Package: BLMEngineInR (via r-universe)

May 14, 2026

Type Package

Version 0.1.7

Date 2025-09-03

Title Biotic Ligand Model Engine

Description A chemical speciation and toxicity prediction model for the toxicity of metals to aquatic organisms. The Biotic Ligand Model (BLM) engine was originally programmed in 'PowerBasic' by Robert Santore and others. The main way the BLM can be used is to predict the toxicity of a metal to an organism with a known sensitivity (i.e., it is known how much of that metal must accumulate on that organism's biotic ligand to cause a physiological effect in a certain percentage of the population, such as a 20% loss in reproduction or a 50% mortality rate). The second way the BLM can be used is to estimate the chemical speciation of the metal and other constituents in water, including estimating the amount of metal accumulated to an organism's biotic ligand during a toxicity test. In the first application of the BLM, the amount of metal associated with a toxicity endpoint, or regulatory limit will be predicted, while in the second application, the amount of metal is known and the portions of that metal that exist in various forms will be determined. This version of the engine has been re-structured to perform the calculations in a different way that will make it more efficient in R, while also making it more flexible and easier to maintain in the future. Because of this, it does not currently match the desktop model exactly, but we hope to improve this comparability in the future.

License Apache License (>= 2)

URL https://www.windwardenv.com/biotic-ligand-model/

Encoding UTF-8

Language en-US

LazyData true

Imports methods, openxlsx, Rcpp (>= 1.0.10), utils

LinkingTo Rcpp, RcppArmadillo

Suggests testthat (>= 3.0.0), withr

Config/testthat/edition 3

RoxygenNote 7.3.3

Depends R (>= 3.5)

Config/pak/sysreqs libicu-dev

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

Date/Publication 2025-09-10 21:06:06 UTC

RemoteUrl https://github.com/windwardenv/blmengineinr

RemoteRef HEAD

RemoteSha 002364e93e9e177ac2d8d2616222807d251a9b11

Contents

`All_NIST20170203_reactions` *All NIST_20170203.dbs reactions*

Description

A large problem using the WHAM V "NIST_20170203.dbs" thermodynamic database. This is the thermodynamic database used in some of the newer Windows BLM parameter files, including the Environment and Climate Change Canada (ECCC) copper Federal Water Quality Guideline.

Usage

```
All_NIST20170203_reactions
```

Format

An object of class `list` of length 24.

`All_WATER23_reactions` *All WATER23.dbs reactions*

Description

A large problem using the WHAM V "WATER23.dbs" thermodynamic database. This is the thermodynamic database used in most of the Windows BLM parameter files, including the United States Environmental Protection Agency's (USEPA) final acute value (FAV).

Usage

```
All_WATER23_reactions
```

Format

An object of class `list` of length 24.

`BlankProblem` *Make a blank input problem list object*

Description

Make a blank input problem list object

Usage

```
BlankProblem()
```

Value

A list object with a template for defining the chemical problem for the 'BLMEngineInR' functions. Each element in the list is a vector, list, or data.frame object grouping related parameters together. See 'str(BlankProblem())' for the structure and names of the list object.

See Also

Other problem manipulation functions: `Components`, `CriticalValues`, `InLabs`, `InVars`, `MassCompartments`, `Phases`, `SpecialDefs`, `Species`

Examples

```
# Make a blank problem:
ThisProblem = BlankProblem()
str(ThisProblem)

# Add Water as a mass compartment
ThisProblem = AddMassCompartments(
  ThisProblem,
  MassName = "Water",
  MassAmt = 1,
  MassUnit = "L"
)

# Add temperature and pH variables:
ThisProblem = AddInVars(ThisProblem,
                        InVarName = c("Temp", "pH"),
                        InVarMCName = rep("Water", 2),
                        InVarType = c("Temperature", "pH"))

# Add CO3 as a component:
ThisProblem = AddInComps(
  ThisProblem,
  InCompName = "CO3",
  InCompCharge = -2,
  InCompMCName = "Water",
  InCompType = "MassBal",
```

```
  InCompActCorr = "Debye"
)

# Add reactions (using SpecCompNames and SpecCompStoichs for arguments):
# HCO3 = H + CO3     logK = 10.329
# H2CO3 = 2*H + CO3  logK = 10.329 + 6.352 = 16.681
ThisProblem = AddSpecies(
  ThisProblem,
  SpecName = c("HCO3", "H2CO3"),
  SpecMCName = "Water",
  SpecActCorr = "Debye",
  SpecCompNames = list(c("H", "CO3"), c("H", "CO3")),
  SpecCompStoichs = list(c(1, 1), c(1, 2)),
  SpecLogK = c(10.329, 16.681),
  SpecDeltaH = c(-14997.55155, -24166.23162),
  SpecTempKelvin = 298.1514609
)
# ...ThisProblem now simulates carbonate reactions.

# Add major ions and copper as components
ThisProblem = AddInComps(
  ThisProblem,
  InCompName = c("Cu", "Ca", "Mg", "Na", "K", "SO4", "Cl", "S"),
  InCompCharge = c(2, 2, 2, 1, 1, -2, -1, -2),
  InCompMCName = "Water",
  InCompType = "MassBal",
  InCompActCorr = "Debye"
)

# Add reactions (using SpecEquation as an argument):
ThisProblem = AddSpecies(
  ThisProblem,
  SpecEquation = c(
    "CuOH = 1 * Cu + 1 * OH",
    "Cu(OH)2 = 1 * Cu + 2 * OH",
    "CuSO4 = 1 * Cu + 1 * SO4",
    "CuCl = 1 * Cu + 1 * Cl",
    "CuCO3 = 1 * Cu + 1 * CO3",
    "Cu(CO3)2 = 1 * Cu + 2 * CO3",
    "CuHCO3 = 1 * Cu + 1 * CO3 + 1 * H",
    "CaHCO3 = 1 * Ca + 1 * H + 1 * CO3",
    "CaCO3 = 1 * Ca + 1 * CO3",
    "CaSO4 = 1 * Ca + 1 * SO4",
    "MgHCO3 = 1 * Mg + 1 * H + 1 * CO3",
    "MgCO3 = 1 * Mg + 1 * CO3",
    "MgSO4 = 1 * Mg + 1 * SO4"
  ),
  SpecMCName = "Water",
  SpecActCorr = "Debye",
  SpecLogK = c(6.48, 11.78, 2.360, 0.400, 6.750, 9.920, 14.620,
               11.44, 3.22, 2.30, 11.4, 2.98, 2.37),
  SpecDeltaH = c(0, 0, 8844.385918, 6738.57975, 0, 0, 0,
                 -3664.102737, 14951.22381, 6949.160364, -11666.16619,
```

```
                11413.46945, 19163.83616),
  SpecTempKelvin = 298.15
)

# Add BL mass compartment:
ThisProblem = AddMassCompartments(
  ThisProblem,
  MassName = "BL",
  MassAmt = 1,
  MassUnit = "kg wet"
)

# Add BL1 defined component:
ThisProblem = AddDefComps(ThisProblem,
                          DefCompName = "BL1",
                          DefCompFromNum = 1.78E-5,
                          DefCompCharge = -1,
                          DefCompMCName = "BL",
                          DefCompType = "MassBal",
                          DefCompActCorr = "None",
                          DefCompSiteDens = 3E-5)

# Add biotic ligand reactions (using SpecStoich):
spec_that_bind = c("Cu", "CuOH", "Ca", "Mg", "H", "Na")
temp_stoich_mat = ThisProblem$SpecStoich[spec_that_bind, ]
rownames(temp_stoich_mat) = paste0("BL1-", spec_that_bind)
temp_stoich_mat[, "BL1"] = 1L
temp_stoich_mat["BL1-CuOH", c("H","OH")] = c(-1L, 0L)
ThisProblem = AddSpecies(
  ThisProblem,
  SpecName = paste0("BL1-", spec_that_bind),
  SpecMCName = "BL",
  SpecActCorr = "None",
  SpecLogK = c(7.4, -1.3, 3.6, 3.6, 5.4, 3.0),
  SpecDeltaH = ThisProblem$Spec$DeltaH[match(spec_that_bind, ThisProblem$Spec$Name)],
 SpecTempKelvin = ThisProblem$Spec$TempKelvin[match(spec_that_bind, ThisProblem$Spec$Name)],
  SpecStoich = temp_stoich_mat
)

# Add special definitions for the toxic species:
ThisProblem = AddSpecialDefs(
  ThisProblem,
  Value = c("BL1","Cu","BL1-Cu","BL1-CuOH"),
  SpecialDef = c("BL","Metal","BLMetal","BLMetal")
)
# ...ThisProblem now simulates copper toxicity in the absence of organic matter.

# Add DOC: first we add DOC and HA input variables...
ThisProblem = AddInVars(
  ThisProblem,
  InVarName = c("DOC", "HA"),
  InVarMCName = "Water",
  InVarType = c("WHAM-HAFA", "PercHA")
```

```
)

# ...then we add a WHAM version as a special definition.
ThisProblem = AddSpecialDefs(
  ThisProblem,
  Value = "V",
  SpecialDef = "WHAM"
)

# As a finishing touch, we already know our critical values:
ThisProblem = AddCriticalValues(
  ThisProblem,
  CATab = data.frame(
    CA = c(0.05541, 0.03395),
    Species = c("Ceriodaphnia dubia","FAV"),
    TestType = "Acute",
    Duration = c("48h","DIV=2.00"),
    Lifestage = c("Neonate (<24h)","ACR=3.22"),
    Endpoint = c("Mortality","FAV"),
    Quantifier = c("EC50; LC50", "NA"),
   References = c("Gensemer et al. 2002; Hyne et al. 2005; 2002; 2003; Van Genderen et al. 2007",
                  "US EPA 2007"),
    Miscellaneous = c("SMEA calculated by median", NA)
  )
)
# ThisProblem can now calculate the Cu WQC

# Now what about CO2 dissolving from the atmosphere?
ThisProblem = AddPhases(
  ThisProblem,
  PhaseName = "CO2(g)",
  PhaseCompNames = list(c("CO3", "H")),
  PhaseCompStoichs = list(c(1, 2)),
  PhaseLogK = -1.5,
  PhaseDeltaH = 0,
  PhaseTempKelvin = 0,
  PhaseMoles = 10^-3.2
)

# Actually, scratch that - no CO2 dissolution
ThisProblem = RemovePhases(ThisProblem, "CO2(g)")

# Actually, I don't want C. dubia in this parameter file.
ThisProblem = RemoveCriticalValues(ThisProblem, 1)

# I know we usually have sulfide in there, but it's really not doing anything
# for us, so let's remove that.
ThisProblem = RemoveComponents(ThisProblem, "S")

# I kinda want to try this with WHAM VII instead of V...
ThisProblem = RemoveSpecialDefs(ThisProblem, SpecialDefToRemove = "WHAM")
ThisProblem = AddSpecialDefs(ThisProblem, Value = "VII", SpecialDef = "WHAM")
```

```
# Now what if I wanted to make this just a simulation of organic matter binding,
# sans biotic ligand?
ThisProblem = RemoveMassCompartments(ThisProblem, MCToRemove = "BL")
```

`BlankWHAM` *Make a blank WHAM parameter list object*

Description

Make a blank WHAM parameter list object

Usage

```
BlankWHAM()
```

Value

A list object with a template for defining the organic matter binding in a chemical problem for the 'BLMEngineInR' functions. Each element in the list is a vector, matrix, or data.frame object grouping related parameters together. See 'str(BlankWHAM())' for the structure and names of the list object.

`BLM` *Run the Biotic Ligand Model*

Description

'BLM' will run the Windward Environmental Biotic Ligand Model (BLM) with the provided parameter file, input file, and options.

Usage

```
BLM(
  ParamFile = character(),
  InputFile = character(),
  ThisProblem = list(),
  AllInput = list(),
  DoTox = logical(),
  CritAccumIndex = 1L,
  CritAccumValue = numeric(),
  QuietFlag = c("Very Quiet", "Quiet", "Debug"),
  ConvergenceCriteria = 1e-04,
  MaxIter = 100L
)
```

Arguments

ParamFile	(optional) The path and file name of the parameter file
InputFile	(optional) The path and file name of the chemistry input file
ThisProblem	(optional) A problem list object, such as returned by 'DefineProblem'.
AllInput	(optional) An input chemistry list object, such as returned by 'GetData'.
DoTox	Should this be a speciation (TRUE) or toxicity (FALSE) run? In a speciation run, the total metal is input and the free metal and metal bound to the biotic ligand is calculated. In a toxicity run, the critical accumulation is input and the free and total metal concentrations that would result in that amount bound to the biotic ligand is calculated.
CritAccumIndex	(unnecessary unless DoTox = TRUE) The index of the critical accumulation value in the parameter file critical accumulation table. If this is a single value, then it will be applied to all observations. If it is a vector with the same length as the inputs, then each value given will be used for the corresponding observation. Ignored if 'CritAccumValue' is given.
CritAccumValue	(unnecessary unless DoTox = TRUE) The critical accumulation value to use, in nmol/gw. If this is a single value, then it will be applied to all observations. If it is a vector with the same length as the inputs, then each value given will be used for the corresponding observation.
QuietFlag	Either "Quiet", "Very Quiet", or "Debug". With "Very Quiet", the simulation will run silently. With "Quiet", the simulation will print "Obs=1", "Obs=2", etc... to the console. With "Debug", intermediate information from the CHESS function will print to the console.
ConvergenceCriteria	(numeric) The maximum allowed CompError in for the simulation to be considered complete. CompError = abs(CalcTotMoles - TotMoles) / TotMoles
MaxIter	(integer) The maximum allowed CHESS iterations before the program should give up.

Value

A data frame with chemistry speciation information, including species concentrations, species activities, and total concentrations.

Examples

```
# running the BLM function with a parameter file and input file:
mypfile = system.file("extdata","ParameterFiles","carbonate_system_only.dat4",
                      package = "BLMEngineInR",
                      mustWork = TRUE)
myinputfile = system.file("extdata","InputFiles","carbonate_system_test.blm4",
                          package = "BLMEngineInR",
                          mustWork = TRUE)
BLM(ParamFile = mypfile, InputFile = myinputfile, DoTox = FALSE)

# running the BLM with parameter and input objects
myinputs = GetData(InputFile = myinputfile, ThisProblem = carbonate_system_problem)
```

```
BLM(ThisProblem = carbonate_system_problem, AllInput = myinputs, DoTox = FALSE)

# here we only read in the same files, but the inputs could also be constructed
```

`carbonate_system_problem` *Carbonate system problem*

Description

An example BLMEngineInR problem object, which describes a water-only system with only the (closed) carbonate system.

Usage

```
carbonate_system_problem
```

Format

An object of class `list` of length 24.

Details

This problem consists of two components (hydrogen "H" and carbonate "CO3") and three reactions (dissociation of water/formation of hydroxide "OH", formation of bicarbonate "HCO3" and formation of carbonic acid "H2CO3"). The pH and temperature are supplied as input variables, and the input label "ID" is supplied as well.

Examples

```
carbonate_system_problem$Comp[, c("Name", "Charge", "Type")]
carbonate_system_problem$Spec[, c("Equation", "ActCorr", "LogK", "DeltaH")]
```

`CheckBLMObject` *Check an object for use in the BLMEngineInR package*

Description

This function will compare an object to a reference object to make sure the required list elements are present and that they are the correct types.

Usage

```
CheckBLMObject(Object, Reference, BreakOnError = TRUE)
```

Arguments

`Object`, `Reference`
: R objects that are to be compared. Assumed to be list objects.

`BreakOnError`
: A logical value indicating if an error should stop whatever function or code it might be embedded in ('TRUE', the default) or allow it to proceed without stopping ('FALSE').

Value

The returned value depends on the value of 'BreakOnError':

`TRUE` `TRUE` will be returned invisibly if all checks succeed, and an error with the error list as the text will be triggered if at least one check fails.

`FALSE` The error list will be returned, which will be a zero-length vector if all checks succeed.

Examples

```
# This one works:
myproblem = BlankProblem()
myproblem = AddMassCompartments(
  ThisProblem = myproblem,
  MassName = "Water",
  MassAmt = 1.0,
  MassUnit = "L"
)
CheckBLMObject(Object = myproblem,
               Reference = BlankProblem(),
               BreakOnError = FALSE)

# This one fails:
myproblem$N = NULL
CheckBLMObject(Object = myproblem,
               Reference = BlankProblem(),
               BreakOnError = FALSE)
```

CHESS *CHemical Equilibria in Soils and Solutions*

Description

Given a chemical system, equilibria equations, and total concentrations of components, calculate the species concentrations of each chemical product in the system.

Usage

```
CHESS(
  QuietFlag,
  ConvergenceCriteria,
  MaxIter,
  NMass,
  MassName,
  MassAmt,
  NComp,
  CompName,
  CompType,
  TotConc,
  NSpec,
  SpecName,
  SpecType,
  SpecMCR,
  SpecK,
  SpecTempKelvin,
  SpecDeltaH,
  SpecStoich,
  SpecCharge,
  SpecActCorr,
  DoWHAM,
  AqueousMCR,
  WHAMDonnanMCR,
  HumicSubstGramsPerLiter,
  WHAMMolWt,
  WHAMRadius,
  WHAMP,
  WHAMDLF,
  WHAMKZED,
  SysTempKelvin,
  DoTox,
  MetalName,
  MetalCompR,
  BLCompR,
  NBLMetal,
  BLMetalSpecsR,
  CATarget,
  DodVidCj,
  DodVidCjDonnan,
  DodKidCj,
  DoGammai,
  DoJacDonnan,
  DoJacWHAM,
  DoWHAMSimpleAdjust,
  DoDonnanSimpleAdjust
)
```

Arguments

QuietFlag	character, one of "Very Quiet" (only print out when run is done), "Quiet" (print out Obs=iObs), or "Debug" (print out lots of info)
ConvergenceCriteria	numeric, the maximum value of MaxError that counts as convergence by the Newton-Raphson root-finding algorithm
MaxIter	integer, the maximum number of iterations the Newton-Raphson root-finding algorithm should do before giving up
NMass	integer, number of mass compartments
MassName	CharacterVector (NMass), the names of the mass compartments
MassAmt	NumericVector (NMass), The amount of each mass compartment.
NComp	integer, number of components
CompName	character vector (NComp), the name of each component in the simulation
CompType	character vector (NComp), the type of each component in the simulation
TotConc	numeric vector (NComp), the total concentrations of each component in the simulation (units of e.g., mol/L and mol/kg)
NSpec	integer, number of species reactions
SpecName	character vector (NSpec), the name of the chemical species for which we have formation reactions
SpecType	character vector (NSpec), the type or category of the chemical species for which we have formation reactions.
SpecMCR	IntegerVector (NSpec), the mass compartment of the chemical species for which we have formation reactions
SpecK	numeric vector (NSpec), the equilibrium coefficient of the formation reactions.
SpecTempKelvin	NumericVector (NSpec), the temperature associated with K/logK and DeltaH of the formation reactions
SpecDeltaH	numeric vector (NSpec), the enthalpy change of the formation reactions
SpecStoich	signed integer matrix (NSpec x NComp), the reaction stoichiometry of the formation reactions
SpecCharge	signed integer vector (NSpec), the charge of the chemical species for which we have formation reactions
SpecActCorr	character vector (NSpec), the activity correction method of the chemical species for which we have formation reactions
DoWHAM	boolean, true=there are WHAM species, false=no WHAM species
AqueousMCR	integer, the (1-based) position of the water/aqueous mass compartment. (transformed to 0-based at the beginning of the function)
WHAMDonnanMCR	the mass compartments corresponding to the humic acid (0) and fulvic acid (1) Donnan layers. (transformed to 0-based at the beginning of the function)
HumicSubstGramsPerLiter	NumericVector, length of 2, grams per liter of each organic matter component (HA and FA) in solution

WHAMMolWt	numeric (2), WHAM's molecular weight parameter for organic matter
WHAMRadius	numeric (2), WHAM's molecular radius parameter for organic matter
WHAMP	numeric (2), WHAM's P parameter...
WHAMDLF	numeric (2), WHAM's Double layer overlap factor
WHAMKZED	numeric (2), WHAM's Constant to control DDL at low ZED
SysTempKelvin	double; input temperature for the current observation, in Kelvin
DoTox	logical, TRUE for toxicity mode where the MetalName component concentration is adjusted to try to match the CATarget with BLMetalSpecs
MetalName	character string, the name of the toxic metal
MetalCompR	integer, the position of the metal in the component arrays (i.e., which is the toxic metal component) Note: this is base-1 indexed on input then converted.
BLCompR	integer, the position of the biotic ligand in the component arrays. Note: this is base-1 indexed on input, then converted.
NBLMetal	integer, the number of biotic ligand-bound metal species that are associated with toxic effects.
BLMetalSpecsR	integer vector, the positions of the species in the arrays which contribute to toxicity (i.e., which species are the toxic metal bound to the relevant biotic ligand) Note: these are base-1 indexed on input then converted.
CATarget	numeric, the target critical accumulation in units of mol / kg (only used when DoTox == TRUE)
DodVidCj	boolean, should the Jacobian matrix include the change in the main water solution (excluding Donnan layer volume)?
DodVidCjDonnan	boolean, should the Jacobian matrix include the change in the Donnan layer volume?
DodKidCj	boolean, should the Jacobian matrix include the change in the DOC equilibrium constants?
DoGammai	boolean, should the Jacobian matrix include the change in the activity coefficients?
DoJacDonnan	boolean, should the Jacobian matrix be used to solve the Donnan layer concentrations?
DoJacWHAM	boolean, should the Jacobian matrix be used to solve the WHAM component concentrations?
DoWHAMSimpleAdjust	boolean, should SimpleAdjust be used to solve the WHAM component concentrations?
DoDonnanSimpleAdjust	boolean, should SimpleAdjust be used to solve the Donnan layer concentrations?

Value

list with the following elements:

SpecConc numeric vector (NSpec), the concentrations of each species for which we have formation reactions

FinalIter integer, the number of Newton-Raphson iterations that we needed to reach convergence

FinalMaxError numeric, the highest final absolute error fraction =max(abs(Resid / TotMoles))

CalcTotConc numeric vector (NComp), the calculated total concentrations of each component in the simulation (units of e.g., mol/L and mol/kg)

`CommonParameterDefinitions`

Common Parameter Definitions

Description

These are parameters that are commonly used in the BLMEngineInR package. They will appear throughout the various internal functions, and this central repository of their definitions is helpful.

Arguments

`NMass`	integer, the number of mass compartments.
`MassName`	character vector (NMass), The name of each mass compartment.
`MassAmt`	numeric vector (NMass), The amount of each mass compartment.
`MassUnit`	character vector (NMass), The units for each mass compartment.
`AqueousMCR`	integer, the (1-based) position of the water/aqueous mass compartment.
`BioticLigMCR`	integer, the (1-based) position of the biotic ligand mass compartment(s).
`WHAMDonnanMCR`	integer (2), the mass compartments corresponding to the humic acid (1) and fulvic acid (2) Donnan layers.
`WHAMDonnanMC`	integer (2), the mass compartments corresponding to the humic acid (0) and fulvic acid (1) Donnan layers.
`NInLab`	integer, the number of input label fields
`InLabName`	character vector (NInLab), The names of the input label fields.
`NInVar`	integer, the number of input variables
`InVarName`	character vector (NInVar), The name of each input variable.
`InVarMCR`	integer vector (NInVar), The mass compartment of each input variable. (1-based)
`InVarMC`	integer vector (NInVar), The mass compartment of each input variable. (0-based)
`InVarType`	character vector (NInVar), The type of each input variable. Should be one of "Temperature" (the temperature in degrees C), "pH" (the -log[H]...you know, pH), "WHAM-HA", "WHAM-FA", "WHAM-HAFA" (Windemere Humic Aqueous Model organic matter (input mg C/L), as all humic acid, all fulvic acid, or a mix of humics and fulvics, respectively.), "PercHA" (optionally indicate the percent humic acid in a the WHAM-HAFA component for that compartment.), or "PercAFA" (optionally indicate the percent of active fulvic acid for the WHAM-FA or WHAM-HAFA component for that compartment)

NInComp	integer, the number of input components
InCompName	character vector (NInComp), The names of the input components.
NDefComp	integer, the number of defined components
DefCompName	character vector (NDefComp), the names of each defined component
DefCompFromNum	numeric vector (NDefComp), the number used for deriving the concentration of each defined component
DefCompFromVar	character vector (NDefComp), the variable used for deriving the concentration of each defined component
DefCompCharge	signed integer vector (NDefComp), the charge of each defined component
DefCompMCR	integer vector (NDefComp), the mass compartment number each defined component (1-based)
DefCompMC	integer vector (NDefComp), the mass compartment number each defined component (0-based)
DefCompType	character vector (NDefComp), the type of each defined component
DefCompActCorr	character vector (NDefComp), the activity correction method to use with each defined component
DefCompSiteDens	numeric vector (NDefComp), the site density of each defined component
NComp	integer, the combined number of components in the simulation, including the input components, defined components (and including the defined components that get added by ExpandWHAM)
CompName	character vector (NComp), the name of each component in the simulation
CompCharge	signed integer vector (NComp), the charge of each component in the simulation
CompMCR	integer vector (NComp), the mass compartment of each component in the simulation (1-based)
CompMC	integer vector (NComp), the mass compartment of each component in the simulation (0-based)
CompCtoM	numeric vector (NSpec), the concentration to mass conversion factor of the components
CompType	character vector (NComp), the type of each component in the simulation
CompActCorr	character vector (NComp), the activity correction method of each component in the simulation
CompSiteDens	numeric vector (NComp), the site density of each component in the simulation
CompConc	numeric vector (NComp), the free ion concentrations of each component in the simulation
TotConc	numeric vector (NComp), the total concentrations of each component in the simulation (units of e.g., mol/L and mol/kg)
TotMoles	numeric vector (NComp), the total moles of each component in the simulation (units of mol)
NSpec	integer, the number of chemical species for which we have formation reactions in the simulation

SpecName	character vector (NSpec), the name of the chemical species for which we have formation reactions
SpecMCR	integer vector (NSpec), the mass compartment of the chemical species for which we have formation reactions (1-based)
SpecMC	integer vector (NSpec), the mass compartment of the chemical species for which we have formation reactions (0-based)
SpecActCorr	character vector (NSpec), the activity correction method of the chemical species for which we have formation reactions
SpecNC	integer vector (NSpec), the number of components for the formation reactions
SpecCompList	integer matrix (NSpec x max(SpecNC)), the list of components for the formation reactions
SpecCtoM	numeric vector (NSpec), the concentration to mass conversion factor of the chemical species for which we have formation reactions
SpecCharge	signed integer vector (NSpec), the charge of the chemical species for which we have formation reactions
SpecK	numeric vector (NSpec), the equilibrium coefficient of the formation reactions
SpecLogK	numeric vector (NSpec), the log10-transformed equilibrium coefficient of the formation reactions
SpecDeltaH	numeric vector (NSpec), the enthalpy change of the formation reactions
SpecTempKelvin	numeric vector (NSpec), the temperature associated with K/logK and DeltaH of the formation reactions
SpecStoich	signed integer matrix (NSpec x NComp), the reaction stoichiometry of the formation reactions
SpecConc	numeric vector (NSpec), the concentrations of each species for which we have formation reactions
SpecMoles	numeric vector (NSpec), the moles of each species for which we have formation reactions
NPhase	integer, the number of phases in the phase list
PhaseName	character vector (NPhase), the name of the phases for which we have phase reactions
PhaseNC	integer vector (NPhase), the number of components for the phase reactions
PhaseCompList	integer matrix (NPhase x max(PhaseNC)), the list of components for the phase reactions
PhaseStoich	signed integer matrix (NPhase x NComp), the reaction stoichiometry for the phase reactions
PhaseK	numeric vector (NPhase), the equilibrium coefficient for the phase reactions
PhaseLogK	numeric vector (NPhase), the log10-transformed equilibrium coefficient for the phase reactions
PhaseDeltaH	numeric vector (NPhase), the enthalpy change for the phase reactions
PhaseTemp	numeric vector (NPhase), the temperature associated with K/logK and DeltaH for the phase reactions

PhaseMoles	numeric vector (NPhase), the number of moles of the phases for which we have phase reactions
NSpecialDef	integer, the number of special definitions in the parameter file, including biotic ligands, metals, WHAM versions, etc.
NBL	integer, the number of biotic ligand components associated with toxic effects...typically one...and things might get messed up if it's not one.
NMetal	integer, the number of metal components associated with toxic effects...typically one...and things might get messed up if it's not one.
NBLMetal	integer, the number of biotic ligand-bound metal species that are associated with toxic effects.
BLName	The name of the component that corresponds to the biotic ligand associated with toxic effects.
MetalName	The name of the component that corresponds to the metal associated with toxic effects.
BLMetalName	The names of the species that are the biotic ligand-bound metal associated with toxic effects.
BLCompR	integer vector (NBL), the (1-based) position of the biotic ligand component(s) in the component arrays
BLComp	integer vector (NBL), the (0-based) position of the biotic ligand component(s) in the component arrays
MetalCompR	integer vector (NMetal), the (1-based) position of the metal component(s) in the component arrays (i.e., which is the toxic metal component)
MetalComp	integer vector (NMetal), the (0-based) position of the metal component(s) in the component arrays (i.e., which is the toxic metal component)
BLMetalSpecsR	integer vector (NBLMetal), the (1-based) positions of the species in the arrays which contribute to toxicity (i.e., which species are the toxic metal bound to the relevant biotic ligand)
BLMetalSpecs	integer vector (NBLMetal), the (0-based) positions of the species in the arrays which contribute to toxicity (i.e., which species are the toxic metal bound to the relevant biotic ligand)
DoWHAM	logical, TRUE = there are WHAM species, FALSE = no WHAM species
WHAMDLF	numeric (2), WHAM's Double layer overlap factor
WHAMKZED	numeric (2), WHAM's Constant to control DDL at low ZED
SpecKsel	numeric (NSpec, 2), WHAM's Selectivity coefficient Ksel for diffuse layer binding
WHAMP	numeric (2), WHAM's P parameter...
WHAMRadius	numeric (2), WHAM's molecular radius parameter for organic matter
WHAMMolWt	numeric (2), WHAM's molecular weight parameter for organic matter
HumicSubstGramsPerLiter	numeric (2), grams per liter of each organic matter component (HA and FA) in solution
CATab	data frame, the critical accumulation table from the parameter file.

NCAT	integer, the number of critical accumulations in the parameter file table.
CATarget	numeric, the target critical accumulation in units of mol / kg (only used when DoTox == TRUE)
NObs	integer; the number of chemistry observations
InLabObs	character matrix with NObs rows and InLab columns; the input labels for each observation
InVarObs	matrix with NObs rows and InVar columns; the input variables for each observation
InCompObs	matrix with NObs rows and InComp columns; the input component concentrations for each observation
SysTempCelsiusObs	numeric vector of length NObs; input temperatures, in Celsius
SysTempKelvinObs	numeric vector of length NObs; input temperatures, in Kelvin
SysTempCelsius	double; input temperature for the current observation, in Celsius
SysTempKelvin	double; input temperature for the current observation, in Kelvin
TotConcObs	numeric matrix with NObs rows and NComp columns; the total concentrations of each component, including derived components
pH	numeric vector NObs; input pH for each observation
FinalIter	integer, the number of Newton-Raphson iterations that we needed to reach convergence
FinalMaxError	numeric, the highest final absolute error fraction =max(abs(Resid / TotMoles))
MaxError	numeric, the highest absolute error fraction in this iteration =max(abs(Resid / TotMoles))
CalcTotConc	numeric vector (NComp), the calculated total concentrations of each component in the simulation (units of e.g., mol/L and mol/kg)
QuietFlag	character, one of "Very Quiet" (only print out when run is done), "Quiet" (print out Obs=iObs), or "Debug" (print out lots of info)
DoTox	logical, TRUE for toxicity mode where the MetalName component concentration is adjusted to try to match the CATarget with BLMetalSpecs
ConvergenceCriteria	numeric, the maximum value of MaxError that counts as convergence by the Newton-Raphson root-finding algorithm
MaxIter	integer, the maximum number of iterations the Newton-Raphson root-finding algorithm should do before giving up
IonicStrength	double, the ionic strength of the solution

See Also

Other BLMEngine Functions: GetData(), MatchInputsToProblem(), ReadInputsFromFile()

`Components` *Add or remove components in the problem*

Description

A component should be added either as an input component (with 'AddInComps') or a defined component (with 'AddDefComps'). Both of those functions will call the 'AddComponents' function, but using either 'AddInComps' and 'AddDefComps' ensures that it's very clear where the inputs come from.

Usage

```
AddComponents(
  ThisProblem,
  CompName,
  CompCharge,
  CompMCName = NULL,
  CompType,
  CompActCorr,
  CompSiteDens = 1,
  CompMCR = match(CompMCName, ThisProblem$Mass$Name, nomatch = -1L),
  DoCheck = TRUE
)

RemoveComponents(ThisProblem, ComponentToRemove, DoCheck = TRUE)

AddInComps(
  ThisProblem,
  InCompName,
  InCompCharge,
  InCompMCName = NULL,
  InCompType,
  InCompActCorr,
  InCompMCR = match(InCompMCName, ThisProblem$Mass$Name, nomatch = -1L),
  DoCheck = TRUE
)

RemoveInComps(ThisProblem, InCompToRemove, DoCheck = TRUE)

AddDefComps(
  ThisProblem,
  DefCompName,
  DefCompFromNum = NULL,
  DefCompFromVar = NULL,
  DefCompCharge,
  DefCompMCName = NULL,
  DefCompType,
```

```
  DefCompActCorr,
  DefCompSiteDens,
  DefCompMCR = match(DefCompMCName, ThisProblem$Mass$Name, nomatch = -1L),
  InDefComp = TRUE,
  DoCheck = TRUE
)

RemoveDefComps(ThisProblem, DefCompToRemove, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by 'BlankProblem()'.
CompName, InCompName, DefCompName	A character vector with the name(s) of the components to be added.
CompCharge, InCompCharge, DefCompCharge	An integer vector with the charge(s) of the components to be added.
CompMCName, InCompMCName, DefCompMCName	A character vector with the name(s) of the mass compartments the new components are associated with. Does not need to be specified if 'CompMCR'/'InCompMCR'/'DefCompMCR' is specified instead.
CompType, InCompType, DefCompType	A character vector with the types of the new input variables. Must be one of "MassBal", "FixedAct", "FixedConc", "DonnanHA", or "DonnanFA".
CompActCorr, InCompActCorr, DefCompActCorr	A character vector with the activity correction method(s) of the new components. Must be one of "None", "Debye", "Davies", "DonnanHA", "DonnanFA", "WHAMHA", or "WHAMFA". Generally, "DonnanHA", "DonnanFA", "WHAMHA", and "WHAMFA" will only be used internally.
CompSiteDens, DefCompSiteDens	A numeric vector with the binding site densities of the new components. 'AddInComps' assumes a site density of 1.0.
CompMCR, InCompMCR, DefCompMCR	(optional) A character vector with the indices of the mass compartments the new components are associated with. Only needs to be specified if 'CompMCName'/'InCompMCName'/'DefCompMCName' is not specified.
DoCheck	A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to 'TRUE', as you usually want to make sure something isn't awry, but the value is often set to 'FALSE' when used internally (like in DefineProblem) so the problem is only checked once at the end.
ComponentToRemove, InCompToRemove, DefCompToRemove	A character vector with names or indices of the component(s) to remove from 'ThisProblem'. It is safer to use a name, since the index of the component may be different within 'ThisProblem$Comp$Name' versus 'ThisProblem$InCompName' versus 'ThisProblem$DefComp$Name'.
DefCompFromNum	A numeric vector with the numeric values used to derive the component. Specify 'NA' if the defined component uses a variable to define it.

DefCompFromVar A character vector with the variable names used to derive the component. Specify 'NA' if the defined component uses a number to define it.

InDefComp A logical value indicating if this is a defined component from the parameter file ('TRUE') or was added from another process, such as 'ExpandWHAM' ('FALSE').

Value

'ThisProblem', with the edits done to the component list, including "trickle-down" changes, such as removing formation reactions that used a now-removed component.

See Also

The in-depth example in [BlankProblem()] will show all problem manipulation functions in use.

Other problem manipulation functions: BlankProblem(), CriticalValues, InLabs, InVars, MassCompartments, Phases, SpecialDefs, Species

Examples

```
print(carbonate_system_problem$Comp)
my_new_problem = carbonate_system_problem
my_new_problem = AddInComps(ThisProblem = my_new_problem,
                            InCompName = "Ca",
                            InCompCharge = 2,
                            InCompMCName = "Water",
                            InCompType = "MassBal",
                            InCompActCorr = "Debye")
print(my_new_problem$Comp)
```

ConvertWHAMVThermoFile

Convert from a WHAM V thermodynamic database file

Description

This function will take a thermodynamic database file used by the Windemere Humic Aqueous Model (WHAM) V and the Windows BLM and convert it into a BLMEngineInR chemistry problem list.

Usage

```
ConvertWHAMVThermoFile(ThermoDBSName, RWHAMFile = NULL, RParamFile = NULL)
```

Arguments

ThermoDBSName	Character string with the file path of the WHAM thermodynamic database file to convert (typically ".dbs" file extension).
RWHAMFile	(optional) Character string with the file path of the R-format WHAM parameter file to save (suggest file extension ".wdat").
RParamFile	(optional) Character string with the file path of the R-format BLM parameter file to save (suggest file extension ".dat4").

Value

The BLMEngineInR-compatible chemistry problem object. If RWHAMFile or RParamFile are provided, this will return invisibly.

ConvertWindowsParamFile

Convert From a Windows BLM Parameter File

Description

Convert From a Windows BLM Parameter File

Usage

```
ConvertWindowsParamFile(
  WindowsParamFile,
  RParamFile = NULL,
  RWHAMFile = NULL,
  MarineFile = FALSE
)
```

Arguments

WindowsParamFile	Character string with the file path of the Windows-format BLM parameter file. Typically will have the extension ".dat".
RParamFile	(optional) Character string with the file path of the R-format BLM parameter file to save.
RWHAMFile	(optional) Character string with the file path of the R-format WHAM parameter file to save.
MarineFile	Boolean value - is this a marine file? If so, it uses a lower mass value. In the Windows BLM, this is equivalent to using the "/M" switch. Defaults to 'FALSE'.

Value

The BLMEngineInR-compatible chemistry problem object. If RParamFile is provided, this will return invisibly.

`CriticalValues` *Edit Critical Values Table*

Description

'AddCriticalValues' will add one or more rows to the critical accumulation table (CAT or CATab), while 'RemoveCriticalValues' will remove one or more rows,

Usage

```
AddCriticalValues(
  ThisProblem,
  CATab = data.frame(),
  CA = CATab[, which(colnames(CATab) %in% c("CA", "CA (nmol/gw)"))[1]],
  Species = CATab$Species,
  TestType = CATab[, which(colnames(CATab) %in% c("TestType", "Test.Type",
    "Test Type"))[1]],
  Duration = CATab$Duration,
  Lifestage = CATab$Lifestage,
  Endpoint = CATab$Endpoint,
  Quantifier = CATab$Quantifier,
  References = CATab$References,
  Miscellaneous = CATab$Miscellaneous,
  DoCheck = TRUE
)

RemoveCriticalValues(ThisProblem, CAToRemove, DoCheck = TRUE)
```

Arguments

`ThisProblem`	A list object with a structure like that returned by 'BlankProblem()'.
`CATab`	a data.frame object with, at a minimum, columns named 'CA'/'CA (nmol/gw)', 'Species', 'TestType'/'Test.Type'/'Test Type', 'Duration', 'Lifestage', 'Endpoint', 'Quantifier', 'References', 'Miscellaneous'. See optional parameter descriptions for further descriptions of each of those columns.
`CA`	(optional) a numeric vector of the critical accumulation value(s) in nmol/gw.
`Species`	(optional) a character vector of the species names to include for the corresponding 'CA' value.
`TestType`	(optional) a character vector of the test type (e.g., "Acute" or "Chronic") to include for the corresponding 'CA' value.
`Duration`	(optional) a character vector of the Duration to include for the corresponding 'CA' value. Can also be '"DIV=#.##"' for FAV, FCV, WQS, and HC5 critical values.
`Lifestage`	(optional) a character vector of the organism Lifestage to include for the corresponding 'CA' value. Can also be '"ACR=#.##"' for FAV, WQS, and HC5 critical values.

Endpoint	(optional) a character vector of the Endpoint to include for the corresponding ‘CA‘ value. This can also be either ‘"FAV"‘, ‘"FCV"‘, ‘"HC5"‘, ‘"WQS"‘, ‘"CMC"‘, or ‘"CCC"‘ to indicate this critical value calculates one of those water quality standards.
Quantifier	(optional) a character vector of the Quantifier (e.g., EC50, NOEC, ...) to include for the corresponding ‘CA‘ value. May also be ‘NA‘ if this is a WQS value.
References	(optional) a character vector of the list of References to include for the corresponding ‘CA‘ value. Each ‘CA‘ value requires a single character string with no line breaks.
Miscellaneous	(optional) a character vector of the miscellaneous information (e.g., how the value was calculated, test conditions not covered by other columns, etc.) to include for the corresponding ‘CA‘ value.
DoCheck	A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to ‘TRUE‘, as you usually want to make sure something isn't awry, but the value is often set to ‘FALSE‘ when used internally (like in DefineProblem) so the problem is only checked once at the end.
CAToRemove	an integer vector - the indices/row numbers of the critical values to remove from the table.

Value

The edited version of ‘ThisProblem‘.

See Also

Other problem manipulation functions: BlankProblem(), Components, InLabs, InVars, MassCompartments, Phases, SpecialDefs, Species

Examples

```
my_new_problem = carbonate_system_problem

my_new_problem = AddCriticalValues(
  ThisProblem = my_new_problem,
  CA = 12345,
  Species = "A. species",
  TestType = "Acute",
  Duration = "24h",
  Lifestage = "adult",
  Endpoint = "survival",
  Quantifier = "LC50",
  References = "thin air",
  Miscellaneous = "individual data point"
)

lots_of_data = data.frame(CA = runif(26),
                          Species = paste0(LETTERS,". species"),
                          TestType = "Acute",
```

```
                          Duration = "24h",
                          Lifestage = "adult",
                          Endpoint = "survival",
                          Quantifier = "LC50",
                          References = "thin air")
my_new_problem = AddCriticalValues(
  ThisProblem = my_new_problem,
  CATab = lots_of_data
)

my_new_problem = RemoveCriticalValues(
  ThisProblem = my_new_problem,
  CAToRemove = which((my_new_problem$CATab$Species == "A. species") &
                       is.na(my_new_problem$CATab$Miscellaneous))
)

print(my_new_problem$CATab)
```

`Cu_full_inorganic_problem`

Cu problem with only inorganic components

Description

An example BLMEngineInR problem object, which describes a system with all of the common cations (Ca, Mg, Na, K) and anions (SO4, Cl, CO3) represented with their usual reactions. Copper is also represented as the toxic metal binding to a biotic ligand, and some example critical accumulations values are provided including one for the United States Environmental Protection Agency's (USEPA) final acute value (FAV). These critical accumulation values are the ones calibrated from the full organic model, as the DOC complexation should not affect the amount of organic matter required to induce a toxic effect, in theory. This will not give accurate predictions of toxicity when DOC is present in the water.

Usage

```
Cu_full_inorganic_problem
```

Format

An object of class `list` of length 24.

Cu_full_organic_problem *Copper problem with WHAM V organic matter*

Description

An example BLMEngineInR problem object, which describes a system with organic matter represented by WHAM V, and all of the common cations (Ca, Mg, Na, K) and anions (SO4, Cl, CO3) represented with their usual reactions. Copper is also represented as the toxic metal binding to a biotic ligand, and some example critical accumulations values are provided, including one for the United States Environmental Protection Agency's (USEPA) final acute value (FAV).

Usage

```
Cu_full_organic_problem
```

Format

An object of class `list` of length 24.

DefineProblem *Define the speciation problem*

Description

'DefineProblem' reads in a parameter file, and sets up the required vectors and matrices that will be needed to run the speciation calculations in CHESS.

Usage

```
DefineProblem(ParamFile, WriteLog = FALSE)
```

Arguments

`ParamFile`	the path and file name to a parameter file
`WriteLog`	if TRUE, the CHESS.LOG file will be written, summarizing the current problem

Value

Returns a 'list' object with each list item named according to the template of BlankProblem

Examples

```
mypfile = system.file("extdata", "ParameterFiles",
                      "carbonate_system_only.dat4",
                      package = "BLMEngineInR", mustWork = TRUE)
thisProblem = DefineProblem(mypfile)
```

DefineWHAM *Read a WHAM file and make a WHAM list*

Description

A WHAM file is a text file (typically with the file extension ".wdat") that has all of the information necessary for defining organic matter binding, according to the Windemere Humic Aqueous Model (WHAM). Only constants relating directly to organic matter binding are in this object and file (i.e., nothing related to inorganic binding). This is the information needed by the ‘ExpandWHAM()‘ function and to do organic matter binding in the ‘CHESS‘ subroutine.

Usage

```
DefineWHAM(WHAMVer = "V", WHAMFile = NA)
```

Arguments

WHAMVer	a character string specifying the WHAM version to use, must be one of ‘"V"‘ (default), ‘"VI"‘, or ‘"VII"‘. Ignored if ‘WHAMFile‘ is not ‘NA‘.
WHAMFile	(optional) a character string specifying the file path of a WHAM parameter file

Value

A WHAM list in the format of ‘BlankWHAM()‘.

GetData *Get data from the input file*

Description

‘GetData‘ reads in the input file and prepares it for input to the BLM function.

Usage

```
GetData(
  InputFile,
  ThisProblem = NULL,
  NInLab = ThisProblem$N["InLab"],
  InLabName = ThisProblem$InLabName,
  NInVar = ThisProblem$N["InVar"],
  InVarName = ThisProblem$InVar$Name,
  InVarMCR = ThisProblem$InVar$MCR,
  InVarType = ThisProblem$InVar$Type,
  NInComp = ThisProblem$N["InComp"],
  InCompName = ThisProblem$InCompName,
```

```
  NComp = ThisProblem$N["Comp"],
  CompName = ThisProblem$Comp$Name,
  NDefComp = ThisProblem$N["DefComp"],
  DefCompName = ThisProblem$DefComp$Name,
  DefCompFromNum = ThisProblem$DefComp$FromNum,
  DefCompFromVar = ThisProblem$DefComp$FromVar,
  DefCompSiteDens = ThisProblem$DefComp$SiteDens
)
```

Arguments

`InputFile`	character(1); the path and file name to a BLM input file
`ThisProblem`	a list object following the template of BlankProblem
`NInLab`	integer; number of input label columns
`InLabName`	character vector of length 'NInLab'; names of input columns
`NInVar`	integer; Number of input variables
`InVarName`	character vector of length 'NInVar'; Names of input variables
`InVarMCR`	integer vector of length 'NInVar'; Mass compartments of input variables
`InVarType`	character vector of length 'NInVar'; Types of input variables
`NInComp`	integer; Number in input components
`InCompName`	character vector of length 'NInComp'; names of input components
`NComp`	integer; Number of components
`CompName`	character vector of length 'NComp'; component names
`NDefComp`	integer; Number of defined components
`DefCompName`	character vector of length 'NDefComp'; defined component names
`DefCompFromNum`	numeric vector of length 'NDefComp'; the number the defined component is formed from
`DefCompFromVar`	character vector of length 'NDefComp'; the column used to form the defined component
`DefCompSiteDens`	numeric vector of length 'NDefComp'; the binding site density of each defined component

Value

Returns a 'list' object with the following components:

NObs integer; the number of chemistry observations

InLabObs matrix with NObs rows and InLab columns; the input labels for each observation

InVarObs matrix with NObs rows and InVar columns; the input variables for each observation

InCompObs matrix with NObs rows and InComp columns; the input component concentrations for each observation

`SysTempCelsiusObs` numeric vector of length `NObs`; input temperatures, in Celsius

SysTempKelvinObs numeric vector of length NObs; input temperatures, in Kelvin

pHObs numeric vector (NObs); input pH for each observation

TotConcObs numeric matrix with NObs rows and NComp columns; the total concentrations of each component, including derived components

HumicSubstGramsPerLiterObs numeric matrix with NObs rows and 2 columns; the total concentration of humic substances (humic/HA and fulvic/FA) in grams per liter

See Also

Other BLMEngine Functions: CommonParameterDefinitions, MatchInputsToProblem(), ReadInputsFromFile()

Examples

```
myinputfile = system.file("extdata", "InputFiles",
                          "carbonate_system_test.blm4",
                          package = "BLMEngineInR", mustWork = TRUE)
GetData(InputFile = myinputfile, ThisProblem = carbonate_system_problem)#'
```

InLabs *Add or remove input labels in a problem*

Description

Add or remove input labels in a problem

Usage

```
AddInLabs(ThisProblem, InLabName, DoCheck = TRUE)

RemoveInLabs(ThisProblem, InLabToRemove, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by 'BlankProblem()'.
InLabName	A character vector with the name(s) of the new input label(s).
DoCheck	A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to 'TRUE', as you usually want to make sure something isn't awry, but the value is often set to 'FALSE' when used internally (like in DefineProblem) so the problem is only checked once at the end.
InLabToRemove	A character vector with names or indices of the input label(s) to remove from 'ThisProblem'.

Value

'ThisProblem', with the edited input labels.

See Also

Other problem manipulation functions: BlankProblem(), Components, CriticalValues, InVars, MassCompartments, Phases, SpecialDefs, Species

Examples

```
my_new_problem = carbonate_system_problem
print(carbonate_system_problem$InLabName) # ID only

my_new_problem = AddInLabs(ThisProblem = my_new_problem, InLabName = "ID2")
my_new_problem = RemoveInLabs(my_new_problem, InLabToRemove = "ID")

print(my_new_problem$InLabName) # ID2 only
```

InVars *Add or remove a input variables in a problem*

Description

Add or remove a input variables in a problem

Usage

```
AddInVars(
  ThisProblem,
  InVarName,
  InVarMCName = NULL,
  InVarType = c("Temperature", "pH", "WHAM-FA", "WHAM-HA", "WHAM-HAFA", "PercHA",
    "PercAFA", "Misc"),
  InVarMCR = match(InVarMCName, ThisProblem$Mass$Name, nomatch = -1L),
  DoCheck = TRUE
)

RemoveInVars(ThisProblem, InVarToRemove, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by 'BlankProblem()'.
InVarName	A character vector with the name(s) of the input input variable(s).
InVarMCName	A character vector with the name(s) of the mass compartments the new input variables are associated with. Does not need to be specified if 'InVarMCR' is specified instead.
InVarType	A character vector with the types of the new input variables. Must be one of "Temp", "pH", "WHAM-HA", "WHAM-FA", "WHAM-HAFA", "PercHA", "PercAFA", and "Misc".

InVarMCR	(optional) A character vector with the indices of the mass compartments the new input variables are associated with. Only needs to be specified if 'InVarMCName' is not specified.
DoCheck	A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to 'TRUE', as you usually want to make sure something isn't awry, but the value is often set to 'FALSE' when used internally (like in DefineProblem) so the problem is only checked once at the end.
InVarToRemove	A character vector with names or indices of the input variable(s) to remove from 'ThisProblem'.

Value

'ThisProblem', with the changed input variables. If the input variable being added is pH, "H" and "OH" components will also be added as fixed activity components.

See Also

Other problem manipulation functions: BlankProblem(), Components, CriticalValues, InLabs, MassCompartments, Phases, SpecialDefs, Species

Examples

```
print(carbonate_system_problem$InVar)
my_new_problem = carbonate_system_problem
my_new_problem = AddInVars(ThisProblem = my_new_problem,
                           InVarName = c("Humics", "Fulvics"),
                           InVarMCName = "Water",
                           InVarType = c("WHAM-HA","WHAM-FA"))
my_new_problem = RemoveInVars(ThisProblem = my_new_problem,
                              InVarToRemove = "Humics")
print(my_new_problem$InVar)
```

ListCAT *List Critical Accumulation Table*

Description

List out the critical accumulation table for the user to allow them to pick which CAT number they should specify for a toxicity run where the critical value is coming from the table in the parameter file.

Usage

```
ListCAT(ParamFile)
```

Arguments

`ParamFile` character string; the file name and path of the parameter file.

Value

A `data.frame` object with the CAT table in the given parameter file. Columns include:

`Num` the number or index in the table

`'CA (nmol/gw)'` the critical accumulation in units of nmol/gw

`Species` species name or CA significance, such as HC5 or FAV

`'Test Type'` acute or chronic

`Duration` test duration (e.g., 48 h)

`Lifestage` age or size of the organisms

`Endpoint` toxicity endpoint (e.g., mortality, reproduction)

`Quantifier` endpoint quantifier or effect level (e.g., LC50, EC10, NOEC)

`References` citations of sources with the toxicity data that went into calculating the CA, or the citation of the HC5 or FAV

`Miscellanous` other notes or comments (e.g., number of data points or methods of calculating)

Examples

```
mypfile = system.file("extdata", "ParameterFiles", "Cu_full_organic.dat4",
                      package = "BLMEngineInR",
                      mustWork = TRUE)
ListCAT(ParamFile = mypfile)
```

MassCompartments *Add or remove mass compartments in a problem*

Description

Add or remove mass compartments in a problem

Usage

```
AddMassCompartments(
  ThisProblem,
  MassTable = data.frame(),
  MassName = MassTable$Name,
  MassAmt = MassTable$Amt,
  MassUnit = MassTable$Unit,
  InMass = TRUE,
  DoCheck = TRUE
)

RemoveMassCompartments(ThisProblem, MCToRemove, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by 'BlankProblem()'.
MassTable	A 'data.frame' object with, at a minimum, columns 'Name', 'Amt', and 'Unit', defining the characteristics of the mass compartment(s) to add.
MassName	A character vector with the name(s) of the new mass compartment(s).
MassAmt	A numeric vector with the mass compartment amount(s).
MassUnit	A character vector with the units for the amount(s) of the mass compartment(s).
InMass	A logical value or vector indicating if this mass compartment is in the parameter file ('TRUE', default) or was created as a result of, e.g. the 'ExpandWHAM' function ('FALSE').
DoCheck	A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to 'TRUE', as you usually want to make sure something isn't awry, but the value is often set to 'FALSE' when used internally (like in DefineProblem) so the problem is only checked once at the end.
MCToRemove	A character vector with names or indices of the mass compartment(s) to remove from 'ThisProblem'.

Value

'ThisProblem', with all the edited mass compartments, along with any components, input variables, etc. associated with those mass compartments edited.

See Also

Other problem manipulation functions: BlankProblem(), Components, CriticalValues, InLabs, InVars, Phases, SpecialDefs, Species

Examples

```
print(carbonate_system_problem$Mass)
my_new_problem = carbonate_system_problem
my_new_problem = AddMassCompartments(ThisProblem = my_new_problem,
                                     MassName = c("Soil", "BL"),
                                     MassAmt = 1,
                                     MassUnit = c("kg","kg wet"))
print(my_new_problem$Mass)
my_new_problem = RemoveMassCompartments(ThisProblem = my_new_problem,
                                        MCToRemove = "Soil")
print(my_new_problem$Mass)
```

MatchInputsToProblem *Match Inputs to Problem*

Description

'MatchInputsToProblem' will take the input variables and component concentrations and match/transform them to the inputs for full list of components, including defined components and WHAM components.

Usage

```
MatchInputsToProblem(
  DFInputs = data.frame(),
  NObs = nrow(DFInputs),
  InLabObs = DFInputs[, ThisProblem$InLabName, drop = FALSE],
  InVarObs = DFInputs[, ThisProblem$InVar$Name, drop = FALSE],
  InCompObs = DFInputs[, ThisProblem$InCompName, drop = FALSE],
  ThisProblem = NULL,
  NInVar = ThisProblem$N["InVar"],
  InVarName = ThisProblem$InVar$Name,
  InVarMCR = ThisProblem$InVar$MCR,
  InVarType = ThisProblem$InVar$Type,
  NInComp = ThisProblem$N["InComp"],
  InCompName = ThisProblem$InCompName,
  NComp = ThisProblem$N["Comp"],
  CompName = ThisProblem$Comp$Name,
  NDefComp = ThisProblem$N["DefComp"],
  DefCompName = ThisProblem$DefComp$Name,
  DefCompFromNum = ThisProblem$DefComp$FromNum,
  DefCompFromVar = ThisProblem$DefComp$FromVar,
  DefCompSiteDens = ThisProblem$DefComp$SiteDens
)
```

Arguments

DFInputs	A data.frame object with, at a minimum, columns named for 'ThisProblem$InLabName', 'ThisProblem$InVar$Name' and 'ThisProblem$InCompName'.
NObs	integer; the number of chemistry observations
InLabObs	character matrix with NObs rows and InLab columns; the input labels for each observation
InVarObs	matrix with 'NObs' rows and 'NInVar' columns; the input variables for each observation
InCompObs	matrix with 'NObs' rows and 'NInComp' columns; the input component concentrations for each observation
ThisProblem	a list object following the template of BlankProblem

NInVar	integer; Number of input variables
InVarName	character vector of length 'NInVar'; Names of input variables
InVarMCR	integer vector of length 'NInVar'; Mass compartments of input variables
InVarType	character vector of length 'NInVar'; Types of input variables
NInComp	integer; Number in input components
InCompName	character vector of length 'NInComp'; names of input components
NComp	integer; Number of components
CompName	character vector of length 'NComp'; component names
NDefComp	integer; Number of defined components
DefCompName	character vector of length 'NDefComp'; defined component names
DefCompFromNum	numeric vector of length 'NDefComp'; the number the defined component is formed from
DefCompFromVar	character vector of length 'NDefComp'; the column used to form the defined component
DefCompSiteDens	numeric vector of length 'NDefComp'; the binding site density of each defined component

Value

Returns a list object with the following components:

NObs integer; the number of chemistry observations

InLabObs matrix with NObs rows and InLab columns; the input labels for each observation

InVarObs matrix with NObs rows and InVar columns; the input variables for each observation

InCompObs matrix with NObs rows and InComp columns; the input component concentrations for each observation

SysTempCelsiusObs numeric vector of length NObs; input temperatures, in Celsius

SysTempKelvinObs numeric vector of length NObs; input temperatures, in Kelvin

pHObs numeric vector (NObs); input pH for each observation

TotConcObs numeric matrix with NObs rows and NComp columns; the total concentrations of each component, including derived components

HumicSubstGramsPerLiterObs numeric matrix with NObs rows and 2 columns; the total concentration of humic substances (humic/HA and fulvic/FA) in grams per liter

See Also

Other BLMEngine Functions: CommonParameterDefinitions, GetData(), ReadInputsFromFile()

MW *Molecular and atomic weights*

Description

'MW' is a named list of molecular and atomic weights. The name of the list element is the symbol or formula of the chemical element or molecule (e.g. find hydrogen with "H", carbon dioxide as "CO2").

Usage

```
MW
```

Format

An object of class `numeric` of length 132.

Source

Prohaska, T., Irrgeher, J., Benefield, J., Böhlke, J., Chesson, L., Coplen, T., Ding, T., Dunn, P., Gröning, M., Holden, N., Meijer, H., Moossen, H., Possolo, A., Takahashi, Y., Vogl, J., Walczyk, T., Wang, J., Wieser, M., Yoneda, S., Zhu, X. & Meija, J. (2022). Standard atomic weights of the elements 2021 (IUPAC Technical Report). Pure and Applied Chemistry, 94(5), 573-600. https://doi.org/10.1515/pac-2019-0603

Examples

```
# check that the molecular weight of CaCO3 is the same as Ca + C + O * 3
sum(MW[c("Ca", "C")], MW["O"] * 3) #=100.086
MW["CaCO3"] #=100.086
```

Ni_full_organic_problem *Ni problem with WHAM V organic matter*

Description

An example BLMEngineInR problem object, which describes a system with all of the common cations (Ca, Mg, Na, K) and anions (SO4, Cl, CO3) represented with their usual reactions. Nickel is also represented as the toxic metal binding to the biotic ligand, and the critical accumulations from the Santore et al. (2021) paper.

Usage

```
Ni_full_organic_problem
```

Format

An object of class `list` of length 25.

References

Santore, Robert C., Kelly Croteau, Adam C. Ryan, Christian Schlekat, Elizabeth Middleton, Emily Garman, and Tham Hoang (2021). A Review of Water Quality Factors that Affect Nickel Bioavailability to Aquatic Organisms: Refinement of the Biotic Ligand Model for Nickel in Acute and Chronic Exposures. Environmental Toxicology and Chemistry, col 40, iss. 8, pp 2121-2134. doi: 10.1002/etc.5109

`Ni_HCO3_full_organic_problem`

Ni problem with WHAM V organic matter and NiHCO3 toxic

Description

An example BLMEngineInR problem object, which describes a system with all of the common cations (Ca, Mg, Na, K) and anions (SO4, Cl, CO3) represented with their usual reactions. Nickel is also represented as the toxic metal binding to the biotic ligand, and the critical accumulations from the Santore et al. (2021) paper. This version has a BL-NiHCO3 species whose binding constant has been calibrated to Ceriodaphnia dubia toxicity. Ceriodaphnia dubia are sensitive to bicarbonate toxicity and this file simulates this mixtures effect.

Usage

```
Ni_HCO3_full_organic_problem
```

Format

An object of class `list` of length 25.

References

Santore, Robert C., Kelly Croteau, Adam C. Ryan, Christian Schlekat, Elizabeth Middleton, Emily Garman, and Tham Hoang (2021). A Review of Water Quality Factors that Affect Nickel Bioavailability to Aquatic Organisms: Refinement of the Biotic Ligand Model for Nickel in Acute and Chronic Exposures. Environmental Toxicology and Chemistry, col 40, iss. 8, pp 2121-2134. doi: 10.1002/etc.5109

Phases *Add or remove phase reactions in a problem*

Description

PHASES ARE NOT CURRENTLY IMPLEMENTED. This function is here for as a placeholder since it will require much of the same support infrastructure once it is implemented, but no reactions are processed in CHESS.

Usage

```
AddPhases(
  ThisProblem,
  PhaseEquation = character(),
  PhaseName = character(),
  PhaseCompNames = list(),
  PhaseCompStoichs = list(),
  PhaseStoich = NULL,
  PhaseLogK,
  PhaseDeltaH,
  PhaseTempKelvin,
  PhaseMoles,
  DoCheck = TRUE
)

RemovePhases(ThisProblem, PhasesToRemove, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by 'BlankProblem()'.
PhaseEquation	A character vector giving the chemical equation for a formation reaction. This must include the stoichiometric coefficients for each reactant, even if it's 1. (e.g., the equation for the formation of calcium chloride would be '"CaCl2 = 1 * Ca + 2 * Cl"'). If 'PhaseName' is also supplied, then a partial equation with just the right hand side (reactants) can be supplied (i.e., '"= 1 * Ca + 2 * Cl"'). Can be omitted if either 'PhaseStoich' or both 'PhaseCompNames' and 'PhaseCompStoichs' are supplied.
PhaseName	A character vector with the name(s) of the species to add formation reactions for. Can be omitted if 'SpecEquation' indicates the phase name.
PhaseCompNames	A list where each element is a character vector of the component names used to form each phase. See examples for clarification. Can be omitted if 'PhaseEquation' or 'PhaseStoich' is supplied.
PhaseCompStoichs	A list where each element is an integer vector of the stoichiometric coefficients of each component used to form each phase. See examples for clarification. Can be omitted if 'PhaseEquation' or 'PhaseStoich' is supplied.

`PhaseStoich` A matrix of stoichiometric coefficients, where each row corresponds to a phase reaction and each column corresponds to a component. The columns should match 'ThisProblem$Comp$Name' exactly. Can be omitted if either 'PhaseEquation' or both 'PhaseCompNames' and 'PhaseCompStoichs' are supplied.

`PhaseLogK` A numeric vector with the log10-transformed equilibrium coefficients of the phase formation reactions.

`PhaseDeltaH` A numeric vector with the change in enthalpy of the phase formation reactions.

`PhaseTempKelvin` A numeric vector with the temperatures (in Kelvin) corresponding to 'PhaseDeltaH' values of the phase formation reactions.

`PhaseMoles` A numeric vector with the moles of the phase.

`DoCheck` A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to 'TRUE', as you usually want to make sure something isn't awry, but the value is often set to 'FALSE' when used internally (like in DefineProblem) so the problem is only checked once at the end.

`PhasesToRemove` A character or integer vector indicating the names or indices (respectively) of the phase formation reactions to remove.

Value

'ThisProblem', with the phase reaction(s) changed.

See Also

Other problem manipulation functions: `BlankProblem()`, `Components`, `CriticalValues`, `InLabs`, `InVars`, `MassCompartments`, `SpecialDefs`, `Species`

Examples

```
print(carbonate_system_problem$Phase)
my_new_problem = carbonate_system_problem
my_new_problem = AddPhases(ThisProblem = my_new_problem,
                           PhaseEquation = "CO2(g) = 1 * CO3 + 2 * H",
                           PhaseLogK = -1.5,
                           PhaseDeltaH = 0,
                           PhaseTempKelvin = 0,
                           PhaseMoles = 10^-3.5)
print(my_new_problem$Phase)
```

`ReadInputsFromFile` *Read a BLM Input File*

Description

'ReadInputsFromFile' will read a BLM input file, assuming it matches the problem as defined by the input arguments.

Usage

```
ReadInputsFromFile(
  InputFile,
  ThisProblem = NULL,
  NInLab = ThisProblem$N["InLab"],
  InLabName = ThisProblem$InLabName,
  NInVar = ThisProblem$N["InVar"],
  InVarName = ThisProblem$InVar$Name,
  NInComp = ThisProblem$N["InComp"],
  InCompName = ThisProblem$InCompName
)
```

Arguments

`InputFile`	character; the path and file name to a BLM input file
`ThisProblem`	a list object following the template of BlankProblem
`NInLab`	integer; number of input label columns
`InLabName`	character vector of length 'NInLab'; names of input columns
`NInVar`	integer; Number of input variables
`InVarName`	character vector of length 'NInVar'; Names of input variables
`NInComp`	integer; Number in input components
`InCompName`	character vector of length 'NInComp'; names of input components

Value

Returns a `list` object with the following components:

`NObs` integer; the number of chemistry observations

`InLabObs` matrix with `Obs` rows and `InLab` columns; the input labels for each observation

`InVarObs` matrix with `Obs` rows and `InVar` columns; the input variables for each observation

`InCompObs` matrix with `Obs` rows and `InComp` columns; the input component concentrations for each observation

See Also

Other BLMEngine Functions: `CommonParameterDefinitions`, `GetData()`, `MatchInputsToProblem()`

Examples

```
myinputfile = system.file("extdata", "InputFiles",
                          "carbonate_system_test.blm4",
                          package = "BLMEngineInR", mustWork = TRUE)
ReadInputsFromFile(InputFile = myinputfile,
                   ThisProblem = carbonate_system_problem)
```

SpecialDefs *Add or remove species definitions*

Description

The special definitions in a parameter file include indicating the biotic ligand species relevant to toxicity ("BL"), the toxic metal ("Metal"), the species responsible for the critical accumulation associated with toxicity at the biotic ligand ("BL-Metal"), and the model version of the Windemere Humic Aqueous Model to use to represent organic matter binding ("WHAM").

Usage

```
AddSpecialDefs(ThisProblem, Value, SpecialDef, DoCheck = TRUE)

RemoveSpecialDefs(ThisProblem, SpecialDefToRemove, Index = 1, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by ‘BlankProblem()‘.
Value	A character vector. When ‘SpecialDef‘ is either ‘"BL"‘ or ‘"Metal"‘, this should be the name of a component in ‘ThisProblem$Chem$Name‘. When ‘SpecialDef‘ is ‘"BL-Metal"‘, this should be the name of a chemical species in ‘ThisProblem$Spec$Name‘. When ‘SpecialDef‘ is ‘"WHAM"‘, this should be either a supported WHAM version number (i.e., one of ‘"V"‘, ‘"VI"‘, or ‘"VII"‘), or the file path to a WHAM parameters file (.wdat file) that follows the format of one of the standard versions supplied with this package (see ‘system.file("extdata/WHAM/WHAM_V.wdat", package = "BLMEngineInR")‘ for an example).
SpecialDef	A character vector indicating which special definition to add a value for. Valid values are ‘"BL"‘, ‘"Metal"‘, ‘"BL-Metal"‘, ‘"BLMetal"‘ (same as ‘"BL-Metal"‘), and ‘"WHAM"‘.
DoCheck	A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to ‘TRUE‘, as you usually want to make sure something isn’t awry, but the value is often set to ‘FALSE‘ when used internally (like in DefineProblem) so the problem is only checked once at the end.
SpecialDefToRemove	The name of the special definition to remove.
Index	If applicable (such as if there are two BL-Metal species), the index of which to remove (i.e., the first one or second one).

Value

'ThisProblem', with the special definitions changed.

See Also

Other problem manipulation functions: BlankProblem(), Components, CriticalValues, InLabs, InVars, MassCompartments, Phases, Species

Examples

```
print(carbonate_system_problem[c("BL","Metal","BLMetal","WHAM")])
my_new_problem = carbonate_system_problem
my_new_problem = AddInComps(ThisProblem = my_new_problem, InCompName = "Cu",
                            InCompCharge = 2,
                            InCompMCName = "Water",
                            InCompType = "MassBal",
                            InCompActCorr = "Debye")
my_new_problem = AddSpecialDefs(ThisProblem = my_new_problem,
                                Value = "Cu",
                                SpecialDef = "Metal")
print(my_new_problem[c("BL","Metal","BLMetal","WHAM")])
my_new_problem = RemoveSpecialDefs(ThisProblem = my_new_problem,
                                   SpecialDefToRemove = "Metal")
print(my_new_problem[c("BL","Metal","BLMetal","WHAM")])
```

Species *Add or remove a species reactions in a problem*

Description

Functions to add or remove species formation reactions.

Usage

```
AddSpecies(
  ThisProblem,
  SpecEquation = character(),
  SpecName = character(),
  SpecMCName = NULL,
  SpecType = "Normal",
  SpecActCorr,
  SpecCompNames = list(),
  SpecCompStoichs = list(),
  SpecStoich = NULL,
  SpecLogK,
  SpecDeltaH,
  SpecTempKelvin,
  SpecMCR = match(SpecMCName, ThisProblem$Mass$Name, nomatch = -1L),
```

```
  InSpec = TRUE,
  DoCheck = TRUE
)

RemoveSpecies(ThisProblem, SpeciesToRemove, DoCheck = TRUE)
```

Arguments

ThisProblem	A list object with a structure like that returned by ‘BlankProblem()‘.
SpecEquation	A character vector giving the chemical equation for a formation reaction. This must include the stoichiometric coefficients for each reactant, even if it’s 1. (e.g., the equation for the formation of calcium chloride would be ‘"CaCl2 = 1 * Ca + 2 * Cl"‘). If ‘SpecName‘ is also supplied, then a partial equation with just the right hand side (reactants) can be supplied (i.e., ‘"= 1 * Ca + 2 * Cl"‘). Can be omitted if either ‘SpecStoich‘ or both ‘SpecCompNames‘ and ‘SpecCompStoichs‘ are supplied.
SpecName	A character vector with the name(s) of the species to add formation reactions for. Can be omitted if ‘SpecEquation‘ indicates the species name.
SpecMCName	A character vector with the name(s) of the mass compartments the new species are associated with. Does not need to be specified if ‘SpecMCR‘ is specified instead.
SpecType	A character vector with the species type. SpecType values must be either ‘"Normal"‘, ‘"DonnanHA"‘, ‘"DonnanFA"‘, ‘"WHAMHA"‘, ‘"WHAMFA"‘. The default value is ‘"Normal"‘, while the others are usually only needed for indicating species that are added from ‘ExpandWHAM‘.
SpecActCorr	A character vector with the activity correction method(s) of the new species. Must be one of "None", "Debye", "Davies", "DonnanHA", "DonnanFA", "WHAMHA", or "WHAMFA".
SpecCompNames	A list where each element is a character vector of the component names used to form each species. See examples for clarification. Can be omitted if ‘SpecEquation‘ or ‘SpecStoich‘ is supplied.
SpecCompStoichs	A list where each element is an integer vector of the stoichiometric coefficients of each component used to form each species. See examples for clarification. Can be omitted if ‘SpecEquation‘ or ‘SpecStoich‘ is supplied.
SpecStoich	A matrix of stoichiometric coefficients, where each row corresponds to a chemical species and each column corresponds to a component. The columns should match ‘ThisProblem$Comp$Name‘ exactly. Can be omitted if either ‘SpecEquation‘ or both ‘SpecCompNames‘ and ‘SpecCompStoichs‘ are supplied.
SpecLogK	A numeric vector with the log10-transformed equilibrium coefficients of the species formation reactions.
SpecDeltaH	A numeric vector with the change in enthalpy of the species formation reactions.
SpecTempKelvin	A numeric vector with the temperatures (in Kelvin) corresponding to ‘SpecDeltaH‘ values of the species formation reactions.

SpecMCR (optional) A character vector with the indices of the mass compartments the new species are associated with. Only needs to be specified if 'SpecMCName' is not specified.

InSpec A logical value indicating if this is a species formation reaction indicated from the parameter file ('TRUE', the default) or a reaction that was added from another process such as 'ExpandWHAM' ('FALSE'). This should usually only be 'FALSE' when another function is calling this function, such as 'ExpandWHAM'.

DoCheck A logical value indicating whether checks should be performed on the incoming and outgoing problem objects. Defaults to 'TRUE', as you usually want to make sure something isn't awry, but the value is often set to 'FALSE' when used internally (like in DefineProblem) so the problem is only checked once at the end.

SpeciesToRemove
A character or integer vector indicating the names or indices (respectively) of the species formation reactions to remove.

Value

'ThisProblem', with the species reaction(s) changed.

See Also

Other problem manipulation functions: BlankProblem(), Components, CriticalValues, InLabs, InVars, MassCompartments, Phases, SpecialDefs

Examples

```
print(carbonate_system_problem$Spec)
my_new_problem = carbonate_system_problem
my_new_problem = AddInComps(ThisProblem = my_new_problem,
                            InCompName = "Ca",
                            InCompCharge = 2,
                            InCompMCName = "Water",
                            InCompType = "MassBal",
                            InCompActCorr = "Debye")
my_new_problem = AddSpecies(
  ThisProblem = my_new_problem,
  SpecEquation = c("CaCO3 = 1 * Ca + 1 * CO3",
                   "CaHCO3 = 1 * Ca + 1 * H + 1 * CO3"),
  SpecMCName = "Water",
  SpecActCorr = "Debye",
  SpecLogK = c(3.22, 11.44),
  SpecDeltaH = c(14951, -3664),
  SpecTempKelvin = 298.15
)
print(my_new_problem$Spec)
my_new_problem = RemoveSpecies(ThisProblem = my_new_problem,
                               SpeciesToRemove = "CaCO3")
print(my_new_problem$Spec)
```

water_MC_problem *Water mass compartment only problem*

Description

An example BLMEngineInR problem object, which describes a water-only system with no input variables or components yet, and the input label "ID".

Usage

```
water_MC_problem
```

Format

An object of class `list` of length 24.

water_problem *Water-only problem*

Description

An example BLMEngineInR problem object, which describes a water-only system with pH and temperature are supplied as input variables, and the input label "ID" is supplied as well. The only reaction is water dissociation (hydroxide "OH" formation reaction).

Usage

```
water_problem
```

Format

An object of class `list` of length 24.

WriteDetailedFile *Write a VERY Detailed Output File*

Description

This will write an output XLSX file with everything that is returned by the 'BLM' function. This includes inputs, concentrations, activities, etc.

Usage

```
WriteDetailedFile(
  OutList,
  FileName,
  AdditionalInfo = paste0("Saved on: ", Sys.time())
)
```

Arguments

OutList	The list object returned by the BLM function.
FileName	The name of the file you'd like to write.
AdditionalInfo	This vector will be included in the "Additional Info". By default, it will give the date/time the file was saved.

Value

Returns TRUE (invisibly) if successful.

WriteInputFile *Write a BLM input file*

Description

This function will take a BLM inputs list object and turn it into an input file, effectively doing the opposite of 'GetData'.

Usage

```
WriteInputFile(AllInput, ThisProblem, InputFile)
```

Arguments

AllInput	A list object with a structure like that returned by 'GetData()'.
ThisProblem	A list object with a structure like that returned by
InputFile	'BlankProblem()'.

Value

TRUE (invisibly) if successful.

Examples

```
tf = tempfile()
myinputfile = system.file("extdata", "InputFiles",
                          "carbonate_system_test.blm4",
                          package = "BLMEngineInR",
                          mustWork = TRUE)
myinputs = GetData(InputFile = myinputfile,
                   ThisProblem = carbonate_system_problem)
WriteInputFile(AllInput = myinputs, ThisProblem = carbonate_system_problem,
               InputFile = tf)
scan(tf, what = character(), sep = "\n")
scan(myinputfile, what = character(), sep = "\n")
file.remove(tf)
```

WriteParamFile *Write a BLM Parameter File*

Description

This function will take a BLM chemical problem list object and turn it into a parameter file, effectively doing the opposite of 'DefineProblem'.

Usage

```
WriteParamFile(ThisProblem, ParamFile, Notes = ThisProblem$Notes)
```

Arguments

ThisProblem	A list object with a structure like that returned by 'BlankProblem()'.
ParamFile	a character value, indicating the file path and name of the parameter file to write.
Notes	A character vector of additional notes to include at the bottom of the parameter file. The text "written by USERNAME from R: YYYY-MM-DD HH:MM:SS" will always be written, regardless of the value of this argument. This will be filled in with a "Notes" item in 'ThisProblem', if available.

Value

ThisProblem, with the ParamFile element changed to the ParamFile argument.

Examples

```
tf = tempfile()
WriteParamFile(ThisProblem = carbonate_system_problem, ParamFile = tf)
DefineProblem(ParamFile = tf)
```

`WriteWHAMFile` *Write a WHAM Parameter File*

Description

This function will take a WHAM parameter list object and turn it into a WHAM parameter file, effectively doing the opposite of 'DefineWHAM'.

Usage

```
WriteWHAMFile(ThisWHAM, WHAMFile, Notes = ThisWHAM$Notes)
```

Arguments

`ThisWHAM`	A list object with a structure like that returned by 'BlankWHAM()'.
`WHAMFile`	a character value, indicating the file path and name of the WHAM parameter file to write.
`Notes`	A character vector of additional notes to include at the bottom of the WHAM parameter file. The text "written by USERNAME from R: YYYY-MM-DD HH:MM:SS" will always be written, regardless of the value of this argument. Be default, this will be filled in with a "Notes" item in 'ThisWHAM', if available.

Value

ThisProblem, with the ParamFile element changed to the ParamFile argument.

Examples

```
tf = tempfile()
WriteWHAMFile(ThisWHAM = Cu_full_organic_problem$WHAM, WHAMFile = tf)
DefineWHAM(WHAMFile = tf)
```

Index