruitment across most of the year
pi_m <- WrappedNormal(n_seasons = 12, mu = 3.5, sigma = 3)

# Quarterly model: peak in Q1
pi_m <- WrappedNormal(n_seasons = 4, mu = 1.5, sigma = 0.8)
```

---

## WrasseExStock *Wrasse Example Stocks (Female / Male)*

---

### Description

A pair of example stock objects representing the female and (terminal) male phase of a protogynous hermaphrodite wrasse (*Thalassoma* spp.), paired via HermOM's `Herm()` transition and the male's `SRR@SPFrom`. Not intended to represent any specific stock; illustrates `SRR()`'s `SPFrom` cross-stock spawning-production sourcing and `Herm()`'s age-dependent stock transition together. See HermOM for the full worked example and how the two stocks and the transition are combined.

### Usage

```
WrasseFemaleExStock

WrasseMaleExStock
```

### Format

Each is a stock object with the following slots populated:

- `Name`: `"Example Wrasse Female Stock"` / `"Example Wrasse Male Stock"`.
- `CommonName`: `"Wrasse"`.
- `Species`: `"Thalassoma spp."`.
- `Ages`: An `Ages()` object with `MaxAge = 15`, identical for both stocks - required for the `Herm()` transition, which reclassifies individuals age-for-age between the two stocks.
- `Length`: A `Length()` object with von Bertalanffy parameters. Terminal males grow larger than females (`Linf` 30-35 cm vs 23-27 cm), typical of this style of protogynous species; `K`/`t0` otherwise similar.
- `Weight`, `Maturity`: `Weight()`/`Maturity()` objects; males mature at a larger size than females, consistent with their larger size-at-age.

- `NaturalMortality`: A `NaturalMortality()` object; males have somewhat lower `M` (0.2-0.3 yr^-1^) than females (0.3-0.4 yr^-1^).
- `SRR`: A `SRR()` object. The female is self-recruiting (`SPFrom` unset) with `R0 = 5000`. The male's `SPFrom = "Example Wrasse Female Stock"` sources its spawning production (`SP`/`SP0`) from the female stock. #'
- `Spatial`, `Depletion`: matching `Spatial()`/`Depletion()` objects for both stocks.

An object of class `stock` of length 1.

### See Also

HermOM for the full worked example (including the `Herm()` transition curve linking these two stocks), `SRR()` for `SPFrom`, stocktransition for the transition mechanism, AlbacoreExStock for a single-stock example with the full per-slot `Populate*()` walk-through.

### Examples

```
WrasseFemaleExStock
WrasseMaleExStock
```

---

## writeCSV *Internal function to write CSVs for objects*

---

### Description

Used internally in the DLMtool package to write CSV files from an existing DLMtool object

### Usage

```
writeCSV(
  inobj,
  tmpfile = NULL,
  objtype = c("Stock", "Fleet", "Obs", "Imp", "Data", "OM")
)
```

### Arguments

| | |
|---|---|
| `inobj` | A object of class Stock, Fleet, Obs, Imp, Data, or OM |
| `tmpfile` | The full file path and name for the saved CSV file |
| `objtype` | The class corresonding to the `inobj` |

### Author(s)

A. Hordyk

---

`XL2Data` *Import a Data object from Excel file*

---

### Description

Import a Data object from Excel file

### Usage

```
XL2Data(name, dec = c(".", ","), sheet = 1, silent = FALSE)
```

### Arguments

| | |
|---|---|
| `name` | Name of the data file, with or without file extension. Include full file path if not in working directory |
| `dec` | the character used in the file for decimal points. |
| `sheet` | Sheet number if importing Data from XL file |
| `silent` | Logical. Hide messages? |

### Value

An object of class 'Data'

### Author(s)

A. Hordyk

### Examples

```
## Not run:
MyData <- XL2Data("MyData.xlsx")

## End(Not run)
```

---

`XL2Fleet` *Import Fleet Object from Excel file*

---

### Description

Imports a Fleet Object from a correctly formatted Excel file.

### Usage

```
XL2Fleet(name = NULL, cpars = NULL, msg = TRUE)
```

### Arguments

| | |
|---|---|
| `name` | Name of the OM Excel file. Provide full file path if not in current directory. |
| `cpars` | An optional list of custom parameters (single parameters are a vector nsim long, time series are a matrix nsim rows by nyears columns) |
| `msg` | Should messages be printed? |

### Details

An error message will alert if any slots are missing values, or if the Excel file is missing the required tabs.

### Value

An object of class Fleet

### Author(s)

A. Hordyk

---

`XL2OM` *Load OM from Excel file*

---

### Description

Imports an OM from a correctly formatted Excel file. Create the Excel spreadsheet template using `OMinit` and document each slot in the corresponding text file.

### Usage

```
XL2OM(name = NULL, cpars = NULL, msg = TRUE)
```

### Arguments

| | |
|---|---|
| `name` | Name of the OM Excel file. Provide full file path if not in current directory. |
| `cpars` | An optional list of custom parameters (single parameters are a vector nsim long, time series are a matrix nsim rows by nyears columns) |
| `msg` | Should messages be printed? |

### Details

An error message will alert if any slots are missing values, or if the Excel file is missing the required tabs.

### Value

An object of class OM

### Author(s)

A. Hordyk

### Examples

```
## Not run:
OMinit('myOM', templates=list(Stock='Herring', Fleet='Generic_Fleet', Obs='Generic_Obs',
Imp='Perfect_Imp'), overwrite=TRUE)
myOM <- XL2OM('myOM.xlsx')


## End(Not run)
```

---

## XL2Stock *Import Stock Object from Excel file*

---

### Description

Imports a Stock Object from a correctly formatted Excel file.

### Usage

```
XL2Stock(name = NULL, cpars = NULL, msg = TRUE)
```

### Arguments

| | |
|---|---|
| `name` | Name of the OM Excel file. Provide full file path if not in current directory. |
| `cpars` | An optional list of custom parameters (single parameters are a vector nsim long, time series are a matrix nsim rows by nyears columns) |
| `msg` | Should messages be printed? |

### Details

An error message will alert if any slots are missing values, or if the Excel file is missing the required tabs.

### Value

An object of class Stock

### Author(s)

A. Hordyk

`Years` *Calculate or Extract Time Step Values*

## Description

Two related functions for working with historical and projection time steps:

## Usage

```
CalcYears(nYear, pYear, CurrentYear, Seasons = 1, Period = NULL)

Years(x, Period = NULL)

Years(x) <- value
```

## Arguments

| | |
|---|---|
| `nYear` | Integer. Number of historical time steps. |
| `pYear` | Integer. Number of projection time steps. |
| `CurrentYear` | Integer. Final historical year (or index if `CurrentYear < 1900`). |
| `Seasons` | Integer. Number of seasons per year. Default `1` (annual). |
| `Period` | Character or `NULL`. Controls which time steps are returned. See Details. |
| `x` | An om, hist, or mse object, or a list of such objects. (`Years()` only.) |

## Details

- `CalcYears()` generates a numeric vector of time step values from explicit parameters.
- `Years()` extracts time step values from an om, hist, or mse object, or a list of such objects, by delegating to `CalcYears()`.

The historical period spans `nYear` time steps ending at `CurrentYear`. The projection period spans `pYear` time steps beginning after `CurrentYear`. If `pYear = 0` or `Period = "H"`, only historical values are returned.

For sub-annual seasons (`Seasons > 1`), time steps are expressed as decimal dates (e.g. `2020.5` for mid-2020) using `lubridate::decimal_date()`. Supported resolutions are half-year, quarter, month, week, and day, as determined by `CalcTSUnits()`.

When `CurrentYear < 1900`, a non-calendar index is used and years are counted backwards from `CurrentYear` rather than mapped to real calendar years.

`Period` accepts partial matches to `"Historical"` or `"Projection"` (e.g. `"H"`, `"Hist"`, `"Proj"`). Matching is case-insensitive. `NULL` (default) returns all time steps.

## Value

A numeric vector of time step values, or for `Years()` applied to a list, a list of such vectors.

### See Also

`CalcTSUnits()`, om

### Examples

```
# Annual time steps
CalcYears(nYear=20, pYear=10, CurrentYear=2026)

# Semi-annual time steps
CalcYears(nYear=20, pYear=10, CurrentYear=2026, Seasons=2)

# Historical period only
CalcYears(nYear=20, pYear=10, CurrentYear=2026, Period="H")
```

# Index


* ~~
  initialize-methods, 327
* **ages**
  Ages, 33
  ages-class, 35
* **classes**
  MMSE-legacy-class, 380
  MOM-legacy-class, 384
* **datasets**
  AgeStructuredObs, 36
  Albacore, 40
  Albacore_TwoFleet, 41
  AlbacoreExStock, 41
  AsympExFleet, 58
  Atlantic_mackerel, 60
  ButterfishExStock, 71
  CatchAndSurveyObs, 106
  CommercialFleetObs, 127
  ComplexOM, 135
  DataRichObs, 173
  DataSlots, 174
  DecE_Dom, 175
  DomeExFleet, 199
  FullComplianceImp, 262
  Generic_Obs, 265
  HermOM, 281
  LengthStructuredObs, 351
  LHdatabase, 354
  MultiStockOM, 402
  OverageImp, 442
  Overages, 443
  PartialSizeComplianceImp, 444
  ReqData, 576
  SeasonalSpatialExStock, 594
  SeasonalSpatialOM, 596
  SimulatedData, 624
  SingleStockOM, 626
  Taxa_Table, 695
  testOM, 696
  TwoFleetOM, 713
  UnderageImp, 715
  WrasseExStock, 737
* **data**
  CatchData, 107
  catchdata-class, 108
  CompData, 132
  compdata-class, 134
  EffortData, 213
  effortdata-class, 214
  IndicesData, 319
  indicesdata-class, 320
* **depletion**
  Depletion, 179
  depletion-class, 182
* **fecundity**
  Fecundity, 233
  fecundity-class, 239
* **fleet**
  Bioeconomic, 67
  bioeconomic-class, 70
  Catchability, 101
  catchability-class, 105
  DiscardMortality, 191
  discardmortality-class, 195
  Effort, 202
  effort-class, 210
  Fleet, 252
  fleet-class, 256
  Retention, 577
  retention-class, 581
  Selectivity, 599
  selectivity-class, 603
* **imp**
  FullComplianceImp, 262
  Imp, 292
  imp-class, 294
  ImpSlot, 310
  OverageImp, 442

PartialSizeComplianceImp, 444
UnderageImp, 715
* **keyword(s)**
initialize-methods, 327
* **length**
Length, 343
length-class, 349
* **maturity**
Maturity, 369
maturity-class, 376
* **methods**
initialize-methods, 327
* **naturalmortality**
NaturalMortality, 405
naturalmortality-class, 410
* **obs-examples**
AgeStructuredObs, 36
CatchAndSurveyObs, 106
CommercialFleetObs, 127
DataRichObs, 173
LengthStructuredObs, 351
* **obs**
CatchObs, 110
catchobs-class, 113
CompObs, 137
compobs-class, 140
EffortObs, 215
effortobs-class, 217
ExploitationObs, 226
exploitationobs-class, 227
IndicesObs, 321
indicesobs-class, 326
LifeHistoryObs, 355
lifehistoryobs-class, 357
Obs, 416
obs-class, 419
* **om**
CatchFrac, 109
ComplexOM, 135
DataLag, 171
EffortAllocation, 211
Herm, 278
HermOM, 281
HistoricalWeight, 288
InterimAdvice, 328
Interval, 330
MPStartYear, 388
MultiStockOM, 402
OM, 425
OM-accessors, 430
om-class, 433
RefSeason, 566
SeasonalSpatialOM, 596
SingleStockOM, 626
stocktransition-class, 678
TwoFleetOM, 713
* **other**
initialize-methods, 327
* **possible**
initialize-methods, 327
* **spatial**
Spatial, 630
spatial-class, 637
* **srr**
SRR, 644
srr-class, 652
* **stock**
Stock, 667
stock-class, 672
* **weight**
Weight, 725
weight-class, 730
.AddAutoCorrelation, 15

AAVE *(PerformanceMetric)*, 445
AAVY *(PerformanceMetric)*, 445
abind::abind(), *339*, *340*
AC *(Access)*, 16
AC<- *(Access)*, 16
Access, 16
AddAgeIndices *(AddIndex)*, 22
AddAgeIndices(), *323*
AddDimension *(dimension-utils)*, 190
AddFleet, 21
AddIndex, 22
AddIndex(), *318*, *320*, *323*
addMMPs, 24
addMMPs(), *117*
addMPs *(checkMSE)*, 116
addMPs(), *24*
AdjustSeasonalFecundity, 25
AdjustSeasonalFecundity(), *538*, *609*, *610*
Advice, 26
advice, *26*, *28*, *30*, *31*, *37*, *38*, *223–225*, *317*, *361*, *564*, *691*, *692*

Advice(), *27*, *31*, *32*, *38*, *39*, *114*, *142*, *212*, *224*, *225*, *243*, *291*, *294*, *295*, *316*, *317*, *329*, *391*, *466*, *564*, *689*, *694*
advice-class, 31
AdviceData *(advicedata-class)*, 32
advicedata, *159–161*
advicedata *(advicedata-class)*, 32
advicedata-class, 32
Agency *(OM-accessors)*, 430
Agency<- *(OM-accessors)*, 430
Ages, 33
ages, *33–35*, *74*, *94*, *177*, *355*, *668*, *672*, *721*
Ages(), *35*, *36*, *42*, *71*, *73*, *74*, *153*, *177*, *242*, *269*, *349*, *379*, *410*, *412*, *522*, *526*, *532*, *534*, *535*, *539*, *541*, *543*, *544*, *548*, *585*, *594*, *606*, *668–670*, *672*, *673*, *721*, *722*, *733*, *737*
ages-class, 35
Ages<- *(Ages)*, 33
AgeStructuredObs, 36, *106*, *127*, *174*, *282*, *351*, *403*, *626*, *627*, *714*
aggbaglimit, *38*, *39*
aggbaglimit-class, 37
AggregateBagLimit, 38
AggregateBagLimit(), *37*, *38*
Albacore, 40
Albacore_TwoFleet, 41
AlbacoreExStock, 41, *58*, *59*, *71*, *73*, *136*, *199*, *200*, *402*, *403*, *596*, *626*, *627*, *714*, *738*
ALK *(Length)*, 343
ALK(), *350*, *470*, *471*
ALK<- *(Length)*, 343
allometric *(model-helpers)*, 382
AnglerPerTrip *(Effort)*, 202
AnglerPerTrip(), *28*, *31*
AnglerPerTrip<- *(Effort)*, 202
AnnualData, 44
AnnualData(), *169*, *249*, *250*, *316*, *317*, *690*, *692*
ApplyAC, 45
ApplyAC(), *88*, *89*, *271*, *325*
applyMMP, 46
applyMP, 46
Areas *(Access)*, 16
Areas<- *(Access)*, 16
Arrangement *(Spatial)*, 630
Arrangement<- *(Spatial)*, 630
array(), *229*
Array2DF *(array_df_conversion)*, 47
Array2DF(), *21*
Array2List *(list-array-conversion)*, 358
array_df_conversion, 47
ArrayDivide *(ArrayOperations)*, 48
ArrayDivide(), *89*, *90*
ArrayExtend *(ArrayOperations)*, 48
ArrayFill<- *(ArrayOperations)*, 48
ArrayMultiply *(ArrayOperations)*, 48
ArrayOperations, 48
ArraySubtract *(ArrayOperations)*, 48
ArraySum *(ArrayOperations)*, 48
as.numeric(), *47*
ASAP2Data *(ASAP2OM)*, 51
ASAP2OM, 50, *57*
Assess2MOM, 52
Assess2OM, *52*, *54*, 55, *335*, *735*
AsympExFleet, 58, *135*, *136*, *282*, *403*, *626*, *627*, *714*
Atlantic_mackerel, 60
Author *(OM-accessors)*, 430
Author<- *(OM-accessors)*, 430
avail, 61, *61*, *98*
AverageCatch *(ExampleMPs)*, 223
AverageCatch(), *172*
Awatea2OM, 62
AWK *(Weight)*, 725
AWK(), *471*, *472*, *731*
AWK<- *(Weight)*, 725

B0 *(extract-unfished)*, 231
B0(), *569*, *571*
B_B0 *(relative_ref)*, 568
B_B0(), *465*, *466*, *468*, *476*
B_BMSY *(relative_ref)*, 568
B_BMSY(), *465*, *468*, *641*, *692*
BagLimit *(Advice)*, 26
BagLimit<- *(Advice)*, 26
BAM2MOM, 63
BAM2OM *(BAM2MOM)*, 63
BAMFleetNames, 64
base::vector(), *365*
Beta *(Access)*, 16
Beta<- *(Access)*, 16
BevertonHolt *(SRRModels)*, 654

BevertonHolt(), *498*
BevertonHolt_RelRec *(SRRModels)*, 654
Bias *(Access)*, 16
Bias<- *(Access)*, 16
bio_timeseries, 65
Bioeconomic, 67
bioeconomic, *67–70*, *253*, *257*
Bioeconomic(), *58*, *70*, *104*, *105*, *193*, *195*, *199*, *208*, *211*, *254*, *255*, *257*, *579*, *582*, *601*, *604*
bioeconomic-class, 70
Bioeconomic<- *(Bioeconomic)*, 67
Biomass *(bio_timeseries)*, 65
Biomass(), *132*, *416*, *466*, *468*, *485*, *570*, *571*
BLow *(refpoints-accessors)*, 564
Blue_shark *(Albacore)*, 40
Bluefin_tuna *(Albacore)*, 40
Bluefin_tuna_WAtl *(Albacore)*, 40
BMSY *(MSY-accessors)*, 397
BMSY(), *571*
boxplot.Data, 70
Brody *(Length-at-Age-Models)*, 347
Butterfish *(Albacore)*, 40
ButterfishExStock, *41*, *43*, *58*, *59*, 71, *136*, *199*, *200*, *402*, *403*, *596*

CalcAgeClasses, 73
CalcAgeClasses(), *34*, *36*, *42*, *72*, *595*
CalcAgeSizeKey, 75
CalcAgeSizeKey(), *471*, *472*
CalcAsymDist, 77
CalcAsymDist(), *94*
CalcBLow, 78
CalcBLow(), *87*, *565*, *618*
CalcEquilibrium, 79
CalcEquilibrium(), *221*, *222*, *567*
CalcIndexResiduals, 80
CalcInnovationCov, 81
CalcInnovationCov(), *268*
CalcMGT, 82
CalcMGT(), *78*, *79*, *618*
CalcMSY, 83
CalcMSY(), *79*, *80*, *83*, *87*, *398*, *437*, *564*, *567*, *570*, *618*, *622*, *623*
CalcPerRecruit, 85
CalcPerRecruit(), *80*, *85*, *87*, *446–448*
CalcRefPoints, 86
CalcRefPoints(), *79*, *437*, *564*, *565*, *618*
calcRefYield, 87
CalcResidualStats, 88
CalcResidualStats(), *45*, *271*, *322*, *324–327*
CalcSeasons *(CalcTSUnits)*, 91
CalcSPR0, 89
CalcSPR0(), *83*, *85*, *86*, *658*
CalcSurvival, 90
CalcTSUnits, 91
CalcTSUnits(), *742*, *743*
CalcUnfished_Dynamic, 92
CalcUnfished_Dynamic(), *93*
CalcUnfished_Equilibrium, 93
CalcUnfished_Equilibrium(), *92*
CalcUnfishedDist, 94
CalcUnfishedDist(), *78*
CalcUnfishedNumber, 95
CalcUnfishedNumber(), *25*
CalcUnfishedSurvival, 95
CalcYears *(Years)*, 742
CalcYears(), *92*, *428*, *429*, *434*, *436*
CALsimp, 96
Can, *61*, 97, *577*
Cant, *61*, *577*
Cant *(Can)*, 97
Caption *(Access)*, 16
Caption<- *(Access)*, 16
catch_timeseries, 98
Catchability, 101
catchability, *101–103*, *203*, *205*, *253*, *256*
Catchability(), *58*, *69*, *70*, *105*, *148*, *179–181*, *183*, *193*, *195*, *199*, *208*, *211*, *253–257*, *473*, *516*, *517*, *579*, *582*, *601*, *604*, *669*, *672*
catchability-class, 105
Catchability<- *(Catchability)*, 101
CatchAndSurveyObs, *37*, 106, *127*, *174*, *351*, *403*, *714*
CatchData, 107
catchdata, *100*, *101*, *107*, *108*, *112*, *159–161*, *301*
CatchData(), *108*, *109*, *133*, *135*, *213*, *214*, *301*, *304*, *320*, *321*
catchdata-class, 108
CatchFrac, 109
CatchFrac(), *123*, *136*, *172*, *212*, *279*, *282*, *289*, *330*, *331*, *389*, *403*, *429*, *432*, *436*, *567*, *598*, *627*, *678*, *714*

CatchFrac<- *(CatchFrac)*, 109
CatchObs, 110
catchobs, *37*, *100*, *101*, *106*, *110*, *112*, *113*, *127*, *173*, *351*, *417*, *420*, *517*, *518*
CatchObs(), *37*, *106*, *108*, *113*, *114*, *127*, *139*, *141*, *174*, *217*, *218*, *227*, *228*, *325*, *327*, *351*, *357*, *358*, *417*, *418*, *420*
catchobs-class, 113
CheckCatch, 114
CheckCatch(), *107*, *108*, *225*, *317*
CheckDuplicate, 115
checkHerm *(Herm-int)*, 280
CheckMPs, 115
checkMSE, 116
CheckOM, 117
CheckPackage, 118
CheckParallel *(SetupParallel)*, 611
CheckParallel(), *360*
China_rockfish *(Atlantic_mackerel)*, 60
ChkYrs, *446*
Choose, 118
ChooseEffort *(Choose)*, 118
ChooseM *(Choose)*, 118
ChooseSelect *(Choose)*, 118
Classes *(Access)*, 16
Classes(), *35*, *36*, *193*, *579*, *601*
Classes<- *(Access)*, 16
CleanupOrphanWorkers *(ListParallelWorkers)*, 359
CleanupOrphanWorkers(), *612*
cli::cli_abort(), *340*
cli::cli_warn(), *476*
Closure *(Fleet)*, 252
Closure(), *254*
Closure<- *(Fleet)*, 252
ClosureMode *(Advice)*, 26
ClosureMode<- *(Advice)*, 26
Cobia *(Atlantic_mackerel)*, 60
CollapseSeasons, 120
CombineFleets, 121
CombineFleets(), *120*, *121*, *125*
CombineMMP, 124
CombineMSE, 124
CombineMSE(), *501*, *504*, *506*, *507*
CombineOMs, 125
CombineOMs(), *125*
CommercialFleetObs, *37*, *106*, 127, *174*, *282*, *351*, *403*, *596–598*
CommonName *(Stock)*, 667
CommonName<- *(Stock)*, 667
CompareBAM, 128
CompareBAM(), *129–132*
CompareiSCAM, 129
CompareiSCAM(), *304*
CompareSS, 130
CompareSS(), *129*, *130*, *308*
CompareSS_Biomass *(CompareSS)*, 130
CompareSS_Discards *(CompareSS)*, 130
CompareSS_Landings *(CompareSS)*, 130
CompareSS_Number *(CompareSS)*, 130
CompData, 132
compdata, *99*, *100*, *132*, *133*, *159–161*, *301*, *303*, *457*
CompData(), *108*, *109*, *134*, *135*, *213*, *214*, *320*, *321*
compdata-class, 134
Complexes *(OM-accessors)*, 430
Complexes<- *(OM-accessors)*, 430
ComplexOM, *110*, 135, *172*, *212*, *279*, *282*, *289*, *330*, *331*, *389*, *402*, *403*, *429*, *432*, *436*, *567*, *596*, *598*, *626*, *627*, *678*, *713*, *714*
Compliance *(Access)*, 16
Compliance<- *(Access)*, 16
CompObs, 137
compobs, *37*, *137*, *139*, *173*, *351*, *417*, *420*, *519*, *520*
CompObs(), *37*, *113*, *114*, *132*, *133*, *140*, *141*, *174*, *217*, *218*, *227*, *228*, *325*, *327*, *351*, *357*, *358*, *417*, *418*, *420*, *474*, *520*, *557*, *558*
CompObs-class *(compobs-class)*, 140
compobs-class, 140
ConstrainTAC, 141
ConstrainTAC(), *316*, *317*, *505*, *689*, *691*, *692*
Control *(OM-accessors)*, 430
Control<- *(OM-accessors)*, 430
Converge, 142
Convert, 144
Convert(), *147–151*, *153*, *154*
ConvertData, 145
ConvertData(), *144*, *145*
ConvertFleet, 147

ConvertFleet(), *144*, *145*, *150*, *153*, *154*
ConvertImp, 149
ConvertImp(), *144*, *145*, *148*, *150*, *151*, *153*, *154*, *294*, *295*
ConvertMOM, 149
ConvertMOM(), *144*, *145*, *147–149*, *151*, *153*, *154*, *429*
ConvertObs, 150
ConvertObs(), *144*, *145*, *148–150*, *153*, *154*, *227*, *356*, *418*, *420*
ConvertOM, 152
ConvertOM(), *144*, *145*, *147–151*, *154*, *429*
ConvertStock, 153
ConvertStock(), *144*, *145*, *148*, *150*, *153*
Cos_thresh_tab, 154
COSEWIC_Blow *(runCOSEWIC)*, 585
COSEWIC_Dplot *(runCOSEWIC)*, 585
COSEWIC_Hplot *(runCOSEWIC)*, 585
COSEWIC_Pplot *(runCOSEWIC)*, 585
COSEWIC_report *(runCOSEWIC)*, 585
COSEWIC_tab *(runCOSEWIC)*, 585
COSEWIC_tab_formatted *(runCOSEWIC)*, 585
Cost *(Bioeconomic)*, 67
Cost(), *69*
Cost<- *(Bioeconomic)*, 67
cparscheck, 155
Cplot, 156
CPUE *(Access)*, 16
CPUE(), *459*
CPUE<- *(Access)*, 16
CR2h, 157
curEref *(FMSYref)*, 260
CurrentCatch, *591*
CurrentCatch *(ExampleMPs)*, 223
CurrentCatch(), *172*
CurrentEffort, *591*
CurrentEffort *(ExampleMPs)*, 223
CurrentEffort(), *114*, *564*
CurrentLandings *(ExampleMPs)*, 223
CurrentLandings(), *172*
CurrentYear *(OM-accessors)*, 430
CurrentYear<- *(OM-accessors)*, 430
CV *(Access)*, 16
CV<- *(Access)*, 16
CVatAge *(Access)*, 16
CVatAge<- *(Access)*, 16
CVDist *(Spatial)*, 630
CVDist<- *(Spatial)*, 630
CVStay *(Spatial)*, 630
CVStay<- *(Spatial)*, 630

Data, 158
data, *32*, *33*, *44*, *99–101*, *108*, *109*, *112*, *114*, *133–135*, *144–146*, *158–160*, *169*, *213*, *214*, *223–226*, *229*, *247*, *248*, *284*, *285*, *294*, *299*, *301*, *308*, *314*, *317*, *320*, *321*, *354*, *355*, *361*, *418–420*, *427*, *434*, *455–459*, *464*, *466*, *467*, *479*, *549*, *562–564*, *688*
Data(), *28*, *33*, *108*, *123*, *133*, *160*, *161*, *213*, *226*, *301*, *304*, *309*, *320*, *355*, *418*, *420*, *427*, *429*, *432*, *434*, *436*, *563*
Data-class *(Data-legacy-class)*, 162
data-class, 160
Data-class-objects *(Atlantic_mackerel)*, 60
Data-legacy, *98*, *144–146*, *261*, *577*
Data-legacy-class, 162
Data2csv, 167
Data<- *(Data)*, 158
Data_xl, 166
DataDir, 168
DataHelpers, *44*, 168
DataInit, 170
DataLag, 171
DataLag(), *110*, *136*, *212*, *225*, *279*, *282*, *289*, *330*, *331*, *368*, *369*, *389*, *403*, *427*, *429*, *432*, *434*, *436*, *567*, *598*, *627*, *678*, *714*
DataLag<- *(DataLag)*, 171
DataRichObs, *37*, *106*, *127*, *135*, *136*, 173, *351*
DataSlots, 174
DataTrim, 174
DataTrim(), *170*
DecE_Dom, 175
DecE_HDom *(DecE_Dom)*, 175
DecE_NDom *(DecE_Dom)*, 175
DefaultAges, 176
DefaultAges(), *94*
DefaultSlickPMs, 177
DefaultSlickPMs(), *394*, *396*
DefaultYears *(DefaultAges)*, 176
DefaultYears(), *94*
DeleteLogs, 178

DeleteLogs(), *362*
Depletion, 179
depletion, *179–181*, *355*, *669*, *672*
Depletion(), *42*, *72*, *182*, *183*, *476*, *521*, *594*, *669*, *670*, *672*, *673*, *738*
depletion-class, 182
Depletion<- *(Depletion)*, 179
Depreciation *(Bioeconomic)*, 67
Depreciation(), *69*
Depreciation<- *(Bioeconomic)*, 67
DF2Array *(array_df_conversion)*, 47
DFO_bar, 183
DFO_hist, 184
DFO_plot, 184
DFO_plot2, 185
DFO_proj, 185
DFO_quant, 186
DFO_report, 187
DFO_spider, 187
DFO_tab, 188
DFO_tab_formatted, 188
dim, 189
dimension-utils, 190
DisableParallel *(SetupParallel)*, 611
DisableParallel(), *310*, *360*, *611*
DiscardMortality, 191
discardmortality, *191–193*, *226*, *253*, *257*, *476*
DiscardMortality(), *27–29*, *31*, *32*, *58*, *69*, *70*, *104*, *105*, *195*, *199*, *208*, *211*, *253–255*, *257*, *522*, *523*, *578*, *579*, *582*, *601*, *604*
discardmortality-class, 195
DiscardMortality<- *(DiscardMortality)*, 191
Discards *(catch_timeseries)*, 98
Discards(), *132*, *455*, *457*, *459*, *466–468*
Discards<- *(catch_timeseries)*, 98
DiscardsAtAge *(Access)*, 16
DiscardsAtAge<- *(Access)*, 16
DiscardsAtSize *(Access)*, 16
DiscardsAtSize<- *(Access)*, 16
Discount *(Bioeconomic)*, 67
Discount(), *69*
Discount<- *(Bioeconomic)*, 67
Disinvestment *(Bioeconomic)*, 67
Disinvestment(), *69*
Disinvestment<- *(Bioeconomic)*, 67
Dist *(Access)*, 16
Dist<- *(Access)*, 16
Distribution *(Effort)*, 202
Distribution<- *(Effort)*, 202
DLMDataDir, 196
DLMdiag *(Can)*, 97
dnormal *(model-helpers)*, 382
doHCR, 196
doIfreq, 197
Dom, 198
DomeExFleet, *58*, *59*, 199, *282*, *403*, *596–598*, *714*
doRec, 201
double_normal *(model-helpers)*, 382
DoubleNormal *(Selectivity-Models)*, 604
DoubleNormal(), *148*
DoubleNormalRetention *(Retention-Models)*, 582
DoubleNormalRetention(), *148*
DoubleNormalRetentionWeight *(Retention-Models)*, 582
DoubleNormalWeight *(Selectivity-Models)*, 604
DropDimension *(dimension-utils)*, 190
Dynamic *(unfished-class)*, 716
dynGet(), *275*

Efficiency *(Catchability)*, 101
Efficiency<- *(Catchability)*, 101
Effort, 202
effort, *202*, *203*, *205*, *207*, *208*, *252*, *256*, *524*
Effort(), *27*, *31*, *58*, *69*, *70*, *104*, *105*, *148*, *193*, *195*, *199*, *210*, *211*, *252*, *254–257*, *266*, *294*, *311*, *465*, *466*, *468*, *472*, *479*, *480*, *524*, *579*, *582*, *601*, *604*
effort-class, 210
Effort<- *(Effort)*, 202
EffortAllocation, 211
EffortAllocation(), *29*, *110*, *123*, *136*, *172*, *279*, *282*, *289*, *329–331*, *389*, *403*, *427*, *429*, *432*, *435*, *436*, *567*, *598*, *627*, *678*, *714*
EffortAllocation<- *(EffortAllocation)*, 211
EffortData, 213
effortdata, *159–161*, *213*

EffortData(), *108*, *109*, *133*, *135*, *214*, *320*, *321*
effortdata-class, 214
EffortObs, 215
effortobs, *127*, *173*, *215–217*, *417*, *420*, *525*
EffortObs(), *113*, *114*, *127*, *139*, *141*, *174*, *213*, *217*, *218*, *227*, *228*, *325*, *327*, *357*, *358*, *417*, *418*, *420*
effortobs-class, 217
EffType *(Advice)*, 26
EffType<- *(Advice)*, 26
Email *(OM-accessors)*, 430
Email<- *(OM-accessors)*, 430
Emp, 218
EmptyObject, 220
EmptyObject(), *141*, *337*, *518–520*, *525*, *529*, *531*, *537*
Equilibrium *(unfished-class)*, 716
equilibrium, *80*
equilibrium-class, 221
Error *(Access)*, 16
Error<- *(Access)*, 16
ESS *(Access)*, 16
ESS<- *(Access)*, 16
EstimateBeta, 222
EstimateBeta(), *80*, *81*, *312*, *324*, *327*, *617*
Example_datafile *(Atlantic_mackerel)*, 60
ExampleMPs, 223
ExampleMPs(), *136*, *282*, *403*, *598*, *627*, *714*
expandHerm *(Herm-int)*, 280
Exploitation *(Access)*, 16
Exploitation<- *(Access)*, 16
ExploitationData *(exploitationdata-class)*, 225
exploitationdata, *159–161*, *227*, *228*
exploitationdata-class, 225
ExploitationObs, 226
exploitationobs, *151*, *226*, *227*, *417*, *420*
ExploitationObs(), *113*, *114*, *139*, *141*, *217*, *218*, *227*, *228*, *325*, *327*, *357*, *358*, *417–420*
exploitationobs-class, 227
Extend, 228
Extend(), *49*, *234*, *344*, *370*, *405*, *530*, *726*
ExtendAges *(Extend)*, 228
ExtendAges(), *230*
ExtendAreas *(Extend)*, 228
ExtendAreas(), *230*
ExtendClasses *(Extend)*, 228
ExtendClasses(), *230*
ExtendSims *(Extend)*, 228
ExtendSims(), *66*, *67*, *100*, *229*, *230*, *245*, *570*, *571*
ExtendYears *(Extend)*, 228
ExtendYears(), *230*, *530*, *694*
extract-unfished, 231

F01 *(refpoints-accessors)*, 564
F01(), *87*
F_FMSY *(relative_ref)*, 568
F_FMSY(), *395*, *465*, *468*, *505*, *506*, *564*, *641*, *692*
F_timeseries *(FInteract)*, 244
FCrash *(refpoints-accessors)*, 564
FCrash(), *87*
FDead *(FInteract)*, 244
FDead(), *466*, *468*, *570*, *571*
Fease, 232
Fecundity, 233
fecundity, *233*, *235*, *236*, *355*, *480*, *668*, *672*
Fecundity(), *42*, *72*, *239–242*, *526*, *527*, *594*, *649*, *654*, *668–670*, *672*, *673*
fecundity-class, 239
Fecundity-Models, 241
Fecundity<- *(Fecundity)*, 233
FecundityAtAge *(Fecundity-Models)*, 241
FecundityAtLength *(Fecundity-Models)*, 241
FecundityAtWeight *(Fecundity-Models)*, 241
FecundityModels *(Fecundity-Models)*, 241
FecundityModels(), *43*, *234–236*, *239*, *240*, *267*, *382*, *384*, *596*
FecundityModelsAge *(Fecundity-Models)*, 241
FecundityModelsLength *(Fecundity-Models)*, 241
FecundityModelsWeight *(Fecundity-Models)*, 241
fetch.file.names *(iSCAM)*, 332

FilterTAC, 243
FilterTAC(), *317*
Final *(Depletion)*, 179
Final<- *(Depletion)*, 179
FInteract, 244
FInteract(), *466*, *468*
FitMovement, 246
FitMovement(), *94*, *634*
FitSP, 247
FitSP(), *638*, *640–643*, *687*, *688*, *690–692*
FitStockTargeting, 250
FitStockTargeting(), *82*, *263*, *264*, *666*, *667*
FixFleetNames *(BAMFleetNames)*, 64
FixFleetNames(), *64*
FlatE_Dom *(DecE_Dom)*, 175
FlatE_HDom *(DecE_Dom)*, 175
FlatE_NDom *(DecE_Dom)*, 175
Fleet, 252
fleet, *21*, *58*, *59*, *68–70*, *102*, *103*, *105*, *145*, *147*, *148*, *192*, *193*, *195*, *199*, *200*, *203*, *207*, *208*, *210*, *211*, *252–255*, *404*, *427*, *434*, *468*, *473*, *474*, *477–479*, *481*, *482*, *493–496*, *577–579*, *581*, *582*, *597*, *599–601*, *603*, *604*
Fleet(), *59*, *69*, *70*, *100*, *101*, *103–105*, *180*, *191*, *193*, *195*, *200*, *202*, *207*, *208*, *211*, *256*, *257*, *427*, *429*, *432*, *434*, *436*, *470*, *473*, *474*, *480*, *482*, *528*, *577*, *579*, *582*, *585*, *599*, *601*, *604*, *606*, *669*, *672*
fleet-class, 256
Fleet-class-objects *(DecE_Dom)*, 175
Fleet-legacy, *144*, *145*, *147*, *148*
Fleet-legacy-class, 258
Fleet<- *(Fleet)*, 252
FleetAllocation *(CatchFrac)*, 109
FleetAllocation(), *123*, *212*, *289*, *329*, *427*, *429*, *434*
FleetAllocation<- *(CatchFrac)*, 109
FleetNames *(name-accessors)*, 404
FleetNames(), *125*, *300*
FMax *(refpoints-accessors)*, 564
FMax(), *87*
FMed *(refpoints-accessors)*, 564
FMed(), *87*
FMSY *(MSY-accessors)*, 397
FMSY(), *571*
FMSYref, 260
FMSYref50 *(FMSYref)*, 260
FMSYref75 *(FMSYref)*, 260
FracOther *(Spatial)*, 630
FracOther<- *(Spatial)*, 630
FRetain *(FInteract)*, 244
FRetain(), *466*, *468*
FSPR *(refpoints-accessors)*, 564
FSPR(), *87*
FullComplianceImp, *135*, *136*, 262, *281*, *282*, *294*, *295*, *311*, *402*, *403*, *443*, *444*, *597*, *598*, *626*, *627*, *714*, *715*
furrr::future_map(), *612*
future::availableCores(), *612*
future::plan(), *612*

GenerateStockTargeting, 263
GenerateStockTargeting(), *82*, *251*, *438*, *666*, *667*
Generic_DecE *(DecE_Dom)*, 175
Generic_FlatE *(DecE_Dom)*, 175
Generic_Fleet *(DecE_Dom)*, 175
Generic_IncE *(DecE_Dom)*, 175
Generic_Obs, 265
GenHistEffort, 265
GenHistEffort(), *203*, *205*, *208*
GenMeanAtAge *(GenMeanAtLength)*, 266
GenMeanAtLength, 266
GenMeanAtWeight *(GenMeanAtLength)*, 266
GenMultiStockRecDevs, 267
GenMultiStockRecDevs(), *16*, *126*, *437*, *538*
GenRecDevs, 268
GenRecDevs(), *268*, *492*
GenResiduals, 270
GenResiduals(), *45*, *88*, *89*, *323–325*
get(), *275*
GetBAMFleetNames *(BAMFleetNames)*, 64
GetBAMFleetNames(), *64*
GetBAMOutput *(ImportBAM)*, 297
GetBAMOutput(), *64*, *128*, *129*, *300*, *301*
getBlow(), *721*
getclass, 271
getDataList, 272
getfirstlev, 272
GetMetaData, 273
getmov2, 274

getMP, 275
getnIVs, 275
getsel, 276
GetSSRefPoints, 276
ggplot2::ggplot(), *132*
ggplot2::ggsave(), *129–131*
Gompertz *(Length-at-Age-Models)*, 347
Gulf_blue_tilefish *(Atlantic_mackerel)*, 60

h2CR *(CR2h)*, 157
hconv, 277
Herm, 278
Herm(), *110*, *136*, *172*, *212*, *281*, *282*, *289*, *330*, *331*, *389*, *403*, *427*, *429*, *432*, *435*, *436*, *567*, *598*, *627*, *646*, *678*, *714*, *737*, *738*
Herm-int, 280
Herm<- *(OM-accessors)*, 430
HermOM, *110*, *136*, *172*, *212*, *279*, 281, *289*, *330*, *331*, *389*, *403*, *429*, *432*, *436*, *567*, *598*, *627*, *678*, *714*, *737*, *738*
Herring *(Albacore)*, 40
Hist, 283
hist, *65–67*, *78*, *79*, *83–87*, *89*, *92*, *99–101*, *109*, *130*, *158*, *160*, *171*, *203*, *205*, *207*, *208*, *211*, *231*, *234*, *235*, *240*, *244*, *245*, *279*, *283*, *288*, *289*, *312*, *328*, *330*, *331*, *361*, *369*, *388*, *391*, *397*, *398*, *404*, *415–418*, *426*, *430–432*, *438*, *455–457*, *460*, *464*, *467*, *468*, *470*, *472–482*, *484*, *485*, *487*, *489*, *491*, *493*, *495*, *496*, *498*, *499*, *501*, *513*, *554*, *555*, *565–567*, *569*, *590*, *591*, *622*, *623*, *647*, *653*, *654*, *704*, *705*, *742*
Hist(), *93*, *391*, *561*
Hist-class, *622*
hist-class, 284
Hist-legacy, *554*, *555*, *590*, *591*, *623*
Hist-legacy-class, 285
hist2, 288
HistoricalWeight, 288
HistoricalWeight(), *110*, *123*, *136*, *172*, *212*, *279*, *282*, *330*, *331*, *389*, *403*, *429*, *432*, *436*, *567*, *598*, *627*, *678*, *714*
HistoricalWeight<- *(HistoricalWeight)*, 288
HockeyStick *(SRRModels)*, 654
HockeyStick_RelRec *(SRRModels)*, 654
HockeyStickHCR, 290
HockeyStickHCR(), *315*, *317*, *689*, *691*, *692*

identical(), *337*
ilogit *(logit)*, 362
Imp, 292
imp, *21*, *145*, *149*, *262*, *292–295*, *311*, *427*, *434*, *442–444*, *529*, *715*
Imp(), *123*, *136*, *262*, *282*, *295*, *311*, *403*, *427*, *429*, *432*, *434*, *436*, *443*, *444*, *597*, *627*, *714*, *715*
imp-class, 294
Imp-class-objects *(Overages)*, 443
Imp-legacy, *144*, *145*, *149*
Imp-legacy-class, 295
Imp<- *(Imp)*, 292
ImportBAM, 297
ImportBAM(), *128*, *129*, *304*
ImportBAMData, 299
ImportBAMData(), *298*, *299*, *303*
ImportiSCAM, 302
ImportiSCAM(), *130*, *304*
ImportiSCAMData, 304
ImportSS, 305
ImportSS(), *303*, *304*, *309*, *310*
ImportSSData, 308
ImportSSReport, 309
ImportSSReport(), *131*, *132*, *276*, *277*, *305*, *309*
Imprecise_Biased *(Generic_Obs)*, 265
Imprecise_Unbiased *(Generic_Obs)*, 265
ImpSlot, 310
impslot, *262*, *293–295*, *310*, *311*, *443*, *444*, *529*, *530*, *715*
ImpSlot(), *262*, *293–295*, *443*, *444*, *530*, *715*
IncE_HDom *(DecE_Dom)*, 175
IncE_NDom *(DecE_Dom)*, 175
inch2mm *(unit-conversions)*, 717
IndexFitTable, 311
IndexFitTable(), *324*, *327*, *483*, *485*
IndexMPs, *44*, 313
IndexRate *(IndexMPs)*, 313
IndexRate(), *142*, *243*, *248*, *291*, *628*, *688*, *690*, *692*
IndexSelectivity, 318

IndexSelectivity<- *(IndexSelectivity)*, 318
IndexTarget *(IndexMPs)*, 313
IndexTarget(), *290*, *628*
IndicesData, 319
indicesdata, *159–161*, *301*, *319*, *320*
IndicesData(), *23*, *108*, *109*, *133*, *135*, *213*, *214*, *304*, *315*, *320*, *321*
indicesdata-class, 320
IndicesObs, 321
indicesobs, *22*, *23*, *37*, *102*, *103*, *106*, *127*, *173*, *311*, *321*, *324*, *325*, *351*, *417*, *420*, *530*, *531*
IndicesObs(), *23*, *37*, *106*, *113*, *114*, *127*, *139*, *141*, *174*, *217*, *218*, *227*, *228*, *312*, *318*, *320*, *326*, *327*, *351*, *357*, *358*, *417*, *418*, *420*, *617*
indicesobs-class, 326
Initial *(Depletion)*, 179
Initial<- *(Depletion)*, 179
initialize-methods, 327
Input, 327
Interactions *(catch_timeseries)*, 98
Interactions(), *102*, *103*, *466*, *468*
InterimAdvice, 328
InterimAdvice(), *110*, *123*, *136*, *172*, *212*, *279*, *282*, *289*, *298*, *303*, *306*, *331*, *389*, *403*, *428*, *429*, *432*, *435*, *436*, *567*, *598*, *627*, *678*, *714*
InterimAdvice<- *(InterimAdvice)*, 328
Interval, 330
Interval(), *110*, *136*, *172*, *212*, *279*, *282*, *288*, *289*, *330*, *369*, *389*, *403*, *427*, *429*, *432*, *436*, *567*, *598*, *627*, *678*, *714*
Interval<- *(Interval)*, 330
Investment *(Bioeconomic)*, 67
Investment(), *69*
Investment<- *(Bioeconomic)*, 67
iSCAM, 332
iSCAM2Data *(iSCAM2OM)*, 333
iSCAM2Data(), *303*
iSCAM2OM, *57*, 333, *333*
iSCAM2OM(), *302–304*
iSCAMcomps, 336
iSCAMinds, 336
isNewObject, 337
isNewObject(), *220*
isRel *(Selectivity)*, 599
isRel(), *601*
isRel<- *(Selectivity)*, 599

join_plots, 338
joinData, *117*, 338
joinData(), *146*, *147*
joinHist, *339*
joinHist *(checkMSE)*, 116
joinMSE, *339*
joinMSE *(checkMSE)*, 116
JoinYear, 339
JoinYear(), *549*

kg2_1000lb *(unit-conversions)*, 717
kg2lb *(unit-conversions)*, 717
kg2mt *(unit-conversions)*, 717
Kplot, 340

Lag_Data, 341
Landings *(catch_timeseries)*, 98
Landings(), *132*, *413*, *455*, *457*, *459*, *466–468*
Landings<- *(catch_timeseries)*, 98
LandingsAtAge *(Access)*, 16
LandingsAtAge<- *(Access)*, 16
LandingsAtSize *(Access)*, 16
LandingsAtSize<- *(Access)*, 16
LastHistYearInd *(DataHelpers)*, 168
LastHistYearInd(), *160*
LastTAC *(DataHelpers)*, 168
LastTAC(), *32*, *33*, *142*, *160*, *316*, *317*, *689*
Latitude *(OM-accessors)*, 430
Latitude<- *(OM-accessors)*, 430
lb2kg *(unit-conversions)*, 717
lb2mt *(unit-conversions)*, 717
ldim, 342
Length, 343
length, *192*, *234*, *343–345*, *355*, *370*, *406*, *485*, *578*, *600*, *668*, *672*, *721*, *726*, *727*
Length(), *42*, *71*, *75*, *236*, *240*, *242*, *349*, *350*, *373*, *377*, *379*, *407*, *408*, *411*, *412*, *471*, *522*, *526*, *532–535*, *540*, *541*, *548*, *585*, *594*, *606*, *668–670*, *672*, *673*, *721*, *722*, *727*, *728*, *730*, *731*, *733*, *737*
Length-at-Age-Models, 347
length-class, 349

Length<- *(Length)*, 343
LengthModels *(Length-at-Age-Models)*, 347
LengthModels(), *42*, *267*, *343*, *345*, *350*, *595*
LengthStructuredObs, *37*, *106*, *127*, *174*, 351, *403*
LH2OM, 352
LHdatabase, 354
LifeHistory *(Access)*, 16
LifeHistory<- *(Access)*, 16
LifeHistoryData *(lifehistorydata-class)*, 354
lifehistorydata, *159–161*, *356–358*
lifehistorydata-class, 354
LifeHistoryObs, 355
lifehistoryobs, *151*, *355–357*, *417*, *419*
LifeHistoryObs(), *113*, *114*, *139*, *141*, *217*, *218*, *227*, *228*, *325*, *327*, *357*, *358*, *417–420*
lifehistoryobs-class, 357
LimitType *(Advice)*, 26
LimitType<- *(Advice)*, 26
list(), *229*
list-array-conversion, 358
List2Array *(list-array-conversion)*, 358
ListBAMStocks *(ImportBAM)*, 297
ListParallelWorkers, 359
ListParallelWorkers(), *612*
load.iscam.files, *334*
load.iscam.files *(iSCAM)*, 332
LoadArgs, 360
Log, 361
Log(), *32*, *123*, *134*, *146*, *161*, *284*, *391*, *429*, *432*, *436*, *476*, *641*, *673*, *692*
logistic_50_95 *(model-helpers)*, 382
logit, 362
Longitude *(OM-accessors)*, 430
Longitude<- *(OM-accessors)*, 430
LorenzenMortalityLength *(NaturalMortality-Models)*, 411
LorenzenMortalityWeight *(NaturalMortality-Models)*, 411
Low_Effort_Non_Target *(DecE_Dom)*, 175
LTY *(PerformanceMetric)*, 445
lubridate::decimal_date(), *742*

Mackerel *(Albacore)*, 40
makeMOM, 363
makemov, 364, *619*
makemov2, 364, *620*
MakeNamedList, 365
makeRel, 366
makeTransparent, 368
ManagementScheduleTable, 368
ManagementScheduleTable(), *171*, *172*, *331*, *389*
Matrix::nearPD(), *82*
Maturity, 369
maturity, *355*, *369*, *371*, *373*, *486*, *668*, *672*
Maturity(), *42*, *72*, *236*, *240*, *242*, *376*, *377*, *379*, *526*, *534*, *540*, *541*, *578*, *594*, *600*, *668–670*, *672*, *673*, *737*
maturity-class, 376
Maturity-Models, 378
Maturity<- *(Maturity)*, 369
MaturityAtAge *(Maturity-Models)*, 378
MaturityAtLength, *154*
MaturityAtLength *(Maturity-Models)*, 378
MaturityAtWeight *(Maturity-Models)*, 378
MaturityModels, *267*
MaturityModels *(Maturity-Models)*, 378
MaturityModels(), *43*, *370*, *371*, *373*, *376*, *377*, *382*, *384*, *595*
MaturityModelsAge *(Maturity-Models)*, 378
MaturityModelsLength *(Maturity-Models)*, 378
MaturityModelsWeight *(Maturity-Models)*, 378
MaxAge *(Ages)*, 33
MaxAge<- *(Ages)*, 33
maxF *(OM-accessors)*, 430
maxF<- *(OM-accessors)*, 430
Maximum *(Effort)*, 202
Maximum<- *(Effort)*, 202
Mean *(Access)*, 16
Mean<- *(Access)*, 16
MeanAtAge *(Access)*, 16
MeanAtAge(), *193*, *579*, *601*
MeanAtAge<- *(Access)*, 16

MeanAtLength *(Access)*, 16
MeanAtLength(), *193*, *579*, *601*
MeanAtLength<- *(Access)*, 16
MeanAtWeight *(Access)*, 16
MeanAtWeight(), *579*, *601*
MeanAtWeight<- *(Access)*, 16
methods::new(), *36*, *182*, *239*, *349*, *376*, *410*, *419*, *433*, *637*, *652*, *672*, *730*
MinAge *(Ages)*, 33
MinAge<- *(Ages)*, 33
Misc *(Access)*, 16
Misc<- *(Access)*, 16
ML2D, 380
mm2inch *(unit-conversions)*, 717
MMSE-class *(MMSE-legacy-class)*, 380
MMSE-legacy-class, 380
Mode *(Effort)*, 202
Mode<- *(Effort)*, 202
Model *(Access)*, 16
Model(), *579*, *601*
model-helpers, 382
Model<- *(Access)*, 16
MOM-class *(MOM-legacy-class)*, 384
MOM-class-objects *(Albacore_TwoFleet)*, 41
MOM-legacy, *144*, *145*, *149*, *150*
MOM-legacy-class, 384
MOM_agg_fleets *(SS2MOM)*, 661
MortalityAtAge *(NaturalMortality-Models)*, 411
Movement *(Spatial)*, 630
Movement(), *497*
Movement<- *(Spatial)*, 630
movestockCPP, 386
movfit_Rcpp, 386
MPCalcsNAs, 387
MPFailureRate, 388
MPFailureRate(), *684*
MPs *(Access)*, 16
MPs<- *(Access)*, 16
MPStartYear, 388
MPStartYear(), *110*, *136*, *172*, *212*, *279*, *282*, *289*, *328*, *330*, *331*, *403*, *427*, *429*, *432*, *436*, *567*, *598*, *627*, *678*, *714*
MPStartYear<- *(MPStartYear)*, 388
MPtype, 390, *577*
mse, *65–67*, *99–101*, *109*, *124*, *125*, *158*, *160*, *171*, *203*, *205*, *207*, *208*, *211*, *244*, *245*, *279*, *283*, *288*, *289*, *312*, *328*, *330*, *331*, *361*, *369*, *388*, *393–398*, *404*, *413*, *415–418*, *426*, *430–432*, *455–457*, *460*, *464*, *466–470*, *472–474*, *476–482*, *484*, *485*, *487*, *489*, *491*, *493*, *495*, *496*, *498*, *499*, *501*, *504*, *506*, *513*, *549*, *555*, *566*, *567*, *569*, *570*, *640*, *683*, *694*, *704*, *706*, *708*, *742*
mse-class, 390
MSE-legacy, *554*, *555*, *591*
MSE-legacy-class, 391
MSE2Slick, 393
MSE2Slick(), *177*, *178*
MSEextra, 397
MSY-accessors, 397
MSYCalcs, 398
MSYDiscards *(MSY-accessors)*, 397
MSYLandings *(MSY-accessors)*, 397
MSYRefs *(MSY-accessors)*, 397
multiData, 400
multiDataS, 400
multidebug, 401
multiMSE, *53*, *54*, *366*, *663*, *680*
multiMSE *(SimulateMOM)*, 624
multiMSE(), *721*
MultiStockOM, *110*, *135*, *136*, *172*, *212*, *279*, *281*, *282*, *289*, *330*, *331*, *389*, 402, *429*, *432*, *436*, *567*, *596*, *598*, *626*, *627*, *678*, *713*, *714*

NO *(extract-unfished)*, 231
nAge *(Access)*, 16
Name *(Access)*, 16
name-accessors, 404
Name<- *(Access)*, 16
nArea *(Access)*, 16
NaturalMortality, 405
naturalmortality, *355*, *405–408*, *488*, *668*, *672*
NaturalMortality(), *42*, *71*, *410*, *411*, *535*, *536*, *594*, *668–670*, *672*, *673*, *738*
naturalmortality-class, 410
NaturalMortality-Models, 411
NaturalMortality<- *(NaturalMortality)*, 405

NaturalMortalityModels *(NaturalMortality-Models)*, 411
NaturalMortalityModels(), *43*, *267*, *405*, *406*, *408*, *410*, *411*, *595*
nComplex *(Access)*, 16
Needed, *577*
Needed *(Can)*, 97
NewPM, 413
NewPM(), *509*, *705*, *709*
nFleet *(Access)*, 16
NFref *(FMSYref)*, 260
NIL, 414
NOAA_plot, 415
NOAA_plot2 *(TradePlot)*, 699
NoFishing *(ReferenceMPs)*, 563
nReps *(OM-accessors)*, 430
nReps<- *(OM-accessors)*, 430
nSim *(Access)*, 16
nSim<- *(Access)*, 16
nStock *(Access)*, 16
nStock(), *95*, *96*, *415*
Number, 415
Number(), *67*, *132*, *466*, *468*, *485*
nYear *(OM-accessors)*, 430
nYear<- *(OM-accessors)*, 430

Obs, 416
obs, *21*, *36*, *37*, *99–101*, *106*, *112–114*, *127*, *137*, *139–141*, *144*, *145*, *150*, *151*, *173*, *174*, *203*, *207*, *208*, *217*, *218*, *227*, *228*, *299*, *301*, *318*, *325–327*, *351*, *357*, *358*, *416*, *418*, *427*, *434*, *536*, *537*, *597*
Obs(), *37*, *106*, *113*, *114*, *123*, *127*, *139*, *141*, *174*, *217*, *218*, *227*, *228*, *301*, *325*, *327*, *351*, *357*, *358*, *419*, *420*, *427*, *429*, *432*, *434*, *436*
obs-class, 419
Obs-class-objects *(Generic_Obs)*, 265
Obs-legacy, *144*, *145*, *150*, *151*, *227*, *356*, *418*, *420*
Obs-legacy-class, 420
Obs<- *(Obs)*, 416
OM, 425
om, *21*, *22*, *25*, *78*, *79*, *83–86*, *89*, *92*, *109*, *110*, *117*, *125*, *126*, *130*, *135*, *136*, *145*, *149–153*, *158–160*, *171*, *172*, *192*, *193*, *211*, *212*, *251–254*, *262*, *263*, *268*, *279*, *281*, *282*, *284*, *285*, *288*, *289*, *292–295*, *298–300*, *303*, *304*, *306*, *307*, *318*, *328*, *330*, *331*, *361*, *369*, *388*, *389*, *391*, *402–404*, *413*, *417*, *418*, *425*, *428–433*, *437*, *468*, *470*, *472–474*, *476–479*, *481*, *482*, *485*, *487*, *489*, *491*, *493*, *495*, *496*, *498*, *499*, *501*, *504*, *513*, *566*, *567*, *590*, *591*, *596–598*, *622*, *623*, *626*, *627*, *666–669*, *677*, *713*, *714*, *742*, *743*
OM(), *89*, *93*, *95*, *96*, *110*, *120*, *121*, *136*, *160*, *172*, *212*, *235*, *255*, *279*, *282*, *289*, *294*, *295*, *297*, *298*, *302–308*, *330*, *331*, *344*, *371*, *389*, *403*, *406*, *418*, *420*, *432–436*, *439*, *538*, *539*, *561*, *567*, *595*, *598*, *623*, *627*, *670*, *672*, *673*, *678*, *714*, *727*
OM-accessors, *110*, *172*, *212*, *289*, *330*, *331*, *389*, *429*, 430, *433*, *436*, *567*
OM-class *(OM-legacy-class)*, 436
om-class, 433
OM-class-objects *(testOM)*, 696
OM-legacy, *144*, *145*, *147*, *152*, *153*, *590*, *591*, *622*, *623*
OM-legacy-class, 436
OMControl, *428*, *436*, 437, *555*, *564*, *591*, *623*
OMdoc, 439
OMexample, 441
OMinit, 441
ourReefFish *(Atlantic_mackerel)*, 60
OverageImp, *136*, *262*, *294*, *295*, *311*, *403*, 442, *444*, *597*, *627*, *714*, *715*
Overages, 443

P10 *(PerformanceMetric)*, 445
P100 *(PerformanceMetric)*, 445
P50 *(PerformanceMetric)*, 445
Pars *(Access)*, 16
Pars(), *579*, *601*
Pars<- *(Access)*, 16
PartialSizeComplianceImp, *136*, *262*, *294*, *295*, *311*, *403*, *443*, 444, *597*, *627*, *714*, *715*
patchwork::wrap_plots(), *459*, *467*, *476*, *480*, *485*, *486*, *488*, *493*, *494*, *500*
Perfect_Imp *(Overages)*, 443
Perfect_Info *(Generic_Obs)*, 265

PerformanceMetric, 445
Period *(Access)*, 16
Period<- *(Access)*, 16
perrecruit, *85*, *86*, *221*, *222*, *566*
perrecruit-class, 446
PGK *(PerformanceMetric)*, 445
plot,data,ANY-method *(plot_data)*, 458
plot,MOM,missing-method *(plot.MOM)*, 450
plot.Data, 448
plot.Fleet *(plot.pars)*, 451
plot.Hist *(plot.pars)*, 451
plot.Imp *(plot.pars)*, 451
plot.MMSE, 449
plot.MOM, 450
plot.MSE, 450
plot.Obs *(plot.pars)*, 451
plot.OM *(plot.pars)*, 451
plot.pars, 451
plot.Stock *(plot.pars)*, 451
plot_comp, 455
plot_data, 458, *464*, *466*
plot_hist, *458*, *459*, 460, *480*
plot_mov *(simmov)*, 619
plot_schedule, 468, *478*, *481*, *486*, *487*, *489*, *494*, *496*, *501*
plot_SS2MOM *(SS2MOM)*, 661
plot_SS2OM *(SS2MOM)*, 661
PlotALK, 470
PlotALK(), *486*, *500*
PlotAWK, 471
PlotAWK(), *500*, *501*
PlotBiomass *(plot_hist)*, 460
PlotCatchability, 472
PlotCatchability(), *480–482*
PlotClosure, 473
PlotClosure(), *481*, *482*
PlotCompShift, 474
PlotCPUE *(plot_data)*, 458
PlotCPUE(), *485*
PlotData *(plot_data)*, 458
PlotDepletion, 475
PlotDepletion(), *499*, *500*
PlotDiscardMortality, 476
PlotDiscardMortality(), *469*, *470*, *481*, *482*
PlotDiscardMortalityCurve *(PlotDiscardMortality)*, 476
PlotDiscards *(plot_hist)*, 460
PlotDiscardsAtAge *(plot_comp)*, 455
PlotDiscardsAtSize *(plot_comp)*, 455
PlotDynamics *(plot_hist)*, 460
PlotEffort, 478
PlotEffort(), *467*, *473*, *482*
PlotEffortCurve *(PlotEffort)*, 478
PlotF *(plot_hist)*, 460
PlotFecundity, 480
PlotFecundity(), *469*, *470*, *499*
PlotFecundityCurve *(PlotFecundity)*, 480
PlotFleet, 481
plotFun, 483
PlotIndexFit, 483
PlotIndexFit(), *311*, *312*
PlotInteractions *(plot_hist)*, 460
PlotLandings *(plot_hist)*, 460
PlotLandingsAtAge *(plot_comp)*, 455
PlotLandingsAtSize *(plot_comp)*, 455
PlotLength, 485
PlotLength(), *469–471*, *478*, *499*
PlotLengthCurve *(PlotLength)*, 485
PlotMaturity, 486
PlotMaturity(), *469*, *470*, *499*
PlotMaturityCurve *(PlotMaturity)*, 486
plotmulti, 488
PlotNaturalMortality, 488
PlotNaturalMortality(), *469*, *470*, *499*
PlotNaturalMortalityCurve *(PlotNaturalMortality)*, 488
PlotNumber *(plot_hist)*, 460
plotOFL, 490
plotquant, 490
PlotRecDevs, 491, *499*
plotRel, 492
PlotRemovals *(plot_hist)*, 460
PlotRetention, 493
PlotRetention(), *469*, *470*, *481*, *482*
PlotRetentionCurve *(PlotRetention)*, 493
PlotSBiomass *(plot_hist)*, 460
PlotSelectivity, 494
PlotSelectivity(), *469*, *470*, *481*, *482*
PlotSelectivityCurve *(PlotSelectivity)*, 494
PlotSelectivityCurve(), *494*
PlotSpatial, 496

PlotSpatial(), *499*
PlotSProduction *(plot_hist)*, 460
PlotSRR, 497
PlotSRR(), *476*, *492*, *499*, *500*
PlotSRRCurve *(PlotSRR)*, 497
PlotSRRCurve(), *492*
PlotStock, 499
PlotSurvey *(plot_data)*, 458
PlotSurvey(), *485*
PlotVBiomass *(plot_hist)*, 460
PlotWeight, 500
PlotWeight(), *469*, *470*, *472*, *499*
PlotWeightCurve *(PlotWeight)*, 500
PlusGroup *(Ages)*, 33
PlusGroup<- *(Ages)*, 33
PM, *178*, *396*, *414*, 501, *507*, *509*, *513*, *705*, *707*, *709*
pm, *21*, *395*, *413*, *506*, *509*, *709*
PM(), *389*
pm-class, 506
PM-equations, *501*, *506*, 507, *507*
PM_AAVE *(PM)*, 501
PM_AAVY *(PM)*, 501
PM_AAVY(), *178*, *395*
PM_FFMSY *(PM)*, 501
PM_FFMSY(), *395*
PM_Kobe *(PM)*, 501
PM_Landings *(PM)*, 501
PM_LogYield *(PM)*, 501
PM_MinStatus *(PM)*, 501
PM_MinStatus(), *395*
PM_Rebuild *(PM)*, 501
PM_Red *(PM)*, 501
PM_RelYield *(PM)*, 501
PM_Removals *(PM)*, 501
PM_Safety *(PM)*, 501
PM_Safety(), *178*, *395*
PM_SBSBlim *(PM)*, 501
PM_SBSBMSY *(PM)*, 501
PM_SPSPlim *(PM)*, 501
PM_SPSPMSY *(PM)*, 501
PM_Stability *(PM)*, 501
PM_Status *(PM)*, 501
PM_Status(), *178*, *395*
PM_TACLimited *(PM)*, 501
PM_Yield *(PM)*, 501
PM_Yield(), *178*, *395*
PMLimit, 510
PMObj *(PMLimit)*, 510
PMobj-class, 512
PMYears, 513
PMYears(), *394–396*, *506*
PNOF *(PerformanceMetric)*, 445
popdynamics, *92*, *93*, *284*, *285*, *716*
popdynamics *(popdynamics-class)*, 514
popdynamics-class, 514
Populate, 514
Populate(), *43*, *59*, *73*, *181–183*, *200*, *234–236*, *239*, *240*, *247*, *344*, *345*, *349*, *350*, *370–373*, *376*, *377*, *405–408*, *410*, *411*, *497*, *527*, *528*, *533*, *534*, *536*, *537*, *539*, *544*, *545*, *547*, *548*, *596*, *631*, *632*, *634*, *637*, *638*, *646–649*, *652–655*, *682*, *726–728*, *730*, *731*
PopulateCatchability, 516
PopulateCatchability(), *59*, *102–104*, *200*, *515*
PopulateCatchObs, 517
PopulateCatchObs(), *515*, *529*, *537*
PopulateClosure, 518
PopulateClosure(), *59*, *200*, *253*, *254*
PopulateCompObs, 519
PopulateCompObs(), *137–141*, *515*, *537*
PopulateDepletion, 521
PopulateDepletion(), *43*, *73*, *515*, *546*, *596*
PopulateDiscardMortality, 522
PopulateDiscardMortality(), *59*, *192*, *193*, *200*, *515*
PopulateEffort, 523
PopulateEffort(), *59*, *200*, *203*, *205*, *208*, *515*
PopulateEffortObs, 525
PopulateEffortObs(), *515*, *537*
PopulateFecundity, 526
PopulateFecundity(), *43*, *73*, *515*, *546*, *596*
PopulateFleet, 527
PopulateFleet(), *58*, *59*, *199*, *200*, *253–255*, *257*, *515*, *538*, *539*, *544*
PopulateImp, 529
PopulateImp(), *515*, *530*
PopulateImpSlot, 529
PopulateImpSlot(), *293*, *310*, *529*
PopulateIndexObs, 530

PopulateIndexObs(), *515*, *537*
PopulateLength, 532
PopulateLength(), *42*, *72*, *515*, *527*, *534*, *536*, *546*, *548*, *595*
PopulateMaturity, 533
PopulateMaturity(), *43*, *72*, *515*, *527*, *546*, *595*
PopulateNaturalMortality, 535
PopulateNaturalMortality(), *43*, *72*, *515*, *546*, *595*
PopulateObs, 536
PopulateObs(), *515*, *518*, *520*, *525*, *531*
PopulateOM, 538
PopulateOM(), *117*, *126*, *136*, *149*, *150*, *152*, *153*, *282*, *298*, *303*, *304*, *308*, *403*, *429*, *436*, *468*, *515*, *547*, *598*, *627*, *714*
PopulateRetention, 539
PopulateRetention(), *59*, *200*, *515*, *578*, *579*
PopulateSelectivity, 541
PopulateSelectivity(), *59*, *200*, *515*, *600*, *601*
PopulateSpatial, 543
PopulateSpatial(), *43*, *73*, *515*, *546*, *595*
PopulateSRR, 544
PopulateSRR(), *43*, *73*, *515*, *546*, *596*, *647*
PopulateStock, 545
PopulateStock(), *42*, *43*, *72–74*, *499*, *500*, *515*, *527*, *528*, *533*, *534*, *536*, *538*, *539*, *544*, *545*, *548*, *594*, *596*
PopulateWeight, 547
PopulateWeight(), *42*, *72*, *515*, *527*, *534*, *546*, *595*
Porgy *(Albacore)*, 40
PPD, 549
PPD(), *694*
Pplot, 549
Pplot2, 550
Precise_Biased *(Generic_Obs)*, 265
Precise_Unbiased *(Generic_Obs)*, 265
predict.Rel *(makeRel)*, 366
predictLH *(LH2OM)*, 352
print.multiHist, 552
print.PlotFleetList, 552
print.PlotFleetList(), *482*
print.PlotStockList, 553
print.PlotStockList(), *500*
print.Rel *(makeRel)*, 366
print.RepList, 553
Prob *(Access)*, 16
Prob<- *(Access)*, 16
ProbStaying *(Spatial)*, 630
ProbStaying<- *(Spatial)*, 630
Project, 554
Project(), *205*, *283*, *330*, *391*, *432*, *438*, *439*, *589–591*, *608*, *683*, *684*, *705*, *706*
ProjectionYear *(DataHelpers)*, 168
ProjectionYear(), *160*
ProjectMOM *(SimulateMOM)*, 624
pStar *(OM-accessors)*, 430
pStar<- *(OM-accessors)*, 430
PWhisker, 555
pYear *(OM-accessors)*, 430
pYear<- *(OM-accessors)*, 430

qCV *(Catchability)*, 101
qCV<- *(Catchability)*, 101
qInc *(Catchability)*, 101
qInc(), *323*
qInc<- *(Catchability)*, 101
quantile_plot, 556

R0 *(SRR)*, 644
R0(), *89*, *90*
R0<- *(SRR)*, 644
R0conv *(hconv)*, 277
Random *(Access)*, 16
rDirichletMultinomial, 557
read.control.file *(iSCAM)*, 332
read.data.file *(iSCAM)*, 332
read.mcmc, *334*
read.mcmc *(iSCAM)*, 332
read.par.file *(iSCAM)*, 332
read.projection.file *(iSCAM)*, 332
read.report.file *(iSCAM)*, 332
RealFease, 558
Rec, *261*
Rec-class, 559
RecDevHist *(SRR)*, 644
RecDevHist<- *(SRR)*, 644
RecDevInit *(SRR)*, 644
RecDevInit<- *(SRR)*, 644
RecDevProj *(SRR)*, 644
RecDevProj<- *(SRR)*, 644
Red_snapper *(Atlantic_mackerel)*, 60

Reduce(), *561*
ReduceDims, 560
ReduceDims(), *66*, *83*, *89*, *100*, *232*, *245*, *554*, *569*, *570*, *623*
ReduceNSim, 561
ReduceNSim(), *561*
Ref *(Access)*, 16
Ref<- *(Access)*, 16
RefCV *(Access)*, 16
RefCV<- *(Access)*, 16
RefEffortYears *(RefSeason)*, 566
RefEffortYears<- *(RefSeason)*, 566
Reference *(Access)*, 16
reference, *79*, *83*, *87*, *284*, *285*, *391*, *565*
reference-class, 562
Reference<- *(Access)*, 16
ReferenceData *(referencedata-class)*, 562
referencedata, *159–161*
referencedata-class, 562
ReferenceMPs, *439*, 563
refFCurr *(ReferenceMPs)*, 563
refFMSY *(ReferenceMPs)*, 563
refFMSY50 *(ReferenceMPs)*, 563
refFMSY75 *(ReferenceMPs)*, 563
refMSY *(ReferenceMPs)*, 563
refMSY50 *(ReferenceMPs)*, 563
refMSY75 *(ReferenceMPs)*, 563
refpoints-accessors, 564
refpointsMSY, *84*, *85*, *398*, *562*, *563*, *570*, *623*
refpointsMSY-class, 566
RefSeason, 566
RefSeason(), *84*, *110*, *136*, *172*, *212*, *279*, *282*, *289*, *330*, *331*, *389*, *396*, *403*, *426*, *429*, *432*, *434*, *436*, *506*, *570*, *598*, *627*, *678*, *714*
RefSeason<- *(RefSeason)*, 566
Region *(OM-accessors)*, 430
Region<- *(OM-accessors)*, 430
Relations *(OM-accessors)*, 430
Relations<- *(OM-accessors)*, 430
relative_ref, 568
RelativeSize *(Spatial)*, 630
RelativeSize<- *(Spatial)*, 630
RelRecFun *(SRR)*, 644
RelRecFun<- *(SRR)*, 644
Removals *(catch_timeseries)*, 98
Removals(), *467*, *468*
Rename, 571
Rename(), *573*
Reorder, 572
Reorder(), *572*
Replace, 573
replic8, 574
Report, 575
ReqData, 576
Required, 576
Required(), *390*
Retention, 577
retention, *226*, *253*, *257*, *493*, *577–579*
Retention(), *27*, *29*, *31*, *32*, *58*, *69*, *70*, *104*, *105*, *148*, *152*, *192*, *193*, *195*, *199*, *208*, *211*, *253–255*, *257*, *522*, *539*, *540*, *581*, *582*, *585*, *601*, *604*, *606*
retention-class, 581
Retention-Models, 582
Retention<- *(Retention)*, 577
RetentionAtAge *(Retention-Models)*, 582
RetentionAtLength *(Retention-Models)*, 582
RetentionAtWeight *(Retention-Models)*, 582
RetentionKnifeEdgeAge *(Retention-Models)*, 582
RetentionKnifeEdgeLength *(Retention-Models)*, 582
RetentionModels *(Retention-Models)*, 582
RetentionModels(), *30*, *148*, *267*, *382*, *384*, *577–579*, *581*, *582*
RetentionModelsAge *(Retention-Models)*, 582
RetentionModelsLength *(Retention-Models)*, 582
RetentionModelsWeight *(Retention-Models)*, 582
Revenue *(Bioeconomic)*, 67
Revenue(), *69*
Revenue<- *(Bioeconomic)*, 67
Richards *(Length-at-Age-Models)*, 347
Ricker *(SRRModels)*, 654
Ricker_RelRec *(SRRModels)*, 654
Rockfish *(Albacore)*, 40

runCOSEWIC, 585
runInMP, 588
runMP, 589
runMSE, *56*, 589, *664*, *734*
runMSE(), *136*, *282*, *403*, *429*, *436*, *438*, *555*, *598*, *623*, *627*, *683*, *684*, *714*

S4SlotSizes, 592
SampleSize *(Access)*, 16
SampleSize<- *(Access)*, 16
Save, 593
saveRDS(), *593*
SB0 *(extract-unfished)*, 231
SB_SB0 *(relative_ref)*, 568
SB_SB0(), *465*, *476*
SB_SBMSY *(relative_ref)*, 568
SB_SBMSY(), *395*, *413*, *465*, *506*
SBiomass *(bio_timeseries)*, 65
SBiomass(), *416*, *466*, *468*, *570*
SBMSY *(MSY-accessors)*, 397
SBMSY(), *78*
SD *(Access)*, 16
SD<- *(Access)*, 16
sdconv(), *324*
SeasonalAllocation *(HistoricalWeight)*, 288
SeasonalAllocation(), *110*, *123*, *330*, *331*, *427*, *429*, *435*
SeasonalAllocation<- *(HistoricalWeight)*, 288
SeasonalSpatialExStock, 594, *596–598*
SeasonalSpatialExStock(), *43*, *73*
SeasonalSpatialOM, *110*, *136*, *172*, *212*, *279*, *282*, *289*, *330*, *331*, *389*, *403*, *429*, *432*, *436*, *567*, 596, *627*, *678*, *714*
SeasonOfYear *(DataHelpers)*, 168
Seasons *(OM-accessors)*, 430
Seasons<- *(OM-accessors)*, 430
Seed *(OM-accessors)*, 430
Seed<- *(OM-accessors)*, 430
select_MP, 598
Selectivity, 599
selectivity, *226*, *253*, *257*, *494*, *599–601*
Selectivity(), *27*, *29*, *31*, *32*, *58*, *69*, *70*, *104*, *105*, *148*, *152*, *192*, *193*, *195*, *199*, *208*, *211*, *253–255*, *257*, *522*, *540–542*, *579*, *582*, *585*, *603*, *604*, *606*

selectivity-class, 603
Selectivity-Models, 604
Selectivity<- *(Selectivity)*, 599
SelectivityAtAge *(Selectivity-Models)*, 604
SelectivityAtLength *(Selectivity-Models)*, 604
SelectivityAtWeight *(Selectivity-Models)*, 604
SelectivityKnifeEdgeAge *(Selectivity-Models)*, 604
SelectivityKnifeEdgeLength *(Selectivity-Models)*, 604
SelectivityModels *(Selectivity-Models)*, 604
SelectivityModels(), *30*, *148*, *267*, *382*, *384*, *599–601*, *603*, *604*
SelectivityModelsAge *(Selectivity-Models)*, 604
SelectivityModelsLength *(Selectivity-Models)*, 604
SelectivityModelsWeight *(Selectivity-Models)*, 604
Semelparous *(Maturity)*, 369
Semelparous(), *377*
Semelparous<- *(Maturity)*, 369
Sense, 607
SetMPArgs, 608
SetMPArgs(), *506*, *706–708*
SetSeasonalR0, 609
SetSeasonalR0(), *25*, *594–596*
setup, 610
SetupParallel, 611
SetupParallel(), *84*, *85*, *310*, *360*, *554*, *622*, *705*
Shift *(Access)*, 16
Shift<- *(Access)*, 16
show *(show-MSEtool)*, 613
show,advice-method *(show-methods)*, 613
show,ages-method *(show-methods)*, 613
show,catchability-method *(show-methods)*, 613
show,catchobs-method *(show-methods)*, 613
show,compobs-method *(show-methods)*, 613
show,Data-method *(show-MSEtool)*, 613

show,data-method *(show-methods)*, 613
show,depletion-method *(show-methods)*, 613
show,discardmortality-method *(show-methods)*, 613
show,effort-method *(show-methods)*, 613
show,effortobs-method *(show-methods)*, 613
show,equilibrium-method *(show-methods)*, 613
show,exploitationobs-method *(show-methods)*, 613
show,fecundity-method *(show-methods)*, 613
show,fleet-method *(show-methods)*, 613
show,Hist-method *(show-MSEtool)*, 613
show,hist-method *(show-methods)*, 613
show,indicesobs-method *(show-methods)*, 613
show,length-method *(show-methods)*, 613
show,lifehistoryobs-method *(show-methods)*, 613
show,maturity-method *(show-methods)*, 613
show,MMSE-method *(show-MSEtool)*, 613
show,MOM-method *(show-MSEtool)*, 613
show,MSE-method *(show-MSEtool)*, 613
show,mse-method *(show-methods)*, 613
show,naturalmortality-method *(show-methods)*, 613
show,obs-method *(show-methods)*, 613
show,OM-method *(show-MSEtool)*, 613
show,om-method *(show-methods)*, 613
show,perrecruit-method *(show-methods)*, 613
show,pm-method *(show-methods)*, 613
show,PMobj-method, 614
show,popdynamics-method *(show-methods)*, 613
show,Rec-method, 614
show,refpointsMSY-method *(show-methods)*, 613
show,retention-method *(show-methods)*, 613
show,selectivity-method *(show-methods)*, 613
show,spatial-method *(show-methods)*, 613
show,srr-method *(show-methods)*, 613
show,stock-method *(show-methods)*, 613
show,stocktargeting-method *(show-methods)*, 613
show,stocktransition-method *(show-methods)*, 613
show,tunemp-method *(tunemp-class)*, 707
show,weight-method *(show-methods)*, 613
show-methods, 613
show-MSEtool, 613
SIL, 614
simCAA, 615
simCAL, 616
SimControl, 617
SimControl(), *438*, *439*, *623*
simmov, *364*, 619
simmov2, *364*, *365*, 620
Simulate, 622
Simulate(), *89*, *107*, *112*, *113*, *123*, *126*, *131–133*, *139*, *141*, *204*, *205*, *213*, *231*, *283*, *284*, *302*, *303*, *319*, *324–326*, *432*, *437*, *439*, *472*, *475*, *476*, *554*, *555*, *589–591*, *617*, *618*, *623*
simulate.Rel *(makeRel)*, 366
SimulatedData, 624
SimulateMOM, 624
Simulation_1 *(Atlantic_mackerel)*, 60
SingleStockOM, *110*, *135*, *136*, *172*, *212*, *262*, *279*, *282*, *289*, *330*, *331*, *389*, *402*, *403*, *429*, *432*, *436*, *567*, *590*, *591*, *596*, *598*, 626, *678*, *713*, *714*
Size *(Access)*, 16
Size(), *294*, *311*
Size<- *(Access)*, 16
SketchFun, 627
Slick::App(), *395*, *396*
Slick::Preset(), *396*
Slick::Slick(), *393*, *395*, *396*
SmoothSeries, 628
SmoothSeries(), *314*, *317*
smoothy, 629
Snapper *(Albacore)*, 40
Sole *(Albacore)*, 40

Source *(OM-accessors)*, 430
Source<- *(OM-accessors)*, 430
SP0 *(extract-unfished)*, 231
SP0(), *89*, *90*
SP_SP0 *(relative_ref)*, 568
SP_SP0(), *465*
SP_SPMSY *(relative_ref)*, 568
SP_SPMSY(), *465*, *506*
Spatial, 630
spatial, *154*, *246*, *247*, *355*, *630*, *632*, *634*, *669*, *672*
Spatial(), *42*, *72*, *78*, *94*, *247*, *543*, *594*, *596*, *637*, *638*, *669*, *670*, *672*, *673*, *738*
spatial-class, 637
Spatial<- *(Spatial)*, 630
SpawnLag *(SRR)*, 644
SpawnLag<- *(SRR)*, 644
SpawnTimeFrac *(SRR)*, 644
SpawnTimeFrac<- *(SRR)*, 644
SPControl, 638
SPControl(), *248–250*, *643*, *689*, *690*, *692*
Species *(Stock)*, 667
Species<- *(Stock)*, 667
SPEstimates, 640
SPEstimates(), *690*, *692*
SPFrom *(SRR)*, 644
SPFrom<- *(SRR)*, 644
SpictControl, 641
SpictControl(), *248–250*, *640*, *688–690*, *692*
Splot, 644
SPMSY *(MSY-accessors)*, 397
Sponsor *(OM-accessors)*, 430
Sponsor<- *(OM-accessors)*, 430
SPRcrash *(refpoints-accessors)*, 564
SPRcrash(), *87*
SPRMSY *(MSY-accessors)*, 397
SProduction *(bio_timeseries)*, 65
SProduction(), *416*, *466*, *468*, *570*
SRalphaconv *(hconv)*, 277
SRbetaconv *(hconv)*, 277
SRR, 644
srr, *306*, *355*, *491*, *492*, *644*, *646*, *647*, *649*, *668*, *672*
SRR(), *42*, *72*, *93*, *221*, *235*, *236*, *240*, *282*, *447*, *499*, *544*, *545*, *594*, *652*, *654–656*, *658*, *668–670*, *672*, *673*, *737*, *738*
srr-class, 652
SRR<- *(SRR)*, 644
SRRAlphaBeta *(SRRSteepness)*, 657
SRRModels, 654
SRRModels(), *43*, *596*, *646*, *647*, *649*, *652*, *654*, *658*, *672*
SRRSteepness, 657
SRRSteepness(), *646*, *652*, *655*, *656*
SS2Data, 658, *664*
SS2DataMOM, 660, *664*
SS2MOM, *54*, 661, *661*
SS2OM, *57*, *660*
SS2OM *(SS2MOM)*, 661
SS_output, *659*, *660*, *663*
SSBrefplot, 665
SSMOM2OM *(SS2MOM)*, 661
StandardizeEffort, 665
StandardizeEffort(), *125*, *250*, *251*, *264*, *538*
Stat *(Access)*, 16
Stat<- *(Access)*, 16
Stats *(Access)*, 16
stats::loess(), *628*
stats::nlminb(), *249*, *639*, *640*
stats::optimize(), *79*
stats::setNames(), *365*
Stats<- *(Access)*, 16
Stock, 667
stock, *25*, *33–36*, *41–43*, *71*, *73*, *89*, *145*, *153*, *154*, *179–182*, *200*, *233–236*, *239*, *254*, *257*, *343–345*, *349*, *361*, *369–371*, *373*, *376*, *404–407*, *410*, *426*, *434*, *468*, *470*, *472–474*, *476*, *478*, *479*, *481*, *482*, *485*, *487*, *489*, *491*, *494*, *496*, *498*, *499*, *501*, *594*, *596*, *597*, *626*, *630–632*, *634*, *637*, *644*, *646*, *647*, *649*, *652*, *667*, *669*, *670*, *678*, *714*, *725–728*, *730*, *737*
Stock(), *35*, *43*, *58*, *73*, *92*, *93*, *96*, *180*, *181*, *183*, *199*, *236*, *242*, *254*, *345*, *349*, *373*, *379*, *408*, *412*, *426*, *429*, *432*, *434*, *436*, *470*, *476*, *499*, *500*, *528*, *546*, *547*, *596*, *634*, *638*, *649*, *654*, *672*, *673*, *728*, *733*
stock-class, 671
Stock-class-objects *(Albacore)*, 40
Stock-legacy, *144*, *145*, *153*

Stock-legacy-class, 673
Stock<- *(Stock)*, 667
StockNames *(name-accessors)*, 404
StockNames(), *457*, *465*, *469*, *470*, *472–474*, *476*, *478*, *479*, *481*, *482*, *486*, *487*, *489*, *491*, *494*, *496*, *498*, *499*, *501*
StockTargeting, 676
stocktargeting, *427*, *435*, *677*
StockTargeting(), *427*, *429*, *435*, *436*
stocktargeting-class, 677
StockTargetingLambda *(Effort)*, 202
StockTargetingLambda<- *(Effort)*, 202
stocktransition, *278*, *279*, *281*, *282*, *427*, *435*, *738*
stocktransition-class, 678
str, *613*
STY *(PerformanceMetric)*, 445
Sub, 679, *680*, *681*
SubCpars, *679*, 680, *681*
SubCpars,MOM-method *(SubCpars)*, 680
SubCpars,OM-method *(SubCpars)*, 680
SubOM, *679*, *680*, 681
Subset, 681
subsetHerm *(Herm-int)*, 280
SummarizeConvergence, 683
SummarizeConvergence(), *388*
summary,Data-method, 684
summary,MMSE-method, 685
summary,MSE-method, 685
SumOverAge, 686
SumOverArea *(SumOverAge)*, 686
SumOverClass *(SumOverAge)*, 686
SumOverFleet *(SumOverAge)*, 686
SumOverStock *(SumOverAge)*, 686
SumOverYear *(SumOverAge)*, 686
SurplusProduction, 687
SurplusProduction(), *247*, *250*, *306*, *638–643*
Survey *(Access)*, 16
Survey(), *459*
Survey<- *(Access)*, 16

TAC, 692
TAC<- *(Advice)*, 26
TACfilter, 693
TACImp *(Imp)*, 292
TACImp(), *294*, *311*
TACImp<- *(Imp)*, 292
TACs, 694
TACType *(Advice)*, 26
TACType<- *(Advice)*, 26
TACUnit *(Advice)*, 26
TACUnit<- *(Advice)*, 26
Target_All_Fish *(DecE_Dom)*, 175
Targeting *(Effort)*, 202
Targeting<- *(Effort)*, 202
Targeting_Small_Fish *(DecE_Dom)*, 175
Taxa_Table, 695
TEG, 695
tempfile(), *593*
testOM, 696
Theta *(Effort)*, 202
Theta(), *31*
Theta<- *(Effort)*, 202
Thresh_tab, 696
timeseries, *284*, *285*, *390*, *391*
timeseries-class, 697
Timing *(Access)*, 16
tinyErr, 698
tinyErr,OM-method *(tinyErr)*, 698
Toothfish *(Albacore)*, 40
Tplot *(TradePlot)*, 699
Tplot2 *(TradePlot)*, 699
Tplot3 *(TradePlot)*, 699
TradePlot, 699
TripsScalar *(Effort)*, 202
TripsScalar(), *28*, *31*, *210*
TripsScalar<- *(Effort)*, 202
TruncSD *(Access)*, 16
TruncSD<- *(Access)*, 16
try(), *275*
tune_MP, 701
TuneConstraint *(TuneObjective)*, 708
TuneConstraint(), *705*, *707*, *710*
TuneControl, 702
TuneControl(), *705–708*, *710*, *711*
TuneMP, 704
tunemp, *706*, *710–712*
TuneMP(), *316*, *317*, *413*, *414*, *439*, *608*, *692*, *702*, *704*, *707–709*, *711*, *712*
tunemp-class, 707
TuneObjective, 708
TuneObjective(), *705*, *707*, *710*
TuneTable, 710
TuneTable(), *706–708*
Turing, 712

TuringMOM *(Turing)*, 712
TwoFleetOM, *110*, *135*, *136*, *172*, *212*, *262*, *279*, *282*, *289*, *330*, *331*, *389*, *402*, *403*, *429*, *432*, *436*, *567*, *596*, *598*, *626*, *627*, *678*, 713
Type *(Access)*, 16
Type<- *(Access)*, 16

UnderageImp, *136*, *262*, *294*, *295*, *311*, *403*, *443*, *444*, *597*, *627*, *714*, 715
Unfished *(unfished-class)*, 716
unfished, *284*, *285*, *391*
unfished *(unfished-class)*, 716
unfished-class, 716
UnfishedDist *(Spatial)*, 630
UnfishedDist(), *497*
UnfishedDist<- *(Spatial)*, 630
unit-conversions, 717
Units *(Access)*, 16
Units<- *(Access)*, 16
updateMSE *(checkMSE)*, 116
UpdateObject, 718
UpdateSlots *(UpdateObject)*, 718
UpdateSlots(), *718*
Uses, 719

validcpars, 719
validcpars(), *384*, *437*
ValidUnits, 721
ValidUnits(), *34–36*, *236*, *240*, *306*, *309*, *343*, *345*, *350*, *405*, *408*, *410*, *411*, *726*, *728*, *731*
Value *(Access)*, 16
Value<- *(Access)*, 16
VBiomass *(bio_timeseries)*, 65
VBiomass(), *466*, *468–470*
VOI, 722
VOI2, 723
VOIplot, 723
vonBert *(Length-at-Age-Models)*, 347
VPA2OM, *62*
VPA2OM *(Assess2OM)*, 55

Weight, 725
weight, *355*, *370*, *371*, *377*, *500*, *578*, *600*, *668*, *672*, *721*, *725*, *726*, *728*
Weight(), *42*, *71*, *236*, *240*, *242*, *373*, *374*, *377*, *379*, *412*, *472*, *526*, *534*, *540*, *541*, *548*, *585*, *594*, *606*, *668–670*, *672*, *673*, *721*, *722*, *730–733*, *737*
weight-class, 730
Weight-Models, 732
Weight<- *(Weight)*, 725
WeightatAge *(Weight-Models)*, 732
WeightatLength *(Weight-Models)*, 732
WeightatMeanLength *(Weight-Models)*, 732
WeightFleetRetained *(Fleet)*, 252
WeightFleetRetained(), *253*, *254*
WeightFleetRetained<- *(Fleet)*, 252
WeightFleetSelected *(Fleet)*, 252
WeightFleetSelected(), *254*
WeightFleetSelected<- *(Fleet)*, 252
WeightModels *(Weight-Models)*, 732
WeightModels(), *42*, *267*, *382*, *384*, *595*, *725*, *727*, *728*, *730*, *731*
WHAM2OM, *57*, 733
wormplot, 735
WrappedNormal, 736
WrappedNormal(), *25*, *609*, *610*
WrasseExStock, 737
WrasseFemaleExStock, *281*, *282*
WrasseFemaleExStock *(WrasseExStock)*, 737
WrasseMaleExStock, *281*, *282*
WrasseMaleExStock *(WrasseExStock)*, 737
writeCSV, 738

XL2Data, 739
XL2Fleet, 739
XL2OM, 740
XL2Stock, 741

YearLH *(Access)*, 16
YearLH<- *(Access)*, 16
Years, 742
Years(), *428*, *429*, *432*, *434*, *436*, *549*
Years<- *(Years)*, 742
Yield *(PerformanceMetric)*, 445