BioRTPlusPlus

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 ChemFile Class Reference

Chemical system parameters The class that contains the complete information about the primary species. This file contains the list of primary species including the total concentrations for each primary species. The PrimarySpecies block requires both the name of the primary species and the total concentration. This section also requires the secondary species block to be populated with at least 1 secondary species. For a kinetic simulation, this requires the MineralKinetic block to have at least one mineral entry which contains both a mineral name and a mineral surface area.

`#include <model_inputs.hpp>`

Public Member Functions

- **ChemFile** ()
- **PotionsRunType run_type** ()
- map< string, double > **primary_species** ()

  Primary species map getter.
- vector< string > **secondary_species** ()

  Secondary species vector getter.
- map< string, double > **mineral_surface_areas** ()

  Mineral surface area vector getter.
- map< string, unsigned int > **mineral_map** ()

  Mineral index map getter.
- double **end_time** ()

  End time getter.
- int **num_steps** ()

  Getter for the number of steps in the kinetic simulation.

Static Public Member Functions

- static **ChemFile from_file** (const string &file_path)

  *Load 'chem.yaml' from a file Read a 'chem.yaml' file and produce a **ChemFile** (p. 5) object. This function exits if the 'chem.yaml' file is not formatted correctly.*
- static **ChemFile from_string** (const string &yaml_string)

  *Produce a **ChemFile** (p. 5) object from a string containing the contents of a 'chem.yaml' file.*

3.1.1 Detailed Description

Chemical system parameters The class that contains the complete information about the primary species. This file contains the list of primary species including the total concentrations for each primary species. The PrimarySpecies block requires both the name of the primary species and the total concentration. This section also requires the secondary species block to be populated with at least 1 secondary species. For a kinetic simulation, this requires the MineralKinetic block to have at least one mineral entry which contains both a mineral name and a mineral surface area.

Token specifying the kinetic rate constant in 'cdbs.yaml'

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ChemFile()

```
ChemFile::ChemFile ( )
```

Number of steps to run the simulation

3.1.3 Member Function Documentation

3.1.3.1 end_time()

```
double ChemFile::end_time ( )  [inline]
```

End time getter.

Returns

double The final time (in seconds) of a kinetic simulation

3.1.3.2 from_file()

```
static  ChemFile ChemFile::from_file (
             const string & file_path )  [static]
```

Load 'chem.yaml' from a file Read a 'chem.yaml' file and produce a **ChemFile** (p. 5) object. This function exits if the 'chem.yaml' file is not formatted correctly.

Enum value for whether this is an equilibrium or kinetic simulation

Parameters

file_path	The file path to the 'chem.yaml' file.

3.1.3.3 from_string()

```
static  ChemFile ChemFile::from_string (
             const string & yaml_string )  [static]
```

Produce a **ChemFile** (p. 5) object from a string containing the contents of a 'chem.yaml' file.

Parameters

yaml_string	A string containing the entire contents of 'chem.yaml'

Returns

ChemFile (p. 5)

3.1.3.4 mineral_map()

```
map<string, unsigned int> ChemFile::mineral_map ( )  [inline]
```

Mineral index map getter.

Returns

map<string, unsigned int>

3.1.3.5 mineral_surface_areas()

```
map<string, double> ChemFile::mineral_surface_areas ( )
```

Mineral surface area vector getter.

Returns

map<string, double>

3.1.3.6 num_steps()

```
int ChemFile::num_steps ( )  [inline]
```

Getter for the number of steps in the kinetic simulation.

Returns

int The number of steps for a kinetic simulation

3.1.3.7 primary_species()

```
map<string, double> ChemFile::primary_species ( )
```

Primary species map getter.

Returns

map<string, double>

3.1.3.8 run_type()

```
PotionsRunType ChemFile::run_type ( )
```

Default constructor

3.1.3.9 secondary_species()

```
vector<string> ChemFile::secondary_species ( )
```

Secondary species vector getter.

Returns

vector<string>

The documentation for this class was generated from the following file:

- **model_inputs.hpp**

3.2 ChemicalState Struct Reference

Data structure for storing the chemical state of the system.

```
#include <solution.hpp>
```

Public Member Functions

- **ChemicalState** ()

  Construct a new Chemical State object.
- **ChemicalState** (const vec &conc, const vec &tot_conc)

  Construct a new Chemical State object.

Public Attributes

- vec **concentration**
- vec **total_concentration**

3.2.1 Detailed Description

Data structure for storing the chemical state of the system.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ChemicalState() [1/2]

```
ChemicalState::ChemicalState ( )  [inline]
```

Construct a new Chemical State object.

The vector of total concentrations of each species for mass conservation

3.2.2.2 ChemicalState() [2/2]

```
ChemicalState::ChemicalState (
            const vec & conc,
            const vec & tot_conc )  [inline]
```

Construct a new Chemical State object.

Parameters

conc	
tot_conc	

3.2.3 Member Data Documentation

3.2.3.1 total_concentration

```
vec ChemicalState::total_concentration
```

The vector of concentrations of each species

The documentation for this struct was generated from the following file:

- **solution.hpp**

3.3 Database Class Reference

Data structure for data contained in the chemical database, 'cdbs.yaml' This data structure contains all of the chemical information that the user does not need to change. In the database, there is a list of primary species, secondary species, including the equilibrium parameters, and mineral species including the kinetic and equilibrium parameters.

```
#include <model_inputs.hpp>
```

Public Member Functions

- **Database** ()
  *Construct a new **Database** (p. 10) object.*
- **Database** (vector< string > primary_species, map< string, **SecondarySpecies** > secondary_species, map< string, **MineralSpecies** > mineral_species)
  *Construct a new **Database** (p. 10) object.*
- vector< string > **get_primary_species** ()
  Get the primary species vector.
- map< string, **SecondarySpecies** > **get_secondary_species** ()
  Get the secondary species map.
- map< string, **MineralSpecies** > **get_mineral_species** ()
  Get the mineral kinetic species map.

Static Public Member Functions

- static **Database** **from_file** (const string &file_path)
  Read a 'cdbs.yaml' file.
- static **Database** **from_string** (const string &yaml_string)
  *Produce a **Database** (p. 10) object from a string containing the contents of the 'cdbs.yaml' file.*

3.3.1 Detailed Description

Data structure for data contained in the chemical database, 'cdbs.yaml' This data structure contains all of the chemical information that the user does not need to change. In the database, there is a list of primary species, secondary species, including the equilibrium parameters, and mineral species including the kinetic and equilibrium parameters.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Database() [1/2]

```
Database::Database ( )
```

Construct a new **Database** (p. 10) object.

Map of mineral species information in the database

3.3.2.2 Database() [2/2]

```
Database::Database (
            vector< string > primary_species,
            map< string,  SecondarySpecies > secondary_species,
            map< string,  MineralSpecies > mineral_species )  [inline]
```

Construct a new **Database** (p. 10) object.

Parameters

primary_species	
secondary_species	
mineral_species	

3.3.3 Member Function Documentation

3.3.3.1 from_file()

```
static  Database Database::from_file (
            const string & file_path )  [static]
```

Read a 'cdbs.yaml' file.

Parameters

file_path	The file path to the 'cdbs.yaml'

Returns

Database (p. 10)

3.3.3.2 from_string()

```
static Database Database::from_string (
            const string & yaml_string ) [static]
```

Produce a **Database** (p. 10) object from a string containing the contents of the 'cdbs.yaml' file.

Parameters

yaml_string	

Returns

Database (p. 10)

3.3.3.3 get_mineral_species()

```
map<string, MineralSpecies> Database::get_mineral_species ( )
```

Get the mineral kinetic species map.

Returns

map<string, MineralSpecies>

3.3.3.4 get_primary_species()

```
vector<string> Database::get_primary_species ( )
```

Get the primary species vector.

Returns

vector<string>

3.3.3.5 get_secondary_species()

```
map<string, SecondarySpecies> Database::get_secondary_species ( )
```

Get the secondary species map.

Returns

map<string, SecondarySpecies>

The documentation for this class was generated from the following file:

- **model_inputs.hpp**

3.4 EquilibriumConstants Struct Reference

Data structure containing the constants relating to the equilibrium parameters.

`#include <solution.hpp>`

Public Member Functions

- **EquilibriumConstants** ()

  Construct a new Equilibrium Constants object.
- **EquilibriumConstants** (const mat &stoich_mat, const vec &eq_consts)

  Construct a new Equilibrium Constants object.

Public Attributes

- mat **stoichiometry_matrix**
- mat **stoich_null_space**
- vec **conc_particular_solution**
- vec **log_eq_consts**

3.4.1 Detailed Description

Data structure containing the constants relating to the equilibrium parameters.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 EquilibriumConstants() [1/2]

```
EquilibriumConstants::EquilibriumConstants ( )  [inline]
```

Construct a new Equilibrium Constants object.

Vector of the Base-10 logarithms of the equilibrium constants

3.4.2.2 EquilibriumConstants() [2/2]

```
EquilibriumConstants::EquilibriumConstants (
            const mat & stoich_mat,
            const vec & eq_consts )
```

Construct a new Equilibrium Constants object.

Parameters

stoich_mat	
eq_consts	

3.4.3 Member Data Documentation

3.4.3.1 conc_particular_solution

```
vec EquilibriumConstants::conc_particular_solution
```

Null space of the stoichiometry matrix, used in solving the system

3.4.3.2 log_eq_consts

```
vec EquilibriumConstants::log_eq_consts
```

Particular solution of the underdetermined equilibrium constants

3.4.3.3 stoich_null_space

```
mat EquilibriumConstants::stoich_null_space
```

Stoichiometry matrix with shape (number of species)x(number of secondary species)

The documentation for this struct was generated from the following file:

- **solution.hpp**

3.5 KineticConstants Struct Reference

Data structure containing the kinetic reaction coefficients.

```
#include <solution.hpp>
```

Public Member Functions

- **KineticConstants** ()

  Construct a new Kinetic Constants object.
- **KineticConstants** (const mat &kin_mat, const vec &kin_consts, const vec & **eq_const**)

  Construct a new Kinetic Constants object.

Public Attributes

- mat **kin_mat**
- vec **kin_const**
- vec **eq_const**

3.5.1 Detailed Description

Data structure containing the kinetic reaction coefficients.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 KineticConstants() [1/2]

```
KineticConstants::KineticConstants ( )  [inline]
```

Construct a new Kinetic Constants object.

Vector of the base-10 logarithm of the equilibrium constants

3.5.2.2 KineticConstants() [2/2]

```
KineticConstants::KineticConstants (
            const mat & kin_mat,
            const vec & kin_consts,
            const vec & eq_const )  [inline]
```

Construct a new Kinetic Constants object.

Parameters

kin_mat	
kin_consts	
eq_const	

3.5.3 Member Data Documentation

3.5.3.1 eq_const

```
vec KineticConstants::eq_const
```

Vector of the Base-10 logarithm of the kinetic rate constants

3.5.3.2 kin_const

```
vec KineticConstants::kin_const
```

Kinetic stoichiometry matrix where each row is the formula for the ionic activity product for each mineral

The documentation for this struct was generated from the following file:

- **solution.hpp**

3.6 MineralSpecies Struct Reference

Data structure containing information for a mineral species.

```
#include <model_inputs.hpp>
```

Public Member Functions

- **MineralSpecies** ()
  Construct a new Mineral Species object.
- **MineralSpecies** (const map< string, double > &stoichiometry, double **molar_volume**, double **molar_mass**, double rate_constant, double equilibrium_constant)
  Construct a new Mineral Species object.
- bool **operator==** (const **MineralSpecies** &other)
  *Compare two **MineralSpecies** (p. 16) objects.*

Static Public Member Functions

- static **MineralSpecies** **from_yaml** (const YAML::Node &node)
  *Produce a **MineralSpecies** (p. 16) object from a YAML node object.*

Public Attributes

- map< string, double > **stoichiometry**
- double **molar_volume**
- double **molar_mass**
- double **log_kin_const**
- double **log_eq_const**

3.6.1 Detailed Description

Data structure containing information for a mineral species.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 MineralSpecies()

```
MineralSpecies::MineralSpecies (
            const map< string, double > & stoichiometry,
            double molar_volume,
            double molar_mass,
            double rate_constant,
            double equilibrium_constant )  [inline]
```

Construct a new Mineral Species object.

Parameters

stoichiometry	
molar_volume	
molar_mass	
rate_constant	
equilibrium_constant	

3.6.3 Member Function Documentation

3.6.3.1 from_yaml()

```
static MineralSpecies MineralSpecies::from_yaml (
            const YAML::Node & node )  [static]
```

Produce a **MineralSpecies** (p. 16) object from a YAML node object.

Base-10 logarithm of the equilibrium constant

Parameters

node	

Returns

MineralSpecies (p. 16)

3.6.3.2 operator==()

```
bool MineralSpecies::operator== (
            const MineralSpecies & other )
```

Compare two **MineralSpecies** (p. 16) objects.

Parameters

other	

Returns

true

false

3.6.4 Member Data Documentation

3.6.4.1 log_eq_const

```
double MineralSpecies::log_eq_const
```

Base-10 logarithm of the kinetic constant

3.6.4.2 log_kin_const

```
double MineralSpecies::log_kin_const
```

Mass of a mole of mineral, in g/mol

3.6.4.3 molar_mass

```
double MineralSpecies::molar_mass
```

Volume per mole of mineral, in cm^3/mol

3.6.4.4 molar_volume

```
double MineralSpecies::molar_volume
```

Map for stoichiometry of the dissolution of this mineral

The documentation for this struct was generated from the following file:

- **model_inputs.hpp**

3.7 ModelInputs Class Reference

Wrapper for the model inputs Data structure containing the input data to run the simulations. This is a way to interface with both the **ChemFile** (p. 5) and **Database** (p. 10) objects to simplify the model interface, as this class contains methods to construct the mass conservation, equilibrium, and kinetic constants.

```
#include <model_inputs.hpp>
```

Public Member Functions

- **ModelInputs** ()

  *Default constructor for **ModelInputs** (p. 19).*
- **ModelInputs** (**ChemFile chem**, **Database** cdbs)

  *Construct the model inputs from a **ChemFile** (p. 5) and **Database** (p. 10).*
- **PotionsRunType run_type** ()

  Returns the the model run type, Equilibrium or Kinetic.
- **ChemicalState initial_chem_state** ()

  Determine initial chemical state.
- **EquilibriumConstants equilibrium_constants** ()

  Construct the equilibrium parameters for this chemical system.
- **TotalConstants total_constants** ()

  Construct the mass conservation matrix.
- **KineticConstants kinetic_constants** ()

  Construct the kinetic mass constants.
- vector< string > **species_names** ()

  Get a vector of the species names for this chemical system in the order that they appear in the solution.
- map< string, unsigned int > **species_map** ()

  Get a map of aqueous species names and indices in the system.
- **ChemFile chem** ()

  *Getter for the **ChemFile** (p. 5) object.*
- **Database database** ()

  *Getter for the **Database** (p. 10) object.*
- map< string, double > **surface_areas** ()

  Getter for the mineral surface areas for this chemical system.

Static Public Member Functions

- static **ModelInputs read_inputs** (const string &inputName)

  Read input files.

3.7.1 Detailed Description

Wrapper for the model inputs Data structure containing the input data to run the simulations. This is a way to interface with both the **ChemFile** (p. 5) and **Database** (p. 10) objects to simplify the model interface, as this class contains methods to construct the mass conservation, equilibrium, and kinetic constants.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 ModelInputs() [1/2]

```
ModelInputs::ModelInputs ( )
```

Default constructor for **ModelInputs** (p. 19).

The **Database** (p. 10) object for the simulation

3.7.2.2 ModelInputs() [2/2]

```
ModelInputs::ModelInputs (
              ChemFile chem,
              Database cdbs )
```

Construct the model inputs from a **ChemFile** (p. 5) and **Database** (p. 10).

Construct the model inputs from a **ChemFile** (p. 5) object and **Database** (p. 10) object. There is post-processing with some of this data to be done.

Parameters

chem	The **ChemFile** (p. 5) read from an 'chem.yaml' file
cdbs	The **Database** (p. 10) read from the 'cdbs.yaml' file

Returns

none

3.7.3 Member Function Documentation

3.7.3.1 chem()

```
ChemFile ModelInputs::chem ( )  [inline]
```

Getter for the **ChemFile** (p. 5) object.

Returns

The **ChemFile** (p. 5) object for this chemical system

3.7.3.2 database()

```
Database ModelInputs::database ( )  [inline]
```

Getter for the **Database** (p. 10) object.

Returns

The **Database** (p. 10) object for this chemical system

3.7.3.3 equilibrium_constants()

```
EquilibriumConstants ModelInputs::equilibrium_constants ( )
```

Construct the equilibrium parameters for this chemical system.

This function reads the list of primary and secondary species from the **ChemFile** (p. 5) object and determines the stoichiometry and equilibrium parameters from the **Database** (p. 10) object. The stoichiometry matrix describes how changes in concentrations propagate throughout the system, and the equilibrium parameters sets the requirement for a system at equilibrium.

Returns

The **EquilibriumConstants** (p. 13) object for this chemical system

3.7.3.4 initial_chem_state()

```
ChemicalState ModelInputs::initial_chem_state ( )
```

Determine initial chemical state.

Read the initial total concentrations from the chem.yaml file and initialize the total concentration vector. The concentration vector is set at a uniform value of 1e-7 moles per liter. This number is arbitrary and only intended so that the concentrations are not zero, which could be a problem later on.

Returns

Returns the total concentrations at the start and a uniform concentration vector

3.7.3.5 kinetic_constants()

```
KineticConstants ModelInputs::kinetic_constants ( )
```

Construct the kinetic mass constants.

This function construct the kinetic constants data structures for this chemical system. This function takes the minerals listed in the **ChemFile** (p. 5) object and reads the stoichiometry, rate constant, and kinetic equilibrium parameters from the **Database** (p. 10) object. These parameters are used in the kinetic portions of the model, which uses a Transition-State-Theory-type rate law.

Returns

The **KineticConstants** (p. 14) object for this chemical system

3.7.3.6 read_inputs()

```
static ModelInputs ModelInputs::read_inputs (
            const string & inputName )  [static]
```

Read input files.

Static method for reading the input files

Parameters

input_name	The name of the folder in the input folder containing your input files

Returns

The model inputs object after having read the inputs

3.7.3.7 run_type()

```
PotionsRunType ModelInputs::run_type ( )
```

Returns the the model run type, Equilibrium or Kinetic.

Returns

The enum type of the model run

3.7.3.8 species_map()

```
map<string, unsigned int> ModelInputs::species_map ( )
```

Get a map of aqueous species names and indices in the system.

Returns

A map of aqueous species in this system.

3.7.3.9 species_names()

```
vector<string> ModelInputs::species_names ( )
```

Get a vector of the species names for this chemical system in the order that they appear in the solution.

Returns

An ordered vector of species names

3.7.3.10 surface_areas()

```
map<string, double> ModelInputs::surface_areas ( )  [inline]
```

Getter for the mineral surface areas for this chemical system.

Returns

The vector of mineral surface areas for this chemical system

3.7.3.11 total_constants()

```
TotalConstants ModelInputs::total_constants ( )
```

Construct the mass conservation matrix.

This function constructs the mass conservation matrix from the list of primary species in this model. This matrix represents how each of the secondary species relate to the primary species. This model does not explicitly track H+ and instead uses a charge balance for all species for the entire solution. This allows the system to have closure while also imposing another physical law on the system.

Returns

The **TotalConstants** (p. 28) object for this chemical system

The documentation for this class was generated from the following file:

- **model_inputs.hpp**

3.8 RunOptions Class Reference

Wrapper class for the command line options.

```
#include <potions_input.hpp>
```

Public Member Functions

- **RunOptions** ()
  Construct a new Run Options object.
- string **output_name** ()
  Getter for the input folder name.

Static Public Member Functions

- static **RunOptions from_arguments** (int argc, char ∗argv[ ])
  Directly handle the command line parameters.

3.8.1 Detailed Description

Wrapper class for the command line options.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 RunOptions()

```
RunOptions::RunOptions ( )  [inline]
```

Construct a new Run Options object.

Name of the input folder name

3.8.3 Member Function Documentation

3.8.3.1 from_arguments()

```
static  RunOptions RunOptions::from_arguments (
            int argc,
            char * argv[] )  [static]
```

Directly handle the command line parameters.

Parameters

argc	The number of command line arguments
argv	Arguments passed to the command line

Returns

RunOptions (p. 24)

3.8.3.2 output_name()

```
string RunOptions::output_name ( )  [inline]
```

Getter for the input folder name.

Returns

string

The documentation for this class was generated from the following file:

- **potions_input.hpp**

3.9 SecondarySpecies Struct Reference

The secondary species information struct This data structure contains the information for how to represent a secondary species in terms of primary species.

`#include <model_inputs.hpp>`

Public Member Functions

- **SecondarySpecies** ()
  Construct a new Secondary Species object.
- **SecondarySpecies** (map< string, double > &stoichiometry, double **log_eq_const**)
  Construct a new Secondary Species object.
- bool **operator==** (const **SecondarySpecies** &other)
  *Compare two **SecondarySpecies** (p. 26) objects.*

Static Public Member Functions

- static **SecondarySpecies** **from_yaml** (const YAML::Node &node)
  *Produce a **SecondarySpecies** (p. 26) object from a YAML node object.*

Public Attributes

- map< string, double > **stoichiometry**
- double **log_eq_const**

3.9.1 Detailed Description

The secondary species information struct This data structure contains the information for how to represent a secondary species in terms of primary species.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 SecondarySpecies()

```
SecondarySpecies::SecondarySpecies (
            map< string, double > & stoichiometry,
            double log_eq_const )  [inline]
```

Construct a new Secondary Species object.

Parameters

stoichiometry	The stoichiometric coefficients for this object
log_eq_const	The base-10 logarithm of the equilibrium constant for this species

3.9.3 Member Function Documentation

3.9.3.1 from_yaml()

```
static SecondarySpecies SecondarySpecies::from_yaml (
            const YAML::Node & node ) [static]
```

Produce a **SecondarySpecies** (p. 26) object from a YAML node object.

The base-10 logarithm of the equilibrium constant

Parameters

node	The YAML node object representing the secondary species object

Returns

SecondarySpecies (p. 26)

3.9.3.2 operator==()

```
bool SecondarySpecies::operator== (
            const SecondarySpecies & other )
```

Compare two **SecondarySpecies** (p. 26) objects.

Parameters

other	

Returns

true

false

3.9.4 Member Data Documentation

3.9.4.1 log_eq_const

```
double SecondarySpecies::log_eq_const
```

Map of stoichiometric coefficients of primary species

The documentation for this struct was generated from the following file:

- **model_inputs.hpp**

3.10 TotalConstants Struct Reference

Data structure containing the mass conservation matrix.

```
#include <solution.hpp>
```

Public Member Functions

- **TotalConstants** ()
  Construct a new Total Constants object.
- **TotalConstants** (const mat &tot_mat)
  Construct a new Total Constants object.

Public Attributes

- mat **tot_mat**

3.10.1 Detailed Description

Data structure containing the mass conservation matrix.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 TotalConstants() [1/2]

```
TotalConstants::TotalConstants ( )  [inline]
```

Construct a new Total Constants object.

Matrix of shape (number of primary species)x(number of species)

3.10.2.2 TotalConstants() [2/2]

```
TotalConstants::TotalConstants (
            const mat & tot_mat )  [inline]
```

Construct a new Total Constants object.

Parameters

tot_mat	

The documentation for this struct was generated from the following file:

- **solution.hpp**

Chapter 4

File Documentation

4.1 constants.hpp File Reference

Enumerations

- enum **PotionsRunType** { **EQUILIBRIUM** = 0 , **KINETIC** = 1 }

  Type of simulation to run.

Variables

- const double **TOLERANCE** = 1e-10
- const int **MAX_ITERS** = 50

4.2 equilibrium.hpp File Reference

```
#include "armadillo"
#include "potions.hpp"
```

Functions

- **ChemicalState** **solve_equilibrium** (const vec &tot_conc, const **EquilibriumConstants** &eq_params, const **TotalConstants** &tot_params)

  Solve for the thermodynamic equilibrium This functions solves for the equilibrium speciation for primary and secondary species. In general solving for equilibrium is a highly nonlinear process that involves numerical optimization. The number of equations that we must solve is equal to the number of primary species plus the number of secondary species, and the governing equations come from satisfying both a mass balance and the equilibrium relations:

4.2.1 Function Documentation

4.2.1.1 solve_equilibrium()

```
ChemicalState solve_equilibrium (
            const vec & tot_conc,
            const  EquilibriumConstants & eq_params,
            const  TotalConstants & tot_params )
```

Solve for the thermodynamic equilibrium This functions solves for the equilibrium speciation for primary and secondary species. In general solving for equilibrium is a highly nonlinear process that involves numerical optimization. The number of equations that we must solve is equal to the number of primary species plus the number of secondary species, and the governing equations come from satisfying both a mass balance and the equilibrium relations:

- For ever primary species, there is one mass conservation relationship∗
  - ∗For H+, we use a charge balance rather than a conservation of H+
- For every secondary species, there is an equilibrium relationship to satisfy

Parameters

tot_conc	The vector of total concentrations for each species for the mass balance.
eq_params	The stoichiometry matrix and equilibrium constants for the chemical system
tot_params	The mass concentration matrix

Returns

ChemicalState (p. 8) The **ChemicalState** (p. 8) object of equilibrium concentrations and a copy of the total concentrations at equilibrium.

4.3 kinetic.hpp File Reference

```
#include "potions.hpp"
```

Functions

- vec **kinetic_rate** (const vec &tot_conc, const vec &surface_area, const **EquilibriumConstants** &eq, const **KineticConstants** &kin_params, const **TotalConstants** &tot_params)

  Rate of change of species in terms of moles per liter Use the Transition-State-Theory type rate law to determine the rate of change of concentrations of each species in the system. This includes both primary and secondary species. Note: each evaluation of this functions involves solving equilibrium.
- **ChemicalState solve_kinetic_equilibrium** (const **ChemicalState** &chem, const vec &surfaceArea, const **KineticConstants** &kin, const **EquilibriumConstants** &eq, const **TotalConstants** &tot, double dt)

4.3.1 Function Documentation

4.3.1.1 kinetic_rate()

```
vec kinetic_rate (
            const vec & tot_conc,
            const vec & surface_area,
            const  EquilibriumConstants & eq,
            const  KineticConstants & kin_params,
            const  TotalConstants & tot_params )
```

Rate of change of species in terms of moles per liter Use the Transition-State-Theory type rate law to determine the rate of change of concentrations of each species in the system. This includes both primary and secondary species. Note: each evaluation of this functions involves solving equilibrium.

Parameters

tot_conc	The vector of total concentrations with the length being the number of species
surface_area	The vector of mineral surface areas with the length being the number of minerals
eq	The equilibrium parameters for this system
kin_params	The kinetic matrix, rate constants, and kinetic equilibrium constants
tot_params	The mass conservation matrix wrapper structure

Returns

vec The rate of changes of species with the length being the number of species.

4.4 model_inputs.hpp File Reference

```
#include <filesystem>
#include <string>
#include <map>
#include <vector>
#include "yaml-cpp/yaml.h"
```

Classes

- class **ChemFile**

  Chemical system parameters The class that contains the complete information about the primary species. This file contains the list of primary species including the total concentrations for each primary species. The PrimarySpecies block requires both the name of the primary species and the total concentration. This section also requires the secondary species block to be populated with at least 1 secondary species. For a kinetic simulation, this requires the MineralKinetic block to have at least one mineral entry which contains both a mineral name and a mineral surface area.
- struct **SecondarySpecies**

  The secondary species information struct This data structure contains the information for how to represent a secondary species in terms of primary species.
- struct **MineralSpecies**

  Data structure containing information for a mineral species.
- class **Database**

  Data structure for data contained in the chemical database, 'cdbs.yaml' This data structure contains all of the chemical information that the user does not need to change. In the database, there is a list of primary species, secondary species, including the equilibrium parameters, and mineral species including the kinetic and equilibrium parameters.

- class **ModelInputs**

  *Wrapper for the model inputs Data structure containing the input data to run the simulations. This is a way to interface with both the **ChemFile** (p. 5) and **Database** (p. 10) objects to simplify the model interface, as this class contains methods to construct the mass conservation, equilibrium, and kinetic constants.*

Functions

- bool **operator==** (const map< string, **SecondarySpecies** > &l, const map< string, **SecondarySpecies** > &r)

  Compare two maps of secondary species objects.
- bool **operator==** (const map< string, **MineralSpecies** > &l, const map< string, **MineralSpecies** > &r)

  *Compare maps of **MineralSpecies** (p. 16) objects.*

Variables

- const string **SIMULATION_TYPE_TOKEN** = "SimulationType"
- const string **PRIMARY_SECTION_TOKEN** = "PrimarySpecies"
- const string **SECONDARY_SECTION_TOKEN** = "SecondarySpecies"
- const string **MINERAL_SECTION_TOKEN** = "MineralSpecies"
- const string **SIMTYPE_EQ_TOKEN** = "Equilibrium"
- const string **SIMTYPE_KIN_TOKEN** = "Kinetic"
- const string **END_TIME_TOKEN** = "EndTime"
- const string **NUM_STEPS** = "NumSteps"
- const string **DBS_ID_TOKEN** = "Database"
- const string **DBS_PRIMARY_SECTION_TOKEN** = "PrimarySpecies"
- const string **DBS_SECONDARY_SECTION_TOKEN** = "SecondarySpecies"
- const string **DBS_MINERAL_SECTION_TOKEN** = "MineralSpecies"
- const string **DBS_STOICHIOMETRY_TOKEN** = "stoichiometry"
- const string **DBS_LOGK_TOKEN** = "logK"
- const string **DBS_MOLAR_MASS_TOKEN** = "molarMass"
- const string **DBS_MOLAR_VOLUME_TOKEN** = "molarVolume"
- const string **DBS_RATE_CONST_TOKEN** = "rate"

4.4.1 Function Documentation

4.4.1.1 operator==() [1/2]

```
bool operator== (
            const map< string,  MineralSpecies > & l,
            const map< string,  MineralSpecies > & r )
```

Compare maps of **MineralSpecies** (p. 16) objects.

Parameters

l	
r	

Returns

true

false

4.4.1.2 operator==() [2/2]

```
bool operator== (
            const map< string,  SecondarySpecies > & l,
            const map< string,  SecondarySpecies > & r )
```

Compare two maps of secondary species objects.

Parameters

l	
r	

Returns

true

false

4.4.2 Variable Documentation

4.4.2.1 DBS_ID_TOKEN

```
const string DBS_ID_TOKEN = "Database"
```

Token for the number of steps to use in the simulation

4.4.2.2 DBS_LOGK_TOKEN

```
const string DBS_LOGK_TOKEN = "logK"
```

Token specifying a stoichiometry block in the database

4.4.2.3 DBS_MINERAL_SECTION_TOKEN

```
const string DBS_MINERAL_SECTION_TOKEN = "MineralSpecies"
```

Token specifying the secondary block in 'cdbs.yaml'

4.4.2.4 DBS_MOLAR_MASS_TOKEN

```
const string DBS_MOLAR_MASS_TOKEN = "molarMass"
```

Token specifying the equilibrium constant in the database

4.4.2.5 DBS_MOLAR_VOLUME_TOKEN

```
const string DBS_MOLAR_VOLUME_TOKEN = "molarVolume"
```

Token specifying the molar mass the 'cdbs.yaml'

4.4.2.6 DBS_PRIMARY_SECTION_TOKEN

```
const string DBS_PRIMARY_SECTION_TOKEN = "PrimarySpecies"
```

Token for the database ID token

4.4.2.7 DBS_RATE_CONST_TOKEN

```
const string DBS_RATE_CONST_TOKEN = "rate"
```

Token specifying the molar volume in the 'cdbs.yaml'

4.4.2.8 DBS_SECONDARY_SECTION_TOKEN

```
const string DBS_SECONDARY_SECTION_TOKEN = "SecondarySpecies"
```

Token specifying the primary section list in 'cdbs.yaml'

4.4.2.9 DBS_STOICHIOMETRY_TOKEN

```
const string DBS_STOICHIOMETRY_TOKEN = "stoichiometry"
```

Token specifying the mineral section in 'cdbs.yaml'

4.4.2.10 END_TIME_TOKEN

```
const string END_TIME_TOKEN = "EndTime"
```

Token specifying a kinetic simulation in 'chem.yaml'

4.4.2.11 MINERAL_SECTION_TOKEN

```
const string MINERAL_SECTION_TOKEN = "MineralSpecies"
```

Token for secondary species block in 'chem.yaml'

4.4.2.12 NUM_STEPS

```
const string NUM_STEPS = "NumSteps"
```

Token specifying the time to run the simulation to

4.4.2.13 PRIMARY_SECTION_TOKEN

```
const string PRIMARY_SECTION_TOKEN = "PrimarySpecies"
```

Name of the parameter for species

4.4.2.14 SECONDARY_SECTION_TOKEN

```
const string SECONDARY_SECTION_TOKEN = "SecondarySpecies"
```

Token for the primary species block in 'chem.yaml'

4.4.2.15 SIMTYPE_EQ_TOKEN

```
const string SIMTYPE_EQ_TOKEN = "Equilibrium"
```

Token for mineral species block in 'chem.yaml'

4.4.2.16 SIMTYPE_KIN_TOKEN

```
const string SIMTYPE_KIN_TOKEN = "Kinetic"
```

Token specifying an equilibrium simulation in 'chem.yaml'

4.5 potions_input.hpp File Reference

```
#include <string>
#include "constants.hpp"
```

Classes

- class **RunOptions**

  Wrapper class for the command line options.

4.6 solution.hpp File Reference

```
#include <tuple>
#include "armadillo"
```

Classes

- struct **ChemicalState**

  Data structure for storing the chemical state of the system.
- struct **EquilibriumConstants**

  Data structure containing the constants relating to the equilibrium parameters.
- struct **TotalConstants**

  Data structure containing the mass conservation matrix.
- struct **KineticConstants**

  Data structure containing the kinetic reaction coefficients.

4.7 utils.hpp File Reference

```
#include <vector>
#include <string>
#include <filesystem>
#include <utility>
#include "model_inputs.hpp"
```

Functions

- string **get_output_file_path** (string simulation_name)

  Construct the output file path Construct the output file path to write the simulation results to. The file path has the format "<sim_name>_YYYYMMDD_HHMMSS_results.csv", where YYYYMMDD_HHMMSS is a time stamp for the simulation time and 'sim_name' is the name of the input folder. This is intended so that the results name will always be unique and always be idenfitiable. The results are written in a comma-separated format, so the file is a CSV file.
- bool **compare_doubles** (double a, double b)

  Compare floating point numbers safely.
- bool **save_equilibrium_results** (const **ChemicalState** &chms, vector< string > species, const string &file_path)

  Write the results of an equilibrium simulation to a file This function writes the equilibrium concentrations to a CSV file with 2 columns:
- bool **save_kinetic_results** (const vector< pair< double, **ChemicalState** >> &res, const vector< string > &species, const string &filePath)

  Write the results of a kinetic simulation to a CSV file This function writes the results of the simulation to a CSV file as a table. Only the concentrations, not total concentrations, are written to the file. The first row of the CSV file are the column names. The first is the time (in seconds), followed by each chemical species name. The columns in this file are:
- int **get_charge** (const string &name)

  Get the ionic charge of a chemical species Determine the ionic charge of a chemical species by its formula. Example: get_charge("Cl-") -> -1 get_charge("Ca++") -> 2 get_charge("H2CO3") -> 0.
- template<typename T >
  void **print_matrix** (const Mat< T > &m)

  Print the contents of a matrix to the standard output.
- bool **plot_results** (const vector< pair< double, **ChemicalState** >> &results, const vector< string > &species_names)

  Produce a plot of the kinetic simulation results Produce a time series line plot of the concentrations of each chemical species from a kinetic simulation.
- bool **operator==** (const map< string, double > &l, const map< string, double > &r)

  Utility function to compare maps of doubles.

4.7.1 Function Documentation

4.7.1.1 compare_doubles()

```
bool compare_doubles (
            double a,
            double b )
```

Compare floating point numbers safely.

Parameters

a	
b	

Returns

true

false

4.7.1.2 get_charge()

```
int get_charge (
            const string & name )
```

Get the ionic charge of a chemical species Determine the ionic charge of a chemical species by its formula. Example: get_charge("Cl-") -> -1 get_charge("Ca++") -> 2 get_charge("H2CO3") -> 0.

Parameters

name	

Returns

int

4.7.1.3 get_output_file_path()

```
string get_output_file_path (
            string simulation_name )
```

Construct the output file path Construct the output file path to write the simulation results to. The file path has the format "<sim_name>_YYYYMMDD_HHMMSS_results.csv", where YYYYMMDD_HHMMSS is a time stamp for the simulation time and 'sim_name' is the name of the input folder. This is intended so that the results name will always be unique and always be idenfitiable. The results are written in a comma-separated format, so the file is a CSV file.

Parameters

simulation_name	The name of the input folders

Returns

string

4.7.1.4 operator==()

```
bool operator== (
            const map< string, double > & l,
            const map< string, double > & r )
```

Utility function to compare maps of doubles.

Parameters

l	
r	

Returns

true

false

4.7.1.5 plot_results()

```
bool plot_results (
            const vector< pair< double,  ChemicalState >> & results,
            const vector< string > & species_names )
```

Produce a plot of the kinetic simulation results Produce a time series line plot of the concentrations of each chemical species from a kinetic simulation.

Parameters

results	The vector of pairs of simulation step results
species_names	The vector of species names

Returns

true

false

4.7.1.6 print_matrix()

```
template<typename T >
void print_matrix (
            const Mat< T > & m )
```

Print the contents of a matrix to the standard output.

Template Parameters

T	

Parameters

m	

4.7.1.7 save_equilibrium_results()

```
bool save_equilibrium_results (
            const  ChemicalState & chms,
            vector< string > species,
            const string & file_path )
```

Write the results of an equilibrium simulation to a file This function writes the equilibrium concentrations to a CSV file with 2 columns:

1. The name of each of the species
2. The concentration (in molars) of each species

Parameters

chms	
species	
file_path	

Returns

true

false

4.7.1.8 save_kinetic_results()

```
bool save_kinetic_results (
            const vector< pair< double,  ChemicalState >> & res,
            const vector< string > & species,
            const string & filePath )
```

Write the results of a kinetic simulation to a CSV file This function writes the results of the simulation to a CSV file as a table. Only the concentrations, not total concentrations, are written to the file. The first row of the CSV file are the column names. The first is the time (in seconds), followed by each chemical species name. The columns in this file are:

1. The time (in seconds) for the simulation step
2. From now on, the concentrations of each species

Parameters

res	
species	
filePath	

Returns

true
false

Index