Ecosystem

Version 0.0 beta

Generated by Doxygen 1.8.3.1

Mon Sep 21 2015 16:56:05

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 ARNOLDI_DATA Struct Reference

Data structure for the construction of the Krylov subspaces for a linear system.

```
#include <lark.h>
```

Public Attributes

- int k

  Desired size of the Krylov subspace.
- int iter

  Actual size of the Krylov subspace.
- double beta

  Normalization parameter.
- double hp1

  Additional row element of H (separate storage for holding)
- bool Output = true

  True = print messages to console.
- std::vector< Matrix< double > > Vk

  (N) x (k) orthonormal vector basis stored as a vector of column matrices
- Matrix< double > Hkp1

  (k+1) x (k) upper Hessenberg matrix
- Matrix< double > yk

  (k) x (1) vector search direction
- Matrix< double > e1

  (k) x (1) orthonormal vector with 1 in first position
- Matrix< double > w

  (N) x (1) interim result of the matrix_vector multiplication
- Matrix< double > v

  (N) x (1) holding cell for the column entries of Vk and other interims
- Matrix< double > sum

  (N) x (1) running sum of subspace vectors for use in altering w

4.1.1 Detailed Description

Data structure for the construction of the Krylov subspaces for a linear system.

C-style object used in conjunction with the Arnoldi algorithm to construct an orthonormal basis and upper Hessenberg representation of a given linear operator. This is used to solve a linear system both iteratively (i.e., in conjunction with GMRESLP) and directly (i.e., in conjunction with FOM). Alternatively, you can just store the factorized components for later use in another routine.

4.1.2 Member Data Documentation

4.1.2.1 double ARNOLDI_DATA::beta

Normalization parameter.

4.1.2.2 Matrix<double> ARNOLDI_DATA::e1

(k) x (1) orthonormal vector with 1 in first position

4.1.2.3 Matrix<double> ARNOLDI_DATA::Hkp1

(k+1) x (k) upper Hessenberg matrix

4.1.2.4 double ARNOLDI_DATA::hp1

Additional row element of H (separate storage for holding)

4.1.2.5 int ARNOLDI_DATA::iter

Actual size of the Krylov subspace.

4.1.2.6 int ARNOLDI_DATA::k

Desired size of the Krylov subspace.

4.1.2.7 bool ARNOLDI_DATA::Output = true

True = print messages to console.

4.1.2.8 Matrix<double> ARNOLDI_DATA::sum

(N) x (1) running sum of subspace vectors for use in altering w

4.1.2.9 Matrix<double> ARNOLDI_DATA::v

(N) x (1) holding cell for the column entries of Vk and other interims

4.1.2.10 std::vector< Matrix<double> > ARNOLDI_DATA::Vk

(N) x (k) orthonormal vector basis stored as a vector of column matrices

4.1.2.11 Matrix<double> ARNOLDI_DATA::w

(N) x (1) interim result of the matrix_vector multiplication

4.1.2.12 Matrix<double> ARNOLDI_DATA::yk

(k) x (1) vector search direction

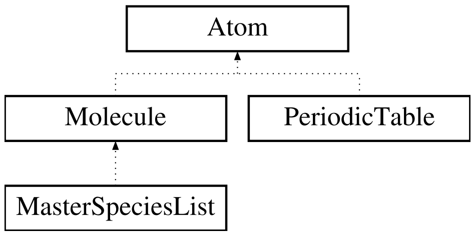
The documentation for this struct was generated from the following file:

- lark.h

4.2 Atom Class Reference

```
#include <eel.h>
```

Inheritance diagram for Atom:



Public Member Functions

- Atom ()
- ∼Atom ()
- Atom (std::string Name)
- Atom (int number)
- void Register (std::string Symbol)
- void Register (int number)
- void editAtomicWeight (double AW)
- void editOxidationState (int state)
- void editProtons (int proton)
- void editNeutrons (int neutron)
- void editElectrons (int electron)
- void editValence (int val)
- void removeProton ()
- void removeNeutron ()
- void removeElectron ()
- double AtomicWeight ()
- int OxidationState ()
- int Protons ()
- int Neutrons ()
- int Electrons ()
- int BondingElectrons ()
- std::string AtomName ()
- std::string AtomSymbol ()
- std::string AtomCategory ()
- std::string AtomState ()
- int AtomicNumber ()
- void DisplayInfo ()

Protected Attributes

- double atomic_weight
- int oxidation_state
- int protons
- int neutrons
- int electrons
- int valence_e

Private Attributes

- std::string Name
- std::string Symbol
- std::string Category
- std::string NaturalState
- int atomic_number

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Atom::Atom ( )

4.2.1.2 Atom::∼Atom ( )

4.2.1.3 Atom::Atom (std::string *Name*)

4.2.1.4 Atom::Atom (int *number*)

4.2.2 Member Function Documentation

4.2.2.1 std::string Atom::AtomCategory ( )

4.2.2.2 int Atom::AtomicNumber ( )

4.2.2.3 double Atom::AtomicWeight ( )

4.2.2.4 std::string Atom::AtomName ( )

4.2.2.5 std::string Atom::AtomState ( )

4.2.2.6 std::string Atom::AtomSymbol ( )

4.2.2.7 int Atom::BondingElectrons ( )

4.2.2.8 void Atom::DisplayInfo ( )

4.2.2.9 void Atom::editAtomicWeight (double *AW*)

4.2.2.10 void Atom::editElectrons (int *electron*)

4.2.2.11 void Atom::editNeutrons (int *neutron*)

4.2.2.12 void Atom::editOxidationState (int *state*)

4.2.2.13 void Atom::editProtons (int *proton*)

4.2.2.14 void Atom::editValence (int *val*)

4.2.2.15 int Atom::Electrons ( )

4.2.2.16 int Atom::Neutrons ( )

4.2.2.17 int Atom::OxidationState ( )

4.2.2.18 int Atom::Protons ( )

4.2.2.19 void Atom::Register (std::string *Symbol*)

4.2.2.20 void Atom::Register (int *number*)

4.2.2.21 void Atom::removeElectron ( )

4.2.2.22 void Atom::removeNeutron ( )

4.2.2.23 void Atom::removeProton ( )

4.2.3 Member Data Documentation

4.2.3.1 int Atom::atomic_number `[private]`

4.2.3.2 double Atom::atomic_weight `[protected]`

4.2.3.3 std::string Atom::Category `[private]`

4.2.3.4 int Atom::electrons `[protected]`

4.2.3.5 std::string Atom::Name `[private]`

4.2.3.6 std::string Atom::NaturalState `[private]`

4.2.3.7 int Atom::neutrons `[protected]`

4.2.3.8 int Atom::oxidation_state `[protected]`

4.2.3.9 int Atom::protons `[protected]`

4.2.3.10 std::string Atom::Symbol `[private]`

4.2.3.11 int Atom::valence_e `[protected]`

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

- eel.h
- eel.cpp

4.3 BACKTRACK_DATA Struct Reference

Data structure for the implementation of Backtracking Linesearch.

```
#include <lark.h>
```

Public Attributes

- double alpha = 1e-4

  Scaling parameter for determination of search step size.
- double rho = 0.1

  Scaling parameter for to change step size by.
- double lambdaMin =DBL_EPSILON

  Smallest allowable step length.
- double normFkp1

  New residual norm of the Newton step.
- bool constRho = false

  True = use a constant value for rho.
- Matrix< double > Fk

  Old residual vector of the Newton step.
- Matrix< double > xk

  Old solution vector of the Newton step.

4.3.1 Detailed Description

Data structure for the implementation of Backtracking Linesearch.

C-style object used in conjunction with the Backtracking Linesearch algorithm to smooth out convergence of Netwon based iterative methods for non-linear systems of equations. The actual algorithm has been separated from the interior of the Newton method so that it can be included in any future Newton based iterative methods being developed.

4.3.2 Member Data Documentation

4.3.2.1 double BACKTRACK_DATA::alpha = 1e-4

Scaling parameter for determination of search step size.

4.3.2.2 bool BACKTRACK_DATA::constRho = false

True = use a constant value for rho.

4.3.2.3 Matrix<double> BACKTRACK_DATA::Fk

Old residual vector of the Newton step.

4.3.2.4 double BACKTRACK_DATA::lambdaMin =DBL_EPSILON

Smallest allowable step length.

4.3.2.5 double BACKTRACK_DATA::normFkp1

New residual norm of the Newton step.

4.3.2.6 double BACKTRACK_DATA::rho = 0.1

Scaling parameter for to change step size by.

4.3.2.7 Matrix< double > BACKTRACK_DATA::xk

Old solution vector of the Newton step.

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

- lark.h

4.4 BiCGSTAB_DATA Struct Reference

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

```
#include <lark.h>
```

Public Attributes

- int maxit = 0

  Maximum allowable iterations - default = min(2∗vector_size,1000)
- int iter = 0

  Actual number of iterations.
- bool breakdown

  Boolean to determine if the method broke down.
- double alpha

  Step size parameter for next solution.
- double beta

  Step size parameter for search direction.
- double rho

  Scaling parameter for alpha and beta.
- double rho_old

  Previous scaling parameter for alpha and beta.
- double omega

  Scaling parameter and additional step length.
- double omega_old

  Previous scaling parameter and step length.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm.
- double relres

  Relative residual norm.
- double relres_base

  Initial residual norm.
- double bestres

  Best found residual norm.
- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

Best found solution to the linear system.

- Matrix< double > r

  Residual vector for the linear system.
- Matrix< double > r0

  Initial residual vector.
- Matrix< double > v

  Search direction for p.
- Matrix< double > p

  Search direction for updating.
- Matrix< double > y

  Preconditioned search direction.
- Matrix< double > s

  Residual updating vector.
- Matrix< double > z

  Preconditioned residual updating vector.
- Matrix< double > t

  Search direction for resdidual updates.

4.4.1 Detailed Description

Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.

C-style object used in conjunction with the Bi-Conjugate Gradient STABalized (BiCGSTAB) algorithm to solve a linear system of equations. This algorithm is generally more efficient than any GMRES or GCR variant, but may not always reduce the residual at each step. However, if used with preconditioning, then this algorithm is very efficient, especially when used for solving grid-based linear systems.

4.4.2 Member Data Documentation

4.4.2.1 double BiCGSTAB_DATA::alpha

Step size parameter for next solution.

4.4.2.2 double BiCGSTAB_DATA::bestres

Best found residual norm.

4.4.2.3 Matrix<double> BiCGSTAB_DATA::bestx

Best found solution to the linear system.

4.4.2.4 double BiCGSTAB_DATA::beta

Step size parameter for search direction.

4.4.2.5 bool BiCGSTAB_DATA::breakdown

Boolean to determine if the method broke down.

4.4.2.6 int BiCGSTAB_DATA::iter = 0

Actual number of iterations.

4.4.2.7 int BiCGSTAB_DATA::maxit = 0

Maximum allowable iterations - default = min(2∗vector_size,1000)

4.4.2.8 double BiCGSTAB_DATA::omega

Scaling parameter and additional step length.

4.4.2.9 double BiCGSTAB_DATA::omega_old

Previous scaling parameter and step length.

4.4.2.10 bool BiCGSTAB_DATA::Output = true

True = print messages to console.

4.4.2.11 Matrix<double> BiCGSTAB_DATA::p

Search direction for updating.

4.4.2.12 Matrix<double> BiCGSTAB_DATA::r

Residual vector for the linear system.

4.4.2.13 Matrix<double> BiCGSTAB_DATA::r0

Initial residual vector.

4.4.2.14 double BiCGSTAB_DATA::relres

Relative residual norm.

4.4.2.15 double BiCGSTAB_DATA::relres_base

Initial residual norm.

4.4.2.16 double BiCGSTAB_DATA::res

Absolute residual norm.

4.4.2.17 double BiCGSTAB_DATA::rho

Scaling parameter for alpha and beta.

4.4.2.18 double BiCGSTAB_DATA::rho_old

Previous scaling parameter for alpha and beta.

4.4.2.19 Matrix<double> BiCGSTAB_DATA::s

Residual updating vector.

4.4.2.20 Matrix<double> BiCGSTAB_DATA::t

Search direction for resdidual updates.

4.4.2.21 double BiCGSTAB_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

4.4.2.22 double BiCGSTAB_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.4.2.23 Matrix<double> BiCGSTAB_DATA::v

Search direction for p.

4.4.2.24 Matrix<double> BiCGSTAB_DATA::x

Current solution to the linear system.

4.4.2.25 Matrix<double> BiCGSTAB_DATA::y

Preconditioned search direction.

4.4.2.26 Matrix<double> BiCGSTAB_DATA::z

Preconditioned residual updating vector.

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

- lark.h

4.5 CGS_DATA Struct Reference

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

```
#include <lark.h>
```

Public Attributes

- int maxit = 0

    Maximum allowable iterations - default = min(2∗vector_size,1000)

- int iter = 0

  Actual number of iterations.
- bool breakdown

  Boolean to determine if the method broke down.
- double alpha

  Step size parameter for next solution.
- double beta

  Step size parameter for search direction.
- double rho

  Scaling parameter for alpha and beta.
- double sigma

  Scaling parameter and additional step length.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm.
- double relres

  Relative residual norm.
- double relres_base

  Initial residual norm.
- double bestres

  Best found residual norm.
- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual vector for the linear system.
- Matrix< double > r0

  Initial residual vector.
- Matrix< double > u

  Search direction for v.
- Matrix< double > w

  Updates sigma and u.
- Matrix< double > v

  Search direction for x.
- Matrix< double > p

  Preconditioning result for w, z, and matvec for Ax.
- Matrix< double > c

  Holds the matvec result between A and p.
- Matrix< double > z

  Full search direction for x.

4.5.1 Detailed Description

Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.

C-style object to be used in conjunction with the Conjugate Gradient Squared (CGS) algorithm to solve linear systems of equations. This algorithm is slightly less computational work than BiCGSTAB, but is much less stable. As a result, I do not recommend using this algorithm unless you also use some form of preconditioning.

4.5.2 Member Data Documentation

4.5.2.1 double CGS_DATA::alpha

Step size parameter for next solution.

4.5.2.2 double CGS_DATA::bestres

Best found residual norm.

4.5.2.3 Matrix<double> CGS_DATA::bestx

Best found solution to the linear system.

4.5.2.4 double CGS_DATA::beta

Step size parameter for search direction.

4.5.2.5 bool CGS_DATA::breakdown

Boolean to determine if the method broke down.

4.5.2.6 Matrix<double> CGS_DATA::c

Holds the matvec result between A and p.

4.5.2.7 int CGS_DATA::iter = 0

Actual number of iterations.

4.5.2.8 int CGS_DATA::maxit = 0

Maximum allowable iterations - default = min(2∗vector_size,1000)

4.5.2.9 bool CGS_DATA::Output = true

True = print messages to console.

4.5.2.10 Matrix<double> CGS_DATA::p

Preconditioning result for w, z, and matvec for Ax.

4.5.2.11 Matrix< double > CGS_DATA::r

Residual vector for the linear system.

4.5.2.12 Matrix< double > CGS_DATA::r0

Initial residual vector.

4.5.2.13 double CGS_DATA::relres

Relative residual norm.

4.5.2.14 double CGS_DATA::relres_base

Initial residual norm.

4.5.2.15 double CGS_DATA::res

Absolute residual norm.

4.5.2.16 double CGS_DATA::rho

Scaling parameter for alpha and beta.

4.5.2.17 double CGS_DATA::sigma

Scaling parameter and additional step length.

4.5.2.18 double CGS_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

4.5.2.19 double CGS_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.5.2.20 Matrix< double > CGS_DATA::u

Search direction for v.

4.5.2.21 Matrix< double > CGS_DATA::v

Search direction for x.

4.5.2.22 Matrix< double > CGS_DATA::w

Updates sigma and u.

4.5.2.23 Matrix<double> CGS_DATA::x

Current solution to the linear system.

4.5.2.24 Matrix<double> CGS_DATA::z

Full search direction for x.

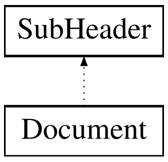
The documentation for this struct was generated from the following file:

- lark.h

4.6 Document Class Reference

```
#include <yaml_wrapper.h>
```

Inheritance diagram for Document:



Public Member Functions

- Document ()
- ∼Document ()
- Document (const Document &doc)
- Document (std::string name)
- Document (const KeyValueMap &map)
- Document (std::string name, const KeyValueMap &map)
- Document (std::string key, const Header &head)
- Document & operator= (const Document &doc)
- ValueTypePair & operator[] (const std::string key)
- ValueTypePair operator[] (const std::string key) const
- Header & operator() (const std::string key)
- Header operator() (const std::string key) const
- std::map< std::string, Header > & getHeadMap ()
- KeyValueMap & getDataMap ()
- Header & getHeader (std::string key)
- std::map< std::string, Header >
  ::const_iterator end () const
- std::map< std::string, Header >
  ::iterator end ()
- std::map< std::string, Header >
  ::const_iterator begin () const
- std::map< std::string, Header >
  ::iterator begin ()
- void clear ()
- void resetKeys ()
- void changeKey (std::string oldKey, std::string newKey)
- void revalidateAllKeys ()

- void addPair (std::string key, std::string val)
- void addPair (std::string key, std::string val, int t)
- void setName (std::string name)
- void setAlias (std::string alias)
- void setNameAliasPair (std::string n, std::string a, int s)
- void setState (int state)
- void DisplayContents ()
- void addHeadKey (std::string key)
- void copyAnchor2Alias (std::string alias, Header &ref)
- int size ()
- std::string getName ()
- std::string getAlias ()
- int getState ()
- bool isAlias ()
- bool isAnchor ()
- Header & getAnchoredHeader (std::string alias)
- Header & getHeadFromSubAlias (std::string alias)

Private Attributes

- std::map< std::string, Header > Head_Map

Additional Inherited Members

4.6.1 Constructor & Destructor Documentation

4.6.1.1 Document::Document ( )

4.6.1.2 Document::∼Document ( )

4.6.1.3 Document::Document (const Document & *doc*)

4.6.1.4 Document::Document (std::string *name*)

4.6.1.5 Document::Document (const KeyValueMap & *map*)

4.6.1.6 Document::Document (std::string *name,* const KeyValueMap & *map*)

4.6.1.7 Document::Document (std::string *key,* const Header & *head*)

4.6.2 Member Function Documentation

4.6.2.1 void Document::addHeadKey (std::string *key*)

4.6.2.2 void Document::addPair (std::string *key,* std::string *val*)

4.6.2.3 void Document::addPair (std::string *key,* std::string *val,* int *t*)

4.6.2.4 std::map< std::string, Header >::const_iterator Document::begin ( ) const

4.6.2.5 std::map< std::string, Header >::iterator Document::begin ( )

4.6.2.6 void Document::changeKey (std::string *oldKey,* std::string *newKey*)

4.6.2.7 void Document::clear ( )

4.6.2.8 void Document::copyAnchor2Alias (std::string *alias,* Header & *ref*)

4.6.2.9 void Document::DisplayContents ( )

4.6.2.10 std::map< std::string, Header >::const_iterator Document::end ( ) const

4.6.2.11 std::map< std::string, Header >::iterator Document::end ( )

4.6.2.12 std::string Document::getAlias ( )

4.6.2.13 Header & Document::getAnchoredHeader (std::string *alias*)

4.6.2.14 KeyValueMap & Document::getDataMap ( )

4.6.2.15 Header & Document::getHeader (std::string *key*)

4.6.2.16 Header & Document::getHeadFromSubAlias (std::string *alias*)

4.6.2.17 std::map< std::string, Header > & Document::getHeadMap ( )

4.6.2.18 std::string Document::getName ( )

4.6.2.19 int Document::getState ( )

4.6.2.20 bool Document::isAlias ( )

4.6.2.21 bool Document::isAnchor ( )

4.6.2.22 Header & Document::operator() (const std::string *key*)

4.6.2.23 Header Document::operator() (const std::string *key*) const

4.6.2.24 Document & Document::operator= (const Document & *doc*)

4.6.2.25 ValueTypePair & Document::operator[] (const std::string *key*)

4.6.2.26 ValueTypePair Document::operator[] (const std::string *key*) const

4.6.2.27 void Document::resetKeys ( )

4.6.2.28 void Document::revalidateAllKeys ( )

4.6.2.29 void Document::setAlias (std::string *alias*)

4.6.2.30 void Document::setName (std::string *name*)

4.6.2.31 void Document::setNameAliasPair (std::string *n,* std::string *a,* int *s*)

4.6.2.32 void Document::setState (int *state*)

4.6.2.33 int Document::size ( )

4.6.3 Member Data Documentation

4.6.3.1 std::map<std::string, Header> Document::Head_Map [private]

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

- yaml_wrapper.h
- yaml_wrapper.cpp

4.7 DOGFISH_DATA Struct Reference

```
#include <dogfish.h>
```

Public Attributes

- unsigned long int total_steps = 0
- double time_old = 0.0
- double time = 0.0
- bool Print2File = true
- bool Print2Console = true
- bool DirichletBC = false
- bool NonLinear = false
- double t_counter = 0.0
- double t_print
- int NumComp
- double end_time
- double total_sorption_old
- double total_sorption
- double fiber_length
- double fiber_diameter
- FILE ∗ OutputFile
- double(∗ eval_R)(int i, int l, const void ∗data)
- double(∗ eval_DI)(int i, int l, const void ∗data)
- double(∗ eval_kf)(int i, const void ∗data)
- double(∗ eval_qs)(int i, const void ∗data)
- const void ∗ user_data
- std::vector< FINCH_DATA > finch_dat
- std::vector< DOGFISH_PARAM > param_dat

4.7.1 Member Data Documentation

4.7.1.1 bool DOGFISH_DATA::DirichletBC = false

4.7.1.2 double DOGFISH_DATA::end_time

4.7.1.3 double(∗ DOGFISH_DATA::eval_DI)(int i, int l, const void ∗data)

4.7.1.4 double(∗ DOGFISH_DATA::eval_kf)(int i, const void ∗data)

4.7.1.5 double(∗ DOGFISH_DATA::eval_qs)(int i, const void ∗data)

4.7.1.6 double(∗ DOGFISH_DATA::eval_R)(int i, int l, const void ∗data)

4.7.1.7 double DOGFISH_DATA::fiber_diameter

4.7.1.8 double DOGFISH_DATA::fiber_length

4.7.1.9 std::vector< FINCH_DATA > DOGFISH_DATA::finch_dat

4.7.1.10 bool DOGFISH_DATA::NonLinear = false

4.7.1.11 int DOGFISH_DATA::NumComp

4.7.1.12 FILE* DOGFISH_DATA::OutputFile

4.7.1.13 std::vector< DOGFISH_PARAM > DOGFISH_DATA::param_dat

4.7.1.14 bool DOGFISH_DATA::Print2Console = true

4.7.1.15 bool DOGFISH_DATA::Print2File = true

4.7.1.16 double DOGFISH_DATA::t_counter = 0.0

4.7.1.17 double DOGFISH_DATA::t_print

4.7.1.18 double DOGFISH_DATA::time = 0.0

4.7.1.19 double DOGFISH_DATA::time_old = 0.0

4.7.1.20 double DOGFISH_DATA::total_sorption

4.7.1.21 double DOGFISH_DATA::total_sorption_old

4.7.1.22 unsigned long int DOGFISH_DATA::total_steps = 0

4.7.1.23 const void* DOGFISH_DATA::user_data

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

- dogfish.h

4.8 DOGFISH_PARAM Struct Reference

```
#include <dogfish.h>
```

Public Attributes

- double intraparticle_diffusion
- double film_transfer_coeff
- double surface_concentration
- double initial_sorption
- double sorbed_molefraction
- Molecule species

4.8.1 Member Data Documentation

4.8.1.1 double DOGFISH_PARAM::film_transfer_coeff

4.8.1.2 double DOGFISH_PARAM::initial_sorption

4.8.1.3 double DOGFISH_PARAM::intraparticle_diffusion

4.8.1.4 double DOGFISH_PARAM::sorbed_molefraction

4.8.1.5 Molecule DOGFISH_PARAM::species

4.8.1.6 double DOGFISH_PARAM::surface_concentration

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

- dogfish.h

4.9 FINCH_DATA Struct Reference

```
#include <finch.h>
```

Public Attributes

- int d = 0
- double dt = 0.0125
- double dt_old = 0.0125
- double T = 1.0
- double dz = 0.1
- double L = 1.0
- double s = 1.0
- double t = 0.0
- double t_old = 0.0
- double uT = 0.0
- double uT_old = 0.0
- double uAvg = 0.0
- double uAvg_old = 0.0
- double uIC = 0.0
- double vIC = 1.0
- double DIC = 1.0
- double kIC = 1.0
- double RIC = 1.0
- double uo = 1.0
- double vo = 1.0
- double Do = 1.0
- double ko = 1.0
- double Ro = 1.0
- double kfn = 1.0
- double kfnp1 = 1.0
- double lambda_I
- double lambda_E
- int LN = 10
- bool CN = true
- bool Update = false
- bool Dirichlet = false
- bool CheckMass = false
- bool ExplicitFlux = false

- bool Iterative = true
- bool SteadyState = false
- bool NormTrack = true
- double beta = 0.5
- double tol_rel = 1e-6
- double tol_abs = 1e-6
- int max_iter = 20
- int total_iter = 0
- int nl_method = FINCH_Picard
- std::vector< double > CL_I
- std::vector< double > CL_E
- std::vector< double > CC_I
- std::vector< double > CC_E
- std::vector< double > CR_I
- std::vector< double > CR_E
- std::vector< double > fL_I
- std::vector< double > fL_E
- std::vector< double > fC_I
- std::vector< double > fC_E
- std::vector< double > fR_I
- std::vector< double > fR_E
- std::vector< double > OI
- std::vector< double > OE
- std::vector< double > NI
- std::vector< double > NE
- std::vector< double > MI
- std::vector< double > ME
- std::vector< double > uz_l_I
- std::vector< double > uz_lm1_I
- std::vector< double > uz_lp1_I
- std::vector< double > uz_l_E
- std::vector< double > uz_lm1_E
- std::vector< double > uz_lp1_E
- Matrix< double > unm1
- Matrix< double > un
- Matrix< double > unp1
- Matrix< double > u_star
- Matrix< double > ubest
- Matrix< double > vn
- Matrix< double > vnp1
- Matrix< double > Dn
- Matrix< double > Dnp1
- Matrix< double > kn
- Matrix< double > knp1
- Matrix< double > Sn
- Matrix< double > Snp1
- Matrix< double > Rn
- Matrix< double > Rnp1
- Matrix< double > Fn
- Matrix< double > Fnp1
- Matrix< double > gI
- Matrix< double > gE
- Matrix< double > res
- Matrix< double > pres
- int(∗ callroutine)(const void ∗user_data)

- int(∗ setic)(const void ∗user_data)
- int(∗ settime)(const void ∗user_data)
- int(∗ setpreprocess)(const void ∗user_data)
- int(∗ solve)(const void ∗user_data)
- int(∗ setparams)(const void ∗user_data)
- int(∗ discretize)(const void ∗user_data)
- int(∗ setbcs)(const void ∗user_data)
- int(∗ evalres)(const Matrix< double > &x, Matrix< double > &res, const void ∗user_data)
- int(∗ evalprecon)(const Matrix< double > &b, Matrix< double > &p, const void ∗user_data)
- int(∗ setpostprocess)(const void ∗user_data)
- int(∗ resettime)(const void ∗user_data)
- PICARD_DATA picard_dat
- PJFNK_DATA pjfnk_dat
- const void ∗ param_data

4.9.1 Member Data Documentation

4.9.1.1 double FINCH_DATA::beta = 0.5

4.9.1.2 int(∗ FINCH_DATA::callroutine)(const void ∗user_data)

4.9.1.3 std::vector<double> FINCH_DATA::CC_E

4.9.1.4 std::vector<double> FINCH_DATA::CC_I

4.9.1.5 bool FINCH_DATA::CheckMass = false

4.9.1.6 std::vector<double> FINCH_DATA::CL_E

4.9.1.7 std::vector<double> FINCH_DATA::CL_I

4.9.1.8 bool FINCH_DATA::CN = true

4.9.1.9 std::vector<double> FINCH_DATA::CR_E

4.9.1.10 std::vector<double> FINCH_DATA::CR_I

4.9.1.11 int FINCH_DATA::d = 0

4.9.1.12 double FINCH_DATA::DIC = 1.0

4.9.1.13 bool FINCH_DATA::Dirichlet = false

4.9.1.14 int(∗ FINCH_DATA::discretize)(const void ∗user_data)

4.9.1.15 Matrix<double> FINCH_DATA::Dn

4.9.1.16 Matrix<double> FINCH_DATA::Dnp1

4.9.1.17 double FINCH_DATA::Do = 1.0

4.9.1.18 double FINCH_DATA::dt = 0.0125

4.9.1.19 double FINCH_DATA::dt_old = 0.0125

4.9.1.20 double FINCH_DATA::dz = 0.1

4.9.1.21 int(∗ FINCH_DATA::evalprecon)(const Matrix< double > &b, Matrix< double > &p, const void ∗user_data)

4.9.1.22 int(∗ FINCH_DATA::evalres)(const Matrix< double > &x, Matrix< double > &res, const void ∗user_data)

4.9.1.23 bool FINCH_DATA::ExplicitFlux = false

4.9.1.24 std::vector<double> FINCH_DATA::fC_E

4.9.1.25 std::vector<double> FINCH_DATA::fC_I

4.9.1.26 std::vector<double> FINCH_DATA::fL_E

4.9.1.27 std::vector<double> FINCH_DATA::fL_I

4.9.1.28 Matrix<double> FINCH_DATA::Fn

4.9.1.29 Matrix<double> FINCH_DATA::Fnp1

4.9.1.30 std::vector<double> FINCH_DATA::fR_E

4.9.1.31 std::vector<double> FINCH_DATA::fR_I

4.9.1.32 Matrix<double> FINCH_DATA::gE

4.9.1.33 Matrix<double> FINCH_DATA::gI

4.9.1.34 bool FINCH_DATA::Iterative = true

4.9.1.35 double FINCH_DATA::kfn = 1.0

4.9.1.36 double FINCH_DATA::kfnp1 = 1.0

4.9.1.37 double FINCH_DATA::kIC = 1.0

4.9.1.38 Matrix<double> FINCH_DATA::kn

4.9.1.39 Matrix<double> FINCH_DATA::knp1

4.9.1.40 double FINCH_DATA::ko = 1.0

4.9.1.41 double FINCH_DATA::L = 1.0

4.9.1.42 double FINCH_DATA::lambda_E

4.9.1.43 double FINCH_DATA::lambda_I

4.9.1.44 int FINCH_DATA::LN = 10

4.9.1.45 int FINCH_DATA::max_iter = 20

4.9.1.46 std::vector<double> FINCH_DATA::ME

4.9.1.47 std::vector<double> FINCH_DATA::MI

4.9.1.48 std::vector< double > FINCH_DATA::NE

4.9.1.49 std::vector< double > FINCH_DATA::NI

4.9.1.50 int FINCH_DATA::nl_method = FINCH_Picard

4.9.1.51 bool FINCH_DATA::NormTrack = true

4.9.1.52 std::vector< double > FINCH_DATA::OE

4.9.1.53 std::vector< double > FINCH_DATA::OI

4.9.1.54 const void∗ FINCH_DATA::param_data

4.9.1.55 PICARD_DATA FINCH_DATA::picard_dat

4.9.1.56 PJFNK_DATA FINCH_DATA::pjfnk_dat

4.9.1.57 Matrix< double > FINCH_DATA::pres

4.9.1.58 Matrix< double > FINCH_DATA::res

4.9.1.59 int(∗ FINCH_DATA::resettime)(const void ∗user_data)

4.9.1.60 double FINCH_DATA::RIC = 1.0

4.9.1.61 Matrix< double > FINCH_DATA::Rn

4.9.1.62 Matrix< double > FINCH_DATA::Rnp1

4.9.1.63 double FINCH_DATA::Ro = 1.0

4.9.1.64 double FINCH_DATA::s = 1.0

4.9.1.65 int(∗ FINCH_DATA::setbcs)(const void ∗user_data)

4.9.1.66 int(∗ FINCH_DATA::setic)(const void ∗user_data)

4.9.1.67 int(∗ FINCH_DATA::setparams)(const void ∗user_data)

4.9.1.68 int(∗ FINCH_DATA::setpostprocess)(const void ∗user_data)

4.9.1.69 int(∗ FINCH_DATA::setpreprocess)(const void ∗user_data)

4.9.1.70 int(∗ FINCH_DATA::settime)(const void ∗user_data)

4.9.1.71 Matrix< double > FINCH_DATA::Sn

4.9.1.72 Matrix< double > FINCH_DATA::Snp1

4.9.1.73 int(∗ FINCH_DATA::solve)(const void ∗user_data)

4.9.1.74 bool FINCH_DATA::SteadyState = false

4.9.1.75 double FINCH_DATA::T = 1.0

4.9.1.76 double FINCH_DATA::t = 0.0

4.9.1.77 double FINCH_DATA::t_old = 0.0

4.9.1.78 double FINCH_DATA::tol_abs = 1e-6

4.9.1.79 double FINCH_DATA::tol_rel = 1e-6

4.9.1.80 int FINCH_DATA::total_iter = 0

4.9.1.81 Matrix<double> FINCH_DATA::u_star

4.9.1.82 double FINCH_DATA::uAvg = 0.0

4.9.1.83 double FINCH_DATA::uAvg_old = 0.0

4.9.1.84 Matrix<double> FINCH_DATA::ubest

4.9.1.85 double FINCH_DATA::uIC = 0.0

4.9.1.86 Matrix<double> FINCH_DATA::un

4.9.1.87 Matrix<double> FINCH_DATA::unm1

4.9.1.88 Matrix<double> FINCH_DATA::unp1

4.9.1.89 double FINCH_DATA::uo = 1.0

4.9.1.90 bool FINCH_DATA::Update = false

4.9.1.91 double FINCH_DATA::uT = 0.0

4.9.1.92 double FINCH_DATA::uT_old = 0.0

4.9.1.93 std::vector<double> FINCH_DATA::uz_l_E

4.9.1.94 std::vector<double> FINCH_DATA::uz_l_I

4.9.1.95 std::vector<double> FINCH_DATA::uz_lm1_E

4.9.1.96 std::vector<double> FINCH_DATA::uz_lm1_I

4.9.1.97 std::vector<double> FINCH_DATA::uz_lp1_E

4.9.1.98 std::vector<double> FINCH_DATA::uz_lp1_I

4.9.1.99 double FINCH_DATA::vIC = 1.0

4.9.1.100 Matrix<double> FINCH_DATA::vn

4.9.1.101 Matrix<double> FINCH_DATA::vnp1

4.9.1.102 double FINCH_DATA::vo = 1.0

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

- finch.h

4.10 GCR_DATA Struct Reference

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

```
#include <lark.h>
```

Public Attributes

- int restart = -1

  Restart parameter for outer iterations - default = 50.
- int maxit = 0

  Maximum allowable outer iterations.
- int iter_outer = 0

  Number of outer iterations taken.
- int iter_inner = 0

  Number of inner iterations taken.
- int total_iter = 0

  Total number of iterations taken.
- bool breakdown = false

  Boolean to determine if a step has failed.
- double alpha

  Inner iteration step size.
- double beta

  Outer iteration step size.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolute tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm for linear system.
- double relres

  Relative residual norm for linear system.
- double relres_base

  Initial residual norm of the linear system.
- double bestres

  Best found residual norm of the linear system.
- bool Output = true

  True = print messages to the console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual Vector.
- Matrix< double > c_temp

  Temporary c vector to be updated.
- Matrix< double > u_temp

  Temporary u vector to be updated.
- std::vector< Matrix< double > > u

  Vector span for updating x.
- std::vector< Matrix< double > > c

  Vector span for updating r.
- OPTRANS_DATA transpose_dat

  Data structure for Operator Transposition.

4.10.1 Detailed Description

Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.

C-style object used in conjunction with the Generalized Conjugate Residual (GCR) algorithm for solving a non-symmetric linear system of equations. When the linear system in question has a positive-definite-symmetric component to it, then this algorithm is equivalent to GMRESRP. However, it is generally less efficient than GMRESRP and can suffer breakdowns.

4.10.2 Member Data Documentation

4.10.2.1 double GCR_DATA::alpha

Inner iteration step size.

4.10.2.2 double GCR_DATA::bestres

Best found residual norm of the linear system.

4.10.2.3 Matrix<double> GCR_DATA::bestx

Best found solution to the linear system.

4.10.2.4 double GCR_DATA::beta

Outer iteration step size.

4.10.2.5 bool GCR_DATA::breakdown = false

Boolean to determine if a step has failed.

4.10.2.6 std::vector<Matrix<double> > GCR_DATA::c

Vector span for updating r.

4.10.2.7 Matrix<double> GCR_DATA::c_temp

Temporary c vector to be updated.

4.10.2.8 int GCR_DATA::iter_inner = 0

Number of inner iterations taken.

4.10.2.9 int GCR_DATA::iter_outer = 0

Number of outer iterations taken.

4.10.2.10 int GCR_DATA::maxit = 0

Maximum allowable outer iterations.

4.10.2.11 bool GCR_DATA::Output = true

True = print messages to the console.

4.10.2.12 Matrix<double> GCR_DATA::r

Residual Vector.

4.10.2.13 double GCR_DATA::relres

Relative residual norm for linear system.

4.10.2.14 double GCR_DATA::relres_base

Initial residual norm of the linear system.

4.10.2.15 double GCR_DATA::res

Absolute residual norm for linear system.

4.10.2.16 int GCR_DATA::restart = -1

Restart parameter for outer iterations - default = 50.

4.10.2.17 double GCR_DATA::tol_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

4.10.2.18 double GCR_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.10.2.19 int GCR_DATA::total_iter = 0

Total number of iterations taken.

4.10.2.20 OPTRANS_DATA GCR_DATA::transpose_dat

Data structure for Operator Transposition.

4.10.2.21 std::vector<Matrix<double> > GCR_DATA::u

Vector span for updating x.

4.10.2.22 Matrix<double> GCR_DATA::u_temp

Temporary u vector to be updated.

4.10.2.23 Matrix<double> GCR_DATA::x

Current solution to the linear system.

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

- lark.h

4.11 GMRESLP_DATA Struct Reference

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

```
#include <lark.h>
```

Public Attributes

- int restart = -1

  Restart parameter - default = min(vector_size,50)
- int maxit = 0

  Maximum allowable iterations - default = min(vector_size,1000)
- int iter = 0

  Number of iterations needed for convergence.
- int steps = 0

  Total number of gmres iterations and krylov iterations.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Absolution redisual norm of the linear system.
- double relres

  Relative residual norm of the linear system.
- double relres_base

  Initial residual norm of the linear system.
- double bestres

  Best found residual norm of the linear system.
- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual vector for the linear system.
- ARNOLDI_DATA arnoldi_dat

  Data structure for the kyrlov subspace.

4.11.1 Detailed Description

Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.

C-style object used in conjunction with Generalized Minimum RESidual Left-Precondtioned (GMRESLP) and Full Orthogonalization Method (FOM) algorithms to iteratively or directly solve a linear system of equations. When using with GMRESLP, you can only check/observe the linear residuals before a restart or after the Arnoldi space is constructed. This is because this object uses Left-side Preconditioning. A faster routine may be GMRESRP, which is able to construct residuals after each Arnoldi iteration.

4.11.2 Member Data Documentation

4.11.2.1 ARNOLDI_DATA GMRESLP_DATA::arnoldi_dat

Data structure for the kyrlov subspace.

4.11.2.2 double GMRESLP_DATA::bestres

Best found residual norm of the linear system.

4.11.2.3 Matrix<double> GMRESLP_DATA::bestx

Best found solution to the linear system.

4.11.2.4 int GMRESLP_DATA::iter = 0

Number of iterations needed for convergence.

4.11.2.5 int GMRESLP_DATA::maxit = 0

Maximum allowable iterations - default = min(vector_size,1000)

4.11.2.6 bool GMRESLP_DATA::Output = true

True = print messages to console.

4.11.2.7 Matrix<double> GMRESLP_DATA::r

Residual vector for the linear system.

4.11.2.8 double GMRESLP_DATA::relres

Relative residual norm of the linear system.

4.11.2.9 double GMRESLP_DATA::relres_base

Initial residual norm of the linear system.

4.11.2.10 double GMRESLP_DATA::res

Absolution redisual norm of the linear system.

4.11.2.11 int GMRESLP_DATA::restart = -1

Restart parameter - default = min(vector_size,50)

4.11.2.12 int GMRESLP_DATA::steps = 0

Total number of gmres iterations and krylov iterations.

4.11.2.13 double GMRESLP_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

4.11.2.14 double GMRESLP_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.11.2.15 Matrix<double> GMRESLP_DATA::x

Current solution to the linear system.

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

- lark.h

4.12 GMRESR_DATA Struct Reference

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

```
#include <lark.h>
```

Public Attributes

- int gcr_restart = -1

  Number of GCR restarts (default = 50, max = N)
- int gcr_maxit = 0

  Number of GCR iterations.
- int gmres_restart = -1

  Number of GMRES restarts (max = 20)
- int gmres_maxit = 1

  Number of GMRES iterations (max = 5, default = 1)
- int N

  Dimension of the linear system.
- int total_iter

  Total GMRES and GCR iterations.
- int iter_outer

  Total GCR iterations.
- int iter_inner

  Total GMRES iterations.
- bool GCR_Output = true

  True = print GCR messages.
- bool GMRES_Output = false

True = print GMRES messages.

- double gmres_tol = 0.1

  Tolerance relative to GCR iterations.
- double gcr_rel_tol = 1e-6

  Relative outer residual tolerance.
- double gcr_abs_tol = 1e-6

  Absolute outer residual tolerance.
- Matrix$<$ double $>$ arg

  Argument matrix passed between preconditioner and iterator.
- GCR_DATA gcr_dat

  Data structure for the outer GCR steps.
- GMRESRP_DATA gmres_dat

  Data structure for the inner GMRES steps.
- int($*$ matvec)(const Matrix$<$ double $>$ &x, Matrix$<$ double $>$ &Ax, const void $*$matvec_data)

  User supplied matrix-vector product function.
- int($*$ terminal_precon)(const Matrix$<$ double $>$ &r, Matrix$<$ double $>$ &p, const void $*$precon_data)

  Optional user supplied terminal preconditioner.
- const void $*$ matvec_data

  Data structure for the user's matvec function.
- const void $*$ term_precon

  Data structure for the user's terminal preconditioner.

4.12.1 Detailed Description

Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)

C-style object to be used in conjunction with the Generalized Minimum RESidual Recurive (GMRESR) algorithm. Although the name suggests that this method used GMRES recursively, what it is actually doing is nesting GMRESRP iterations inside the GCR method to form a preconditioner for GCR. The name GMRESR came from literature (Vorst and Vuik, "GMRESR: A family of nested GMRES methods", 1991).

4.12.2 Member Data Documentation

4.12.2.1 Matrix$<$double$>$ GMRESR_DATA::arg

Argument matrix passed between preconditioner and iterator.

4.12.2.2 double GMRESR_DATA::gcr_abs_tol = 1e-6

Absolute outer residual tolerance.

4.12.2.3 GCR_DATA GMRESR_DATA::gcr_dat

Data structure for the outer GCR steps.

4.12.2.4 int GMRESR_DATA::gcr_maxit = 0

Number of GCR iterations.

4.12.2.5 bool GMRESR_DATA::GCR_Output = true

True = print GCR messages.

4.12.2.6 double GMRESR_DATA::gcr_rel_tol = 1e-6

Relative outer residual tolerance.

4.12.2.7 int GMRESR_DATA::gcr_restart = -1

Number of GCR restarts (default = 50, max = N)

4.12.2.8 GMRESRP_DATA GMRESR_DATA::gmres_dat

Data structure for the inner GMRES steps.

4.12.2.9 int GMRESR_DATA::gmres_maxit = 1

Number of GMRES iterations (max = 5, default = 1)

4.12.2.10 bool GMRESR_DATA::GMRES_Output = false

True = print GMRES messages.

4.12.2.11 int GMRESR_DATA::gmres_restart = -1

Number of GMRES restarts (max = 20)

4.12.2.12 double GMRESR_DATA::gmres_tol = 0.1

Tolerance relative to GCR iterations.

4.12.2.13 int GMRESR_DATA::iter_inner

Total GMRES iterations.

4.12.2.14 int GMRESR_DATA::iter_outer

Total GCR iterations.

4.12.2.15 int(∗ GMRESR_DATA::matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗matvec_data)

User supplied matrix-vector product function.

4.12.2.16 const void∗ GMRESR_DATA::matvec_data

Data structure for the user's matvec function.

4.12.2.17 int GMRESR_DATA::N

Dimension of the linear system.

4.12.2.18 const void∗ GMRESR_DATA::term_precon

Data structure for the user's terminal preconditioner.

4.12.2.19 int(∗ GMRESR_DATA::terminal_precon)(const Matrix< double > &r, Matrix< double > &p, const void ∗precon_data)

Optional user supplied terminal preconditioner.

4.12.2.20 int GMRESR_DATA::total_iter

Total GMRES and GCR iterations.

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

- lark.h

4.13 GMRESRP_DATA Struct Reference

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

```
#include <lark.h>
```

Public Attributes

- int restart = -1

  Restart parameter - default = min(50,vector_size)
- int maxit = 0

  Maximum allowable outer iterations.
- int iter_outer = 0

  Total number of outer iterations.
- int iter_inner = 0

  Total number of inner iterations.
- int iter_total = 0

  Total number of overall iterations.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolute tolerance for convergence - default = 1e-6.
- double res

  Absolute residual norm for linear system.
- double relres

  Relative residual norm for linear system.
- double relres_base

  Initial residual norm of the linear system.
- double bestres

  Best found residual norm of the linear system.

- bool Output = true

  True = print messages to console.
- Matrix< double > x

  Current solution to the linear system.
- Matrix< double > bestx

  Best found solution to the linear system.
- Matrix< double > r

  Residual vector for the linear system.
- std::vector< Matrix< double > > Vk

  (N x k) orthonormal vector basis
- std::vector< std::vector
  < double > > H

  (k+1 x k) upper Hessenberg storage matrix
- std::vector< std::vector
  < double > > H_bar

  (k+1 x k) Factorized matrix
- std::vector< double > y

  (k x 1) Vector search direction
- std::vector< double > e0

  (k+1 x 1) Normalized vector with residual info
- std::vector< double > e0_bar

  (k+1 x 1) Factorized normal vector
- Matrix< double > w

  (N) x (1) interim result of the matrix_vector multiplication
- Matrix< double > v

  (N) x (1) holding cell for the column entries of Vk and other interims
- Matrix< double > sum

  (N) x (1) running sum of subspace vectors for use in altering w

4.13.1 Detailed Description

Data structure for the Restarted GMRES algorithm with Right Preconditioning.

C-style object used in conjunction with Generalized Minimum RESidual Right Preconditioned (GMRESRP) algorithm to iteratively solve a linear system of equations. Unlike GMRESLP, the GMRESRP method is capable of checking linear residuals at both the inner and outer steps. As a result, this algorithm may terminate earlier than GMRESLP if it has found a suitable solution during one of the inner steps.

4.13.2 Member Data Documentation

4.13.2.1 double GMRESRP_DATA::bestres

Best found residual norm of the linear system.

4.13.2.2 Matrix<double> GMRESRP_DATA::bestx

Best found solution to the linear system.

4.13.2.3 std::vector< double > GMRESRP_DATA::e0

(k+1 x 1) Normalized vector with residual info

4.13.2.4 std::vector< double > GMRESRP_DATA::e0_bar

(k+1 x 1) Factorized normal vector

4.13.2.5 std::vector< std::vector< double > > GMRESRP_DATA::H

(k+1 x k) upper Hessenberg storage matrix

4.13.2.6 std::vector< std::vector< double > > GMRESRP_DATA::H_bar

(k+1 x k) Factorized matrix

4.13.2.7 int GMRESRP_DATA::iter_inner = 0

Total number of inner iterations.

4.13.2.8 int GMRESRP_DATA::iter_outer = 0

Total number of outer iterations.

4.13.2.9 int GMRESRP_DATA::iter_total = 0

Total number of overall iterations.

4.13.2.10 int GMRESRP_DATA::maxit = 0

Maximum allowable outer iterations.

4.13.2.11 bool GMRESRP_DATA::Output = true

True = print messages to console.

4.13.2.12 Matrix<double> GMRESRP_DATA::r

Residual vector for the linear system.

4.13.2.13 double GMRESRP_DATA::relres

Relative residual norm for linear system.

4.13.2.14 double GMRESRP_DATA::relres_base

Initial residual norm of the linear system.

4.13.2.15 double GMRESRP_DATA::res

Absolute residual norm for linear system.

4.13.2.16 int GMRESRP_DATA::restart = -1

Restart parameter - default = min(50,vector_size)

4.13.2.17 Matrix< double > GMRESRP_DATA::sum

(N) x (1) running sum of subspace vectors for use in altering w

4.13.2.18 double GMRESRP_DATA::tol_abs = 1e-6

Absolute tolerance for convergence - default = 1e-6.

4.13.2.19 double GMRESRP_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.13.2.20 Matrix< double > GMRESRP_DATA::v

(N) x (1) holding cell for the column entries of Vk and other interims

4.13.2.21 std::vector< Matrix< double > > GMRESRP_DATA::Vk

(N x k) orthonormal vector basis

4.13.2.22 Matrix< double > GMRESRP_DATA::w

(N) x (1) interim result of the matrix_vector multiplication

4.13.2.23 Matrix< double > GMRESRP_DATA::x

Current solution to the linear system.

4.13.2.24 std::vector< double > GMRESRP_DATA::y

(k x 1) Vector search direction

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

- lark.h

4.14 GPAST_DATA Struct Reference

```
#include <magpie.h>
```

Public Attributes

- double x
- double y
- double He
- double q

- std::vector< double > gama_inf
- double qo
- double Plo
- std::vector< double > po
- double poi
- bool present

4.14.1 Member Data Documentation

4.14.1.1 std::vector<double> GPAST_DATA::gama_inf

4.14.1.2 double GPAST_DATA::He

4.14.1.3 double GPAST_DATA::Plo

4.14.1.4 std::vector<double> GPAST_DATA::po

4.14.1.5 double GPAST_DATA::poi

4.14.1.6 bool GPAST_DATA::present

4.14.1.7 double GPAST_DATA::q

4.14.1.8 double GPAST_DATA::qo

4.14.1.9 double GPAST_DATA::x

4.14.1.10 double GPAST_DATA::y

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

- magpie.h

4.15 GSTA_DATA Struct Reference

```
#include <magpie.h>
```

Public Attributes

- double qmax
- int m
- std::vector< double > dHo
- std::vector< double > dSo

4.15.1 Member Data Documentation

4.15.1.1 std::vector<double> GSTA_DATA::dHo

4.15.1.2 std::vector<double> GSTA_DATA::dSo

4.15.1.3 int GSTA_DATA::m

4.15.1.4 double GSTA_DATA::qmax

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

- magpie.h

4.16 GSTA_OPT_DATA Struct Reference

```
#include <gsta_opt.h>
```

Public Attributes

- int total_eval
- int n_par
- double qmax
- int iso
- std::vector< std::vector
  < double > > Fobj
- std::vector< std::vector
  < double > > q
- std::vector< std::vector
  < double > > P
- std::vector< std::vector
  < double > > best_par
- std::vector< std::vector
  < double > > Kno
- std::vector< std::vector
  < std::vector< double > > > all_pars
- std::vector< std::vector
  < double > > norms
- std::vector< double > opt_qmax

4.16.1 Member Data Documentation

4.16.1.1 std::vector<std::vector<std::vector<double> > > GSTA_OPT_DATA::all_pars

4.16.1.2 std::vector<std::vector<double> > GSTA_OPT_DATA::best_par

4.16.1.3 std::vector<std::vector<double> > GSTA_OPT_DATA::Fobj

4.16.1.4 int GSTA_OPT_DATA::iso

4.16.1.5 std::vector<std::vector<double> > GSTA_OPT_DATA::Kno

4.16.1.6 int GSTA_OPT_DATA::n_par

4.16.1.7 std::vector<std::vector<double> > GSTA_OPT_DATA::norms

4.16.1.8 std::vector<double> GSTA_OPT_DATA::opt_qmax

4.16.1.9 std::vector<std::vector<double> > GSTA_OPT_DATA::P

4.16.1.10 std::vector<std::vector<double> > GSTA_OPT_DATA::q

4.16.1.11 double GSTA_OPT_DATA::qmax

4.16.1.12 int GSTA_OPT_DATA::total_eval

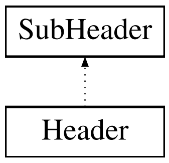
The documentation for this struct was generated from the following file:

- gsta_opt.h

4.17 Header Class Reference

```
#include <yaml_wrapper.h>
```

Inheritance diagram for Header:



Public Member Functions

- Header ()
- ∼Header ()
- Header (const Header &head)
- Header (std::string name)
- Header (const KeyValueMap &map)
- Header (std::string name, const KeyValueMap &map)
- Header (std::string key, const SubHeader &sub)
- Header & operator= (const Header &head)
- ValueTypePair & operator[] (const std::string key)
- ValueTypePair operator[] (const std::string key) const
- SubHeader & operator() (const std::string key)
- SubHeader operator() (const std::string key) const
- std::map< std::string,
  SubHeader > & getSubMap ()
- KeyValueMap & getDataMap ()
- SubHeader & getSubHeader (std::string key)
- std::map< std::string,
  SubHeader >::const_iterator end () const
- std::map< std::string,
  SubHeader >::iterator end ()
- std::map< std::string,
  SubHeader >::const_iterator begin () const
- std::map< std::string,
  SubHeader >::iterator begin ()
- void clear ()
- void resetKeys ()
- void changeKey (std::string oldKey, std::string newKey)
- void addPair (std::string key, std::string val)
- void addPair (std::string key, std::string val, int t)
- void setName (std::string name)
- void setAlias (std::string alias)

- void setNameAliasPair (std::string n, std::string a, int s)
- void setState (int state)
- void DisplayContents ()
- void addSubKey (std::string key)
- void copyAnchor2Alias (std::string alias, SubHeader &ref)
- int size ()
- std::string getName ()
- std::string getAlias ()
- int getState ()
- bool isAlias ()
- bool isAnchor ()
- SubHeader & getAnchoredSub (std::string alias)

Private Attributes

- std::map< std::string, SubHeader > Sub_Map

Additional Inherited Members

4.17.1 Constructor & Destructor Documentation

4.17.1.1 Header::Header ( )

4.17.1.2 Header::∼Header ( )

4.17.1.3 Header::Header (const Header & *head*)

4.17.1.4 Header::Header (std::string *name*)

4.17.1.5 Header::Header (const KeyValueMap & *map*)

4.17.1.6 Header::Header (std::string *name,* const KeyValueMap & *map*)

4.17.1.7 Header::Header (std::string *key,* const SubHeader & *sub*)

4.17.2 Member Function Documentation

4.17.2.1 void Header::addPair (std::string *key,* std::string *val*)

4.17.2.2 void Header::addPair (std::string *key,* std::string *val,* int *t*)

4.17.2.3 void Header::addSubKey (std::string *key*)

4.17.2.4 std::map< std::string, SubHeader >::const_iterator Header::begin ( ) const

4.17.2.5 std::map< std::string, SubHeader >::iterator Header::begin ( )

4.17.2.6 void Header::changeKey (std::string *oldKey,* std::string *newKey*)

4.17.2.7 void Header::clear ( )

4.17.2.8 void Header::copyAnchor2Alias (std::string *alias,* SubHeader & *ref*)

4.17.2.9 void Header::DisplayContents ( )

4.17.2.10 std::map< std::string, **SubHeader** >::const_iterator Header::end ( ) const

4.17.2.11 std::map< std::string, **SubHeader** >::iterator Header::end ( )

4.17.2.12 std::string Header::getAlias ( )

4.17.2.13 **SubHeader** & Header::getAnchoredSub (std::string *alias*)

4.17.2.14 **KeyValueMap** & Header::getDataMap ( )

4.17.2.15 std::string Header::getName ( )

4.17.2.16 int Header::getState ( )

4.17.2.17 **SubHeader** & Header::getSubHeader (std::string *key*)

4.17.2.18 std::map< std::string, **SubHeader** > & Header::getSubMap ( )

4.17.2.19 bool Header::isAlias ( )

4.17.2.20 bool Header::isAnchor ( )

4.17.2.21 **SubHeader** & Header::operator() (const std::string *key*)

4.17.2.22 **SubHeader** Header::operator() (const std::string *key*) const

4.17.2.23 **Header** & Header::operator= (const **Header** & *head*)

4.17.2.24 **ValueTypePair** & Header::operator[] (const std::string *key*)

4.17.2.25 **ValueTypePair** Header::operator[] (const std::string *key*) const

4.17.2.26 void Header::resetKeys ( )

4.17.2.27 void Header::setAlias (std::string *alias*)

4.17.2.28 void Header::setName (std::string *name*)

4.17.2.29 void Header::setNameAliasPair (std::string *n*, std::string *a*, int *s*)

4.17.2.30 void Header::setState (int *state*)

4.17.2.31 int Header::size ( )

4.17.3 Member Data Documentation

4.17.3.1 std::map<std::string, **SubHeader**> Header::Sub_Map `[private]`

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

- yaml_wrapper.h
- yaml_wrapper.cpp

4.18 KeyValueMap Class Reference

```
#include <yaml_wrapper.h>
```

Public Member Functions

- KeyValueMap ()
- ∼KeyValueMap ()
- KeyValueMap (const std::map< std::string, std::string > &map)
- KeyValueMap (std::string key, std::string value)
- KeyValueMap (const KeyValueMap &map)
- KeyValueMap & operator= (const KeyValueMap &map)
- ValueTypePair & operator[] (const std::string key)
- ValueTypePair operator[] (const std::string key) const
- std::map< std::string,
  ValueTypePair > & getMap ()
- std::map< std::string,
  ValueTypePair >
  ::const_iterator end () const
- std::map< std::string,
  ValueTypePair >::iterator end ()
- std::map< std::string,
  ValueTypePair >
  ::const_iterator begin () const
- std::map< std::string,
  ValueTypePair >::iterator begin ()
- void clear ()
- void addKey (std::string key)
- void editValue4Key (std::string val, std::string key)
- void editValue4Key (std::string val, int type, std::string key)
- void addPair (std::string key, ValueTypePair val)
- void addPair (std::string key, std::string val)
- void addPair (std::string key, std::string val, int type)
- void findType (std::string key)
- void assertType (std::string key, int type)
- void findAllTypes ()
- void DisplayMap ()
- int size ()
- std::string getString (std::string key)
- bool getBool (std::string key)
- double getDouble (std::string key)
- int getInt (std::string key)
- std::string getValue (std::string key)
- int getType (std::string key)
- ValueTypePair & getPair (std::string key)

Private Attributes

- std::map< std::string,
  ValueTypePair > Key_Value

4.18.1 Constructor & Destructor Documentation

4.18.1.1 KeyValueMap::KeyValueMap ( )

4.18.1.2 KeyValueMap::∼KeyValueMap ( )

4.18.1.3 KeyValueMap::KeyValueMap (const std::map< std::string, std::string > & *map*)

4.18.1.4 KeyValueMap::KeyValueMap (std::string *key,* std::string *value*)

4.18.1.5 KeyValueMap::KeyValueMap (const KeyValueMap & *map*)

4.18.2 Member Function Documentation

4.18.2.1 void KeyValueMap::addKey (std::string *key*)

4.18.2.2 void KeyValueMap::addPair (std::string *key,* ValueTypePair *val*)

4.18.2.3 void KeyValueMap::addPair (std::string *key,* std::string *val*)

4.18.2.4 void KeyValueMap::addPair (std::string *key,* std::string *val,* int *type*)

4.18.2.5 void KeyValueMap::assertType (std::string *key,* int *type*)

4.18.2.6 std::map< std::string, ValueTypePair >::const_iterator KeyValueMap::begin ( ) const

4.18.2.7 std::map< std::string, ValueTypePair >::iterator KeyValueMap::begin ( )

4.18.2.8 void KeyValueMap::clear ( )

4.18.2.9 void KeyValueMap::DisplayMap ( )

4.18.2.10 void KeyValueMap::editValue4Key (std::string *val,* std::string *key*)

4.18.2.11 void KeyValueMap::editValue4Key (std::string *val,* int *type,* std::string *key*)

4.18.2.12 std::map< std::string, ValueTypePair >::const_iterator KeyValueMap::end ( ) const

4.18.2.13 std::map< std::string, ValueTypePair >::iterator KeyValueMap::end ( )

4.18.2.14 void KeyValueMap::findAllTypes ( )

4.18.2.15 void KeyValueMap::findType (std::string *key*)

4.18.2.16 bool KeyValueMap::getBool (std::string *key*)

4.18.2.17 double KeyValueMap::getDouble (std::string *key*)

4.18.2.18 int KeyValueMap::getInt (std::string *key*)

4.18.2.19 std::map< std::string, ValueTypePair > & KeyValueMap::getMap ( )

4.18.2.20 ValueTypePair & KeyValueMap::getPair (std::string *key*)

4.18.2.21 std::string KeyValueMap::getString (std::string *key*)

4.18.2.22 int KeyValueMap::getType (std::string *key*)

4.18.2.23 std::string KeyValueMap::getValue (std::string *key*)

4.18.2.24 KeyValueMap & KeyValueMap::operator= (const KeyValueMap & *map*)

4.18.2.25 ValueTypePair & KeyValueMap::operator[] (const std::string *key*)

4.18.2.26 ValueTypePair KeyValueMap::operator[] (const std::string *key*) const

4.18.2.27 int KeyValueMap::size ( )

4.18.3 Member Data Documentation

4.18.3.1 std::map<std::string, ValueTypePair > KeyValueMap::Key_Value `[private]`

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

- yaml_wrapper.h
- yaml_wrapper.cpp

4.19 MAGPIE_DATA Struct Reference

```
#include <magpie.h>
```

Public Attributes

- std::vector< GSTA_DATA > gsta_dat
- std::vector< mSPD_DATA > mspd_dat
- std::vector< GPAST_DATA > gpast_dat
- SYSTEM_DATA sys_dat

4.19.1 Member Data Documentation

4.19.1.1 std::vector<GPAST_DATA> MAGPIE_DATA::gpast_dat

4.19.1.2 std::vector<GSTA_DATA> MAGPIE_DATA::gsta_dat

4.19.1.3 std::vector<mSPD_DATA> MAGPIE_DATA::mspd_dat

4.19.1.4 SYSTEM_DATA MAGPIE_DATA::sys_dat

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

- magpie.h

4.20 MassBalance Class Reference

```
#include <shark.h>
```

Public Member Functions

- MassBalance ()
- ∼MassBalance ()
- void Initialize_List (MasterSpeciesList &List)
- void Display_Info ()
- void Set_Delta (int i, double v)
- void Set_TotalConcentration (double v)
- void Set_Name (std::string name)
- double Get_Delta (int i)
- double Sum_Delta ()
- double Get_TotalConcentration ()
- std::string Get_Name ()
- double Eval_Residual (const Matrix< double > &x)

Protected Attributes

- MasterSpeciesList ∗ List
- std::vector< double > Delta
- double TotalConcentration

Private Attributes

- std::string Name

4.20.1 Constructor & Destructor Documentation

4.20.1.1 MassBalance::MassBalance ( )

4.20.1.2 MassBalance::∼MassBalance ( )

4.20.2 Member Function Documentation

4.20.2.1 void MassBalance::Display_Info ( )

4.20.2.2 double MassBalance::Eval_Residual (const **Matrix**< double > & *x*)

4.20.2.3 double MassBalance::Get_Delta (int *i*)

4.20.2.4 std::string MassBalance::Get_Name ( )

4.20.2.5 double MassBalance::Get_TotalConcentration ( )

4.20.2.6 void MassBalance::Initialize_List (**MasterSpeciesList** & *List*)

4.20.2.7 void MassBalance::Set_Delta (int *i*, double *v*)

4.20.2.8 void MassBalance::Set_Name (std::string *name*)

4.20.2.9 void MassBalance::Set_TotalConcentration (double *v*)

4.20.2.10 double MassBalance::Sum_Delta ( )

4.20.3 Member Data Documentation

4.20.3.1 std::vector< double > MassBalance::Delta `[protected]`

4.20.3.2 MasterSpeciesList* MassBalance::List `[protected]`

4.20.3.3 std::string MassBalance::Name `[private]`

4.20.3.4 double MassBalance::TotalConcentration `[protected]`

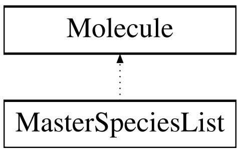
The documentation for this class was generated from the following files:

- shark.h
- shark.cpp

4.21 MasterSpeciesList Class Reference

`#include <shark.h>`

Inheritance diagram for MasterSpeciesList:



Public Member Functions

- MasterSpeciesList ()
- ∼MasterSpeciesList ()
- MasterSpeciesList (const MasterSpeciesList &msl)
- MasterSpeciesList & operator= (const MasterSpeciesList &msl)
- void set_list_size (int i)
- void set_species (int i, std::string formula)
- void set_species (int i, int charge, double enthalpy, double entropy, double energy, bool HS, bool G, std::string Phase, std::string Name, std::string Formula, std::string lin_formula)
- void DisplayInfo (int i)
- void DisplayAll ()
- void DisplayConcentrations (Matrix< double > &C)
- void set_alkalinity (double alk)
- int list_size ()
- Molecule & get_species (int i)
- int get_index (std::string name)
- double charge (int i)
- double alkalinity ()
- std::string speciesName (int i)
- double Eval_ChargeResidual (const Matrix< double > &x)

Protected Attributes

- int size
- std::vector< Molecule > species
- double residual_alkalinity

Additional Inherited Members

4.21.1 Constructor & Destructor Documentation

4.21.1.1 MasterSpeciesList::MasterSpeciesList ( )

4.21.1.2 MasterSpeciesList::∼MasterSpeciesList ( )

4.21.1.3 MasterSpeciesList::MasterSpeciesList (const **MasterSpeciesList** & *msl*)

4.21.2 Member Function Documentation

4.21.2.1 double MasterSpeciesList::alkalinity ( )

4.21.2.2 double MasterSpeciesList::charge (int *i*)

4.21.2.3 void MasterSpeciesList::DisplayAll ( )

4.21.2.4 void MasterSpeciesList::DisplayConcentrations (**Matrix**< double > & *C*)

4.21.2.5 void MasterSpeciesList::DisplayInfo (int *i*)

4.21.2.6 double MasterSpeciesList::Eval_ChargeResidual (const **Matrix**< double > & *x*)

4.21.2.7 int MasterSpeciesList::get_index (std::string *name*)

4.21.2.8 **Molecule** & MasterSpeciesList::get_species (int *i*)

4.21.2.9 int MasterSpeciesList::list_size ( )

4.21.2.10 **MasterSpeciesList** & MasterSpeciesList::operator= (const **MasterSpeciesList** & *msl*)

4.21.2.11 void MasterSpeciesList::set_alkalinity (double *alk*)

4.21.2.12 void MasterSpeciesList::set_list_size (int *i*)

4.21.2.13 void MasterSpeciesList::set_species (int *i*, std::string *formula*)

4.21.2.14 void MasterSpeciesList::set_species (int *i*, int *charge*, double *enthalpy*, double *entropy*, double *energy*, bool *HS*, bool *G*, std::string *Phase*, std::string *Name*, std::string *Formula*, std::string *lin_formula*)

4.21.2.15 std::string MasterSpeciesList::speciesName (int *i*)

4.21.3 Member Data Documentation

4.21.3.1 double MasterSpeciesList::residual_alkalinity `[protected]`

4.21.3.2 int MasterSpeciesList::size `[protected]`

4.21.3.3 std::vector<**Molecule**> MasterSpeciesList::species `[protected]`

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

- shark.h
- shark.cpp

4.22 Matrix< T > Class Template Reference

```
#include <macaw.h>
```

Public Member Functions

- Matrix (int rows, int columns)
- T & operator() (int i, int j)
- T operator() (int i, int j) const
- Matrix (const Matrix &M)
- Matrix & operator= (const Matrix &M)
- Matrix ()
- ∼Matrix ()
- void set_size (int i, int j)
- void zeros ()
- void edit (int i, int j, T value)
- int rows ()
- int columns ()
- T determinate ()
- T norm ()
- T sum ()
- T inner_product (const Matrix &x)
- Matrix & cofactor (const Matrix &M)
- Matrix operator+ (const Matrix &M)
- Matrix operator- (const Matrix &M)
- Matrix operator∗ (const T)
- Matrix operator/ (const T)
- Matrix operator∗ (const Matrix &M)
- Matrix & transpose (const Matrix &M)
- Matrix & transpose_multiply (const Matrix &MT, const Matrix &v)
- Matrix & adjoint (const Matrix &M)
- Matrix & inverse (const Matrix &M)
- void Display (const std::string Name)
- Matrix & tridiagonalSolve (const Matrix &A, const Matrix &b)
- Matrix & ladshawSolve (const Matrix &A, const Matrix &d)
- Matrix & tridiagonalFill (const T A, const T B, const T C, bool Spherical)
- Matrix & naturalLaplacian3D (int m)
- Matrix & sphericalBCFill (int node, const T coeff, T variable)
- Matrix & ConstantICFill (const T IC)
- Matrix & SolnTransform (const Matrix &A, bool Forward)
- T sphericalAvg (double radius, double dr, double bound, bool Dirichlet)
- T IntegralAvg (double radius, double dr, double bound, bool Dirichlet)
- T IntegralTotal (double dr, double bound, bool Dirichlet)
- Matrix & tridiagonalVectorFill (const std::vector< T > &A, const std::vector< T > &B, const std::vector< T > &C)
- Matrix & columnVectorFill (const std::vector< T > &A)
- Matrix & columnProjection (const Matrix &b, const Matrix &b_old, const double dt, const double dt_old)
- Matrix & dirichletBCFill (int node, const T coeff, T variable)
- Matrix & diagonalSolve (const Matrix &D, const Matrix &v)
- Matrix & upperTriangularSolve (const Matrix &U, const Matrix &v)
- Matrix & lowerTriangularSolve (const Matrix &L, const Matrix &v)
- Matrix & upperHessenberg2Triangular (Matrix &b)
- Matrix & lowerHessenberg2Triangular (Matrix &b)
- Matrix & upperHessenbergSolve (const Matrix &H, const Matrix &v)

- Matrix & lowerHessenbergSolve (const Matrix &H, const Matrix &v)
- Matrix & columnExtract (int j, const Matrix &M)
- Matrix & rowExtract (int i, const Matrix &M)
- Matrix & columnReplace (int j, const Matrix &v)
- Matrix & rowReplace (int i, const Matrix &v)
- void rowShrink ()
- void columnShrink ()
- void rowExtend (const Matrix &v)
- void columnExtend (const Matrix &v)

Protected Attributes

- int num_rows
- int num_cols
- std::vector< T > Data

4.22.1 Constructor & Destructor Documentation

4.22.1.1 template<class T > Matrix< T >::Matrix (int *rows,* int *columns*)

4.22.1.2 template<class T > Matrix< T >::Matrix (const Matrix< T > & *M*)

4.22.1.3 template<class T > Matrix< T >::Matrix ( )

4.22.1.4 template<class T > Matrix< T >::∼Matrix ( )

4.22.2 Member Function Documentation

4.22.2.1 template<class T > Matrix< T > & Matrix< T >::adjoint (const Matrix< T > & *M*)

4.22.2.2 template<class T > Matrix< T > & Matrix< T >::cofactor (const Matrix< T > & *M*)

4.22.2.3 template<class T > void Matrix< T >::columnExtend (const Matrix< T > & *v*)

4.22.2.4 template<class T > Matrix< T > & Matrix< T >::columnExtract (int *j,* const Matrix< T > & *M*)

4.22.2.5 template<class T > Matrix< T > & Matrix< T >::columnProjection (const Matrix< T > & *b,* const Matrix< T > & *b_old,* const double *dt,* const double *dt_old*)

4.22.2.6 template<class T > Matrix< T > & Matrix< T >::columnReplace (int *j,* const Matrix< T > & *v*)

4.22.2.7 template<class T > int Matrix< T >::columns ( )

4.22.2.8 template<class T > void Matrix< T >::columnShrink ( )

4.22.2.9 template<class T> Matrix< T > & Matrix< T >::columnVectorFill (const std::vector< T > & *A*)

4.22.2.10 template<class T> Matrix< T > & Matrix< T >::ConstantICFill (const T *IC*)

4.22.2.11 template<class T > T Matrix< T >::determinate ( )

4.22.2.12 template<class T > Matrix< T > & Matrix< T >::diagonalSolve (const Matrix< T > & *D,* const Matrix< T > & *v*)

4.22.2.13 template<class T> Matrix< T > & Matrix< T >::dirichletBCFill (int *node*, const T *coeff*, T *variable*)

4.22.2.14 template<class T > void Matrix< T >::Display (const std::string *Name*)

4.22.2.15 template<class T> void Matrix< T >::edit (int *i*, int *j*, T *value*)

4.22.2.16 template<class T > T Matrix< T >::inner_product (const Matrix< T > & *x*)

4.22.2.17 template<class T > T Matrix< T >::IntegralAvg (double *radius*, double *dr*, double *bound*, bool *Dirichlet*)

4.22.2.18 template<class T > T Matrix< T >::IntegralTotal (double *dr*, double *bound*, bool *Dirichlet*)

4.22.2.19 template<class T > Matrix< T > & Matrix< T >::inverse (const Matrix< T > & *M*)

4.22.2.20 template<class T > Matrix< T > & Matrix< T >::ladshawSolve (const Matrix< T > & *A*, const Matrix< T > & *d*)

4.22.2.21 template<class T > Matrix< T > & Matrix< T >::lowerHessenberg2Triangular (Matrix< T > & *b*)

4.22.2.22 template<class T > Matrix< T > & Matrix< T >::lowerHessenbergSolve (const Matrix< T > & *H*, const Matrix< T > & *v*)

4.22.2.23 template<class T > Matrix< T > & Matrix< T >::lowerTriangularSolve (const Matrix< T > & *L*, const Matrix< T > & *v*)

4.22.2.24 template<class T > Matrix< T > & Matrix< T >::naturalLaplacian3D (int *m*)

4.22.2.25 template<class T > T Matrix< T >::norm ( )

4.22.2.26 template<class T > T & Matrix< T >::operator() (int *i*, int *j*)

4.22.2.27 template<class T > T Matrix< T >::operator() (int *i*, int *j*) const

4.22.2.28 template<class T> Matrix< T > Matrix< T >::operator* (const T *a*)

4.22.2.29 template<class T> Matrix< T > Matrix< T >::operator* (const Matrix< T > & *M*)

4.22.2.30 template<class T > Matrix< T > Matrix< T >::operator+ (const Matrix< T > & *M*)

4.22.2.31 template<class T > Matrix< T > Matrix< T >::operator- (const Matrix< T > & *M*)

4.22.2.32 template<class T> Matrix< T > Matrix< T >::operator/ (const T *a*)

4.22.2.33 template<class T > Matrix< T > & Matrix< T >::operator= (const Matrix< T > & *M*)

4.22.2.34 template<class T > void Matrix< T >::rowExtend (const Matrix< T > & *v*)

4.22.2.35 template<class T > Matrix< T > & Matrix< T >::rowExtract (int *i*, const Matrix< T > & *M*)

4.22.2.36 template<class T > Matrix< T > & Matrix< T >::rowReplace (int *i*, const Matrix< T > & *v*)

4.22.2.37 template<class T > int Matrix< T >::rows ( )

4.22.2.38 template<class T > void Matrix< T >::rowShrink ( )

4.22.2.39 template<class T > void Matrix< T >::set_size (int *i*, int *j*)

4.22.2.40 template< class T > Matrix< T > & Matrix< T >::SolnTransform (const Matrix< T > & *A*, bool *Forward*)

4.22.2.41 template< class T > T Matrix< T >::sphericalAvg (double *radius*, double *dr*, double *bound*, bool *Dirichlet*)

4.22.2.42 template< class T> Matrix< T > & Matrix< T >::sphericalBCFill (int *node*, const T *coeff*, T *variable*)

4.22.2.43 template< class T > T Matrix< T >::sum ( )

4.22.2.44 template< class T > Matrix< T > & Matrix< T >::transpose (const Matrix< T > & *M*)

4.22.2.45 template< class T > Matrix< T > & Matrix< T >::transpose_multiply (const Matrix< T > & *MT*, const Matrix< T > & *v*)

4.22.2.46 template< class T> Matrix< T > & Matrix< T >::tridiagonalFill (const T *A*, const T *B*, const T *C*, bool *Spherical*)

4.22.2.47 template< class T > Matrix< T > & Matrix< T >::tridiagonalSolve (const Matrix< T > & *A*, const Matrix< T > & *b*)

4.22.2.48 template< class T> Matrix< T > & Matrix< T >::tridiagonalVectorFill (const std::vector< T > & *A*, const std::vector< T > & *B*, const std::vector< T > & *C*)

4.22.2.49 template< class T > Matrix< T > & Matrix< T >::upperHessenberg2Triangular (Matrix< T > & *b*)

4.22.2.50 template< class T > Matrix< T > & Matrix< T >::upperHessenbergSolve (const Matrix< T > & *H*, const Matrix< T > & *v*)

4.22.2.51 template< class T > Matrix< T > & Matrix< T >::upperTriangularSolve (const Matrix< T > & *U*, const Matrix< T > & *v*)

4.22.2.52 template< class T > void Matrix< T >::zeros ( )

4.22.3 Member Data Documentation

4.22.3.1 template< class T> std::vector<T> Matrix< T >::Data `[protected]`

4.22.3.2 template< class T> int Matrix< T >::num_cols `[protected]`

4.22.3.3 template< class T> int Matrix< T >::num_rows `[protected]`

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

- macaw.h

4.23 Mechanism Class Reference

`#include <shark.h>`

Protected Attributes

- MasterSpeciesList ∗ List
- std::vector< UnsteadyReaction > reactions
- std::vector< double > weight
- int species_index

4.23.1 Member Data Documentation

4.23.1.1 MasterSpeciesList∗ Mechanism::List [protected]

4.23.1.2 std::vector<UnsteadyReaction> Mechanism::reactions [protected]

4.23.1.3 int Mechanism::species_index [protected]

4.23.1.4 std::vector<double> Mechanism::weight [protected]

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

- shark.h

4.24 MIXED_GAS Struct Reference

```
#include <egret.h>
```

Public Attributes

- int N
- bool CheckMolefractions = true
- double total_pressure
- double gas_temperature
- double velocity
- double char_length
- std::vector< double > molefraction
- double total_density
- double total_dyn_vis
- double kinematic_viscosity
- double total_molecular_weight
- double total_specific_heat
- double Reynolds
- Matrix< double > binary_diffusion
- std::vector< PURE_GAS > species_dat

4.24.1 Member Data Documentation

4.24.1.1 Matrix<double> MIXED_GAS::binary_diffusion

4.24.1.2 double MIXED_GAS::char_length

4.24.1.3 bool MIXED_GAS::CheckMolefractions = true

4.24.1.4 double MIXED_GAS::gas_temperature

4.24.1.5 double MIXED_GAS::kinematic_viscosity

4.24.1.6 std::vector<double> MIXED_GAS::molefraction

4.24.1.7 int MIXED_GAS::N

4.24.1.8 double MIXED_GAS::Reynolds

4.24.1.9 std::vector<PURE_GAS> MIXED_GAS::species_dat

4.24.1.10 double MIXED_GAS::total_density

4.24.1.11 double MIXED_GAS::total_dyn_vis

4.24.1.12 double MIXED_GAS::total_molecular_weight

4.24.1.13 double MIXED_GAS::total_pressure

4.24.1.14 double MIXED_GAS::total_specific_heat

4.24.1.15 double MIXED_GAS::velocity

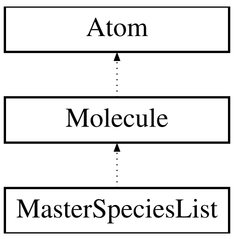
The documentation for this struct was generated from the following file:

- egret.h

4.25 Molecule Class Reference

```
#include <mola.h>
```

Inheritance diagram for Molecule:



Public Member Functions

- Molecule ()
- ∼Molecule ()
- Molecule (int charge, double enthalpy, double entropy, double energy, bool HS, bool G, std::string Phase, std::string Name, std::string Formula, std::string lin_formula)
- void Register (int charge, double enthalpy, double entropy, double energy, bool HS, bool G, std::string Phase, std::string Name, std::string Formula, std::string lin_formula)
- void Register (std::string formula)
- void setFormula (std::string form)
- void recalculateMolarWeight ()
- void setMolarWeigth (double mw)
- void editCharge (int c)
- void editOneOxidationState (int state, std::string Symbol)
- void editAllOxidationStates (int state, std::string Symbol)
- void calculateAvgOxiState (std::string Symbol)
- void editEnthalpy (double enthalpy)
- void editEntropy (double entropy)
- void editHS (double H, double S)

- void editEnergy (double energy)
- void removeOneAtom (std::string Symbol)
- void removeAllAtoms (std::string Symbol)
- int Charge ()
- double MolarWeight ()
- bool HaveHS ()
- bool HaveEnergy ()
- bool isRegistered ()
- double Enthalpy ()
- double Entropy ()
- double Energy ()
- std::string MoleculeName ()
- std::string MolecularFormula ()
- std::string MoleculePhase ()
- void DisplayInfo ()

Protected Attributes

- int charge
- double molar_weight
- double formation_enthalpy
- double formation_entropy
- double formation_energy
- std::string Phase
- std::vector$<$ Atom $>$ atoms

Private Attributes

- std::string Name
- std::string Formula
- bool haveG
- bool haveHS
- bool registered

Additional Inherited Members

4.25.1 Constructor & Destructor Documentation

4.25.1.1 Molecule::Molecule ( )

4.25.1.2 Molecule::∼Molecule ( )

4.25.1.3 Molecule::Molecule (int *charge,* double *enthalpy,* double *entropy,* double *energy,* bool *HS,* bool *G,* std::string *Phase,* std::string *Name,* std::string *Formula,* std::string *lin_formula*)

4.25.2 Member Function Documentation

4.25.2.1 void Molecule::calculateAvgOxiState (std::string *Symbol*)

4.25.2.2 int Molecule::Charge ( )

4.25.2.3 void Molecule::DisplayInfo ( )

4.25.2.4 void Molecule::editAllOxidationStates (int *state,* std::string *Symbol*)

4.25.2.5 void Molecule::editCharge (int *c*)

4.25.2.6 void Molecule::editEnergy (double *energy*)

4.25.2.7 void Molecule::editEnthalpy (double *enthalpy*)

4.25.2.8 void Molecule::editEntropy (double *entropy*)

4.25.2.9 void Molecule::editHS (double *H,* double *S*)

4.25.2.10 void Molecule::editOneOxidationState (int *state,* std::string *Symbol*)

4.25.2.11 double Molecule::Energy ( )

4.25.2.12 double Molecule::Enthalpy ( )

4.25.2.13 double Molecule::Entropy ( )

4.25.2.14 bool Molecule::HaveEnergy ( )

4.25.2.15 bool Molecule::HaveHS ( )

4.25.2.16 bool Molecule::isRegistered ( )

4.25.2.17 double Molecule::MolarWeight ( )

4.25.2.18 std::string Molecule::MolecularFormula ( )

4.25.2.19 std::string Molecule::MoleculeName ( )

4.25.2.20 std::string Molecule::MoleculePhase ( )

4.25.2.21 void Molecule::recalculateMolarWeight ( )

4.25.2.22 void Molecule::Register (int *charge,* double *enthalpy,* double *entropy,* double *energy,* bool *HS,* bool *G,* std::string *Phase,* std::string *Name,* std::string *Formula,* std::string *lin_formula*)

4.25.2.23 void Molecule::Register (std::string *formula*)

4.25.2.24 void Molecule::removeAllAtoms (std::string *Symbol*)

4.25.2.25 void Molecule::removeOneAtom (std::string *Symbol*)

4.25.2.26 void Molecule::setFormula (std::string *form*)

4.25.2.27 void Molecule::setMolarWeigth (double *mw*)

4.25.3 Member Data Documentation

4.25.3.1 std::vector<Atom> Molecule::atoms `[protected]`

4.25.3.2 int Molecule::charge `[protected]`

4.25.3.3 double Molecule::formation_energy `[protected]`

4.25.3.4 double Molecule::formation_enthalpy [protected]

4.25.3.5 double Molecule::formation_entropy [protected]

4.25.3.6 std::string Molecule::Formula [private]

4.25.3.7 bool Molecule::haveG [private]

4.25.3.8 bool Molecule::haveHS [private]

4.25.3.9 double Molecule::molar_weight [protected]

4.25.3.10 std::string Molecule::Name [private]

4.25.3.11 std::string Molecule::Phase [protected]

4.25.3.12 bool Molecule::registered [private]

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

- mola.h
- mola.cpp

4.26 MONKFISH_DATA Struct Reference

```
#include <monkfish.h>
```

Public Attributes

- unsigned long int total_steps = 0
- double time_old = 0.0
- double time = 0.0
- bool Print2File = true
- bool Print2Console = true
- bool DirichletBC = true
- bool NonLinear = false
- bool haveMinMax = false
- bool MultiScale = true
- int level = 2
- double t_counter = 0.0
- double t_print
- int NumComp
- double end_time
- double total_sorption_old
- double total_sorption
- double single_fiber_density
- double avg_fiber_density
- double max_fiber_density
- double min_fiber_density
- double max_porosity
- double min_porosity
- double domain_diameter
- FILE ∗ Output

- double(∗ eval_eps)(int i, int l, const void ∗user_data)
- double(∗ eval_rho)(int i, int l, const void ∗user_data)
- double(∗ eval_Dex)(int i, int l, const void ∗user_data)
- double(∗ eval_ads)(int i, int l, const void ∗user_data)
- double(∗ eval_Ret)(int i, int l, const void ∗user_data)
- double(∗ eval_Cex)(int i, const void ∗user_data)
- double(∗ eval_kf)(int i, const void ∗user_data)
- const void ∗ user_data
- std::vector< FINCH_DATA > finch_dat
- std::vector< MONKFISH_PARAM > param_dat
- std::vector< DOGFISH_DATA > dog_dat

4.26.1 Member Data Documentation

4.26.1.1 double MONKFISH_DATA::avg_fiber_density

4.26.1.2 bool MONKFISH_DATA::DirichletBC = true

4.26.1.3 std::vector<DOGFISH_DATA> MONKFISH_DATA::dog_dat

4.26.1.4 double MONKFISH_DATA::domain_diameter

4.26.1.5 double MONKFISH_DATA::end_time

4.26.1.6 double(∗ MONKFISH_DATA::eval_ads)(int i, int l, const void ∗user_data)

4.26.1.7 double(∗ MONKFISH_DATA::eval_Cex)(int i, const void ∗user_data)

4.26.1.8 double(∗ MONKFISH_DATA::eval_Dex)(int i, int l, const void ∗user_data)

4.26.1.9 double(∗ MONKFISH_DATA::eval_eps)(int i, int l, const void ∗user_data)

4.26.1.10 double(∗ MONKFISH_DATA::eval_kf)(int i, const void ∗user_data)

4.26.1.11 double(∗ MONKFISH_DATA::eval_Ret)(int i, int l, const void ∗user_data)

4.26.1.12 double(∗ MONKFISH_DATA::eval_rho)(int i, int l, const void ∗user_data)

4.26.1.13 std::vector<FINCH_DATA> MONKFISH_DATA::finch_dat

4.26.1.14 bool MONKFISH_DATA::haveMinMax = false

4.26.1.15 int MONKFISH_DATA::level = 2

4.26.1.16 double MONKFISH_DATA::max_fiber_density

4.26.1.17 double MONKFISH_DATA::max_porosity

4.26.1.18 double MONKFISH_DATA::min_fiber_density

4.26.1.19 double MONKFISH_DATA::min_porosity

4.26.1.20 bool MONKFISH_DATA::MultiScale = true

4.26.1.21 bool MONKFISH_DATA::NonLinear = false

4.26.1.22 int MONKFISH_DATA::NumComp

4.26.1.23 FILE∗ MONKFISH_DATA::Output

4.26.1.24 std::vector<MONKFISH_PARAM> MONKFISH_DATA::param_dat

4.26.1.25 bool MONKFISH_DATA::Print2Console = true

4.26.1.26 bool MONKFISH_DATA::Print2File = true

4.26.1.27 double MONKFISH_DATA::single_fiber_density

4.26.1.28 double MONKFISH_DATA::t_counter = 0.0

4.26.1.29 double MONKFISH_DATA::t_print

4.26.1.30 double MONKFISH_DATA::time = 0.0

4.26.1.31 double MONKFISH_DATA::time_old = 0.0

4.26.1.32 double MONKFISH_DATA::total_sorption

4.26.1.33 double MONKFISH_DATA::total_sorption_old

4.26.1.34 unsigned long int MONKFISH_DATA::total_steps = 0

4.26.1.35 const void∗ MONKFISH_DATA::user_data

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

- monkfish.h

4.27 MONKFISH_PARAM Struct Reference

```
#include <monkfish.h>
```

Public Attributes

- double interparticle_diffusion
- double exterior_concentration
- double exterior_transfer_coeff
- double sorbed_molefraction
- double initial_sorption
- double sorption_bc
- double intraparticle_diffusion
- double film_transfer_coeff
- Matrix< double > avg_sorption
- Matrix< double > avg_sorption_old
- Molecule species

4.27.1 Member Data Documentation

4.27.1.1 Matrix<double> MONKFISH_PARAM::avg_sorption

4.27.1.2 Matrix<double> MONKFISH_PARAM::avg_sorption_old

4.27.1.3 double MONKFISH_PARAM::exterior_concentration

4.27.1.4 double MONKFISH_PARAM::exterior_transfer_coeff

4.27.1.5 double MONKFISH_PARAM::film_transfer_coeff

4.27.1.6 double MONKFISH_PARAM::initial_sorption

4.27.1.7 double MONKFISH_PARAM::interparticle_diffusion

4.27.1.8 double MONKFISH_PARAM::intraparticle_diffusion

4.27.1.9 double MONKFISH_PARAM::sorbed_molefraction

4.27.1.10 double MONKFISH_PARAM::sorption_bc

4.27.1.11 Molecule MONKFISH_PARAM::species

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

- monkfish.h

4.28 mSPD_DATA Struct Reference

```
#include <magpie.h>
```

Public Attributes

- double s
- double v
- double eMax
- std::vector< double > eta
- double gama

4.28.1 Member Data Documentation

4.28.1.1 double mSPD_DATA::eMax

4.28.1.2 std::vector<double> mSPD_DATA::eta

4.28.1.3 double mSPD_DATA::gama

4.28.1.4 double mSPD_DATA::s

4.28.1.5 double mSPD_DATA::v

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

- magpie.h

4.29 NUM_JAC_DATA Struct Reference

Data structure to form a numerical jacobian matrix with finite differences.

```
#include <lark.h>
```

Public Attributes

- double eps = sqrt(DBL_EPSILON)

  Perturbation value.
- Matrix< double > Fx

  Vector of function evaluations at x.
- Matrix< double > Fxp

  Vector of function evaluations at x+eps.
- Matrix< double > dxj

  Vector of perturbed x values.

4.29.1 Detailed Description

Data structure to form a numerical jacobian matrix with finite differences.

C-style object to be used in conjunction with the Numerical Jacobian algorithm. This algorithm will used double-precision finite-differences to formulate an approximate Jacobian matrix at the given variable state for the given residual/non-linear function.

4.29.2 Member Data Documentation

4.29.2.1 Matrix<double> NUM_JAC_DATA::dxj

Vector of perturbed x values.

4.29.2.2 double NUM_JAC_DATA::eps = sqrt(DBL_EPSILON)

Perturbation value.

4.29.2.3 Matrix<double> NUM_JAC_DATA::Fx

Vector of function evaluations at x.

4.29.2.4 Matrix<double> NUM_JAC_DATA::Fxp

Vector of function evaluations at x+eps.

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

- lark.h

4.30 OPTRANS_DATA Struct Reference

Data structure for implementation of linear operator transposition.

`#include <lark.h>`

Public Attributes

- Matrix< double > Ii

    The ith column vector of the identity operator.
- Matrix< double > Ai

    The ith column vector of the user's linear operator.

4.30.1 Detailed Description

Data structure for implementation of linear operator transposition.

C-style object used in conjunction with the Operator Transpose algorithm to form an action of A^T∗r when A is only available as a linear operator and not a matrix. This is a sub-routine required by GCR and GMRESR to stabilize the outer iterations.

4.30.2 Member Data Documentation

4.30.2.1 Matrix<double> OPTRANS_DATA::Ai

The ith column vector of the user's linear operator.

4.30.2.2 Matrix<double> OPTRANS_DATA::Ii

The ith column vector of the identity operator.

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

- lark.h

4.31 PCG_DATA Struct Reference

Data structure for implementation of the PCG algorithms for symmetric linear systems.

`#include <lark.h>`

Public Attributes

- int maxit = 0

    Maximum allowable iterations - default = min(vector_size,1000)
- int iter = 0

    Actual number of iterations taken.
- double alpha

    Step size for new solution.
- double beta

    Step size for new search direction.
- double tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.

- double res

  Absolute residual norm.

- double relres

  Relative residual norm.

- double relres_base

  Initial residual norm.

- double bestres

  Best found residual norm.

- bool Output = true

  True = print messages to console.

- Matrix< double > x

  Current solution to the linear system.

- Matrix< double > bestx

  Best found solution to the linear system.

- Matrix< double > r

  Residual vector for the linear system.

- Matrix< double > r_old

  Previous residual vector.

- Matrix< double > z

  Preconditioned residual vector (result of precon function)

- Matrix< double > z_old

  Previous preconditioned residual vector.

- Matrix< double > p

  Search direction.

- Matrix< double > Ap

  Result of matrix-vector multiplication.

4.31.1 Detailed Description

Data structure for implementation of the PCG algorithms for symmetric linear systems.

C-style object used in conjunction with the Preconditioned Conjugate Gradient (PCG) algorithm to iteratively solve a symmetric linear system of equations. This algorithm is optimal if your linear system is symmetric, but will not work at all if your system is asymmetric. For asymmetric systems, use one of the other linear methods.

4.31.2 Member Data Documentation

4.31.2.1 double PCG_DATA::alpha

Step size for new solution.

4.31.2.2 Matrix<double> PCG_DATA::Ap

Result of matrix-vector multiplication.

4.31.2.3 double PCG_DATA::bestres

Best found residual norm.

4.31.2.4 Matrix<double> PCG_DATA::bestx

Best found solution to the linear system.

4.31.2.5 double PCG_DATA::beta

Step size for new search direction.

4.31.2.6 int PCG_DATA::iter = 0

Actual number of iterations taken.

4.31.2.7 int PCG_DATA::maxit = 0

Maximum allowable iterations - default = min(vector_size,1000)

4.31.2.8 bool PCG_DATA::Output = true

True = print messages to console.

4.31.2.9 Matrix<double> PCG_DATA::p

Search direction.

4.31.2.10 Matrix<double> PCG_DATA::r

Residual vector for the linear system.

4.31.2.11 Matrix<double> PCG_DATA::r_old

Previous residual vector.

4.31.2.12 double PCG_DATA::relres

Relative residual norm.

4.31.2.13 double PCG_DATA::relres_base

Initial residual norm.

4.31.2.14 double PCG_DATA::res

Absolute residual norm.

4.31.2.15 double PCG_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

4.31.2.16 double PCG_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.31.2.17 Matrix<double> PCG_DATA::x

Current solution to the linear system.

4.31.2.18 Matrix<double> PCG_DATA::z

Preconditioned residual vector (result of precon function)

4.31.2.19 Matrix<double> PCG_DATA::z_old

Previous preconditioned residual vector.

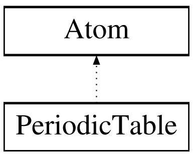
The documentation for this struct was generated from the following file:

- lark.h

4.32 PeriodicTable Class Reference

```
#include <eel.h>
```

Inheritance diagram for PeriodicTable:



Public Member Functions

- PeriodicTable ()
- ∼PeriodicTable ()
- PeriodicTable (int ∗n, int N)
- PeriodicTable (std::vector< std::string > &Symbol)
- PeriodicTable (std::vector< int > &n)
- void DisplayTable ()

Protected Attributes

- std::vector< Atom > Table

Private Attributes

- int number_elements

Additional Inherited Members

4.32.1 Constructor & Destructor Documentation

4.32.1.1 PeriodicTable::PeriodicTable ( )

4.32.1.2 PeriodicTable::∼PeriodicTable ( )

4.32.1.3 PeriodicTable::PeriodicTable (int ∗ *n*, int *N*)

4.32.1.4 PeriodicTable::PeriodicTable (std::vector< std::string > & *Symbol*)

4.32.1.5 PeriodicTable::PeriodicTable (std::vector< int > & *n*)

4.32.2 Member Function Documentation

4.32.2.1 void PeriodicTable::DisplayTable ( )

4.32.3 Member Data Documentation

4.32.3.1 int PeriodicTable::number_elements `[private]`

4.32.3.2 std::vector<Atom> PeriodicTable::Table `[protected]`

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

- eel.h
- eel.cpp

4.33 PICARD_DATA Struct Reference

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

```
#include <lark.h>
```

Public Attributes

- int maxit = 0

  Maximum allowable iterations - default = min(3∗vec_size,1000)
- int iter = 0

  Actual number of iterations.
- double tol_rel = 1e-6

  Relative tolerance for convergence - default = 1e-6.
- double tol_abs = 1e-6

  Absolution tolerance for convergence - default = 1e-6.
- double res

  Residual norm of the iterate.
- double relres

  Relative residual norm of the iterate.
- double relres_base

  Initial residual norm.
- double bestres

  Best found residual norm.

- bool Output = true

    True = print messages to console.
- Matrix< double > x0

    Previous iterate solution vector.
- Matrix< double > bestx

    Best found solution vector.
- Matrix< double > r

    Residual of the non-linear system.

4.33.1 Detailed Description

Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.

C-style object used in conjunction with the Picard algorithm for solving a non-linear system of equations. This is an extradorinarily simple iterative method by which a weak or loose form of the non-linear system is solved based on an initial guess. User must supplied a residual function for the non-linear system and a function representing the weak solution. Generally, this method is less efficient than Newton methods, but is significantly cheaper.

4.33.2 Member Data Documentation

4.33.2.1 double PICARD_DATA::bestres

Best found residual norm.

4.33.2.2 Matrix<double> PICARD_DATA::bestx

Best found solution vector.

4.33.2.3 int PICARD_DATA::iter = 0

Actual number of iterations.

4.33.2.4 int PICARD_DATA::maxit = 0

Maximum allowable iterations - default = min(3∗vec_size,1000)

4.33.2.5 bool PICARD_DATA::Output = true

True = print messages to console.

4.33.2.6 Matrix<double> PICARD_DATA::r

Residual of the non-linear system.

4.33.2.7 double PICARD_DATA::relres

Relative residual norm of the iterate.

4.33.2.8 double PICARD_DATA::relres_base

Initial residual norm.

4.33.2.9 double PICARD_DATA::res

Residual norm of the iterate.

4.33.2.10 double PICARD_DATA::tol_abs = 1e-6

Absolution tolerance for convergence - default = 1e-6.

4.33.2.11 double PICARD_DATA::tol_rel = 1e-6

Relative tolerance for convergence - default = 1e-6.

4.33.2.12 Matrix<double> PICARD_DATA::x0

Previous iterate solution vector.

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

- lark.h

4.34 PJFNK_DATA Struct Reference

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

```
#include <lark.h>
```

Public Attributes

- int nl_iter = 0
  Number of non-linear iterations.
- int l_iter = 0
  Number of linear iterations.
- int nl_maxit = 0
  Maximum allowable non-linear steps.
- int linear_solver = -1
  Flag to denote which linear solver to use - default = PJFNK Chooses.
- double nl_tol_abs = 1e-6
  Absolute Convergence tolerance for non-linear system - default = 1e-6.
- double nl_tol_rel = 1e-6
  Relative Convergence tol for the non-linear system - default = 1e-6.
- double lin_tol_rel = 1e-6
  Relative tolerance of the linear solver - default = 1e-6.
- double lin_tol_abs = 1e-6
  Absolute tolerance of the linear solver - default = 1e-6.
- double nl_res
  Absolute redidual norm for the non-linear system.
- double nl_relres
  Relative residual for the non-linear system.
- double nl_res_base
  Initial residual norm for the non-linear system.
- double nl_bestres

Best found residual norm.

- double eps =sqrt(DBL_EPSILON)

  Value of epsilon used jacvec - default = sqrt(DBL_EPSILON)
- bool NL_Output = true

  True = print PJFNK messages to console.
- bool L_Output = false

  True = print Linear messages to console.
- bool LineSearch = false

  True = use Backtracking Linesearch for global convergence.
- bool Bounce = false

  True = allow Linesearch to go outside local well, False = Strict local convergence.
- Matrix$<$ double $>$ F

  Stored fuction evaluation at x (also the residual)
- Matrix$<$ double $>$ Fv

  Stored function evaluation at x+eps$*$v.
- Matrix$<$ double $>$ v

  Stored vector of x+eps$*$v.
- Matrix$<$ double $>$ x

  Current solution vector for the non-linear system.
- Matrix$<$ double $>$ bestx

  Best found solution vector to the non-linear system.
- GMRESLP_DATA gmreslp_dat

  Data structure for the GMRESLP method.
- PCG_DATA pcg_dat

  Data structure for the PCG method.
- BiCGSTAB_DATA bicgstab_dat

  Data structure for the BiCGSTAB method.
- CGS_DATA cgs_dat

  Data structure for the CGS method.
- GMRESRP_DATA gmresrp_dat

  Data structure for the GMRESRP method.
- GCR_DATA gcr_dat

  Data structure for the GCR method.
- GMRESR_DATA gmresr_dat

  Data structure for the GMRESR method.
- BACKTRACK_DATA backtrack_dat

  Data structure for the Backtracking Linesearch algorithm.
- const void $*$ res_data

  Data structure pointer for user's residual data.
- const void $*$ precon_data

  Data structure pointer for user's preconditioning data.
- int($*$ funeval)(const Matrix$<$ double $>$ &x, Matrix$<$ double $>$ &F, const void $*$res_data)

  Function pointer for the user's function F(x) using there data.
- int($*$ precon)(const Matrix$<$ double $>$ &r, Matrix$<$ double $>$ &p, const void $*$precon_data)

  Function pointer for the user's preconditioning function for the linear system.

4.34.1 Detailed Description

Data structure for the implementation of the PJFNK algorithm for non-linear systems.

C-style object to be used in conjunction with the Preconditioned Jacobian-Free Newton-Krylov (PJFNK) method for solving a non-linear system of equations. You can use any of the Krylov methods listed in the krylov_method enum to solve the linear sub-problem. When FOM is specified as the Krylov method, this algorithm becomes equivalent to an exact Newton method. If no Krylov method is specified, then the algorithm will try to pick a method based on the problem size and availability of preconditioning.

4.34.2 Member Data Documentation

4.34.2.1 BACKTRACK_DATA PJFNK_DATA::backtrack_dat

Data structure for the Backtracking Linesearch algorithm.

4.34.2.2 Matrix<double> PJFNK_DATA::bestx

Best found solution vector to the non-linear system.

4.34.2.3 BiCGSTAB_DATA PJFNK_DATA::bicgstab_dat

Data structure for the BiCGSTAB method.

4.34.2.4 bool PJFNK_DATA::Bounce = false

True = allow Linesearch to go outside local well, False = Strict local convergence.

4.34.2.5 CGS_DATA PJFNK_DATA::cgs_dat

Data structure for the CGS method.

4.34.2.6 double PJFNK_DATA::eps =sqrt(DBL_EPSILON)

Value of epsilon used jacvec - default = sqrt(DBL_EPSILON)

4.34.2.7 Matrix<double> PJFNK_DATA::F

Stored fuction evaluation at x (also the residual)

4.34.2.8 int(* PJFNK_DATA::funeval)(const Matrix< double > &x, Matrix< double > &F, const void *res_data)

Function pointer for the user's function F(x) using there data.

4.34.2.9 Matrix<double> PJFNK_DATA::Fv

Stored function evaluation at x+eps*v.

4.34.2.10 GCR_DATA PJFNK_DATA::gcr_dat

Data structure for the GCR method.

4.34.2.11 GMRESLP_DATA PJFNK_DATA::gmreslp_dat

Data structure for the GMRESLP method.

4.34.2.12 GMRESR_DATA PJFNK_DATA::gmresr_dat

Data structure for the GMRESR method.

4.34.2.13 GMRESRP_DATA PJFNK_DATA::gmresrp_dat

Data structure for the GMRESRP method.

4.34.2.14 int PJFNK_DATA::l_iter = 0

Number of linear iterations.

4.34.2.15 bool PJFNK_DATA::L_Output = false

True = print Linear messages to console.

4.34.2.16 double PJFNK_DATA::lin_tol_abs = 1e-6

Absolute tolerance of the linear solver - default = 1e-6.

4.34.2.17 double PJFNK_DATA::lin_tol_rel = 1e-6

Relative tolerance of the linear solver - default = 1e-6.

4.34.2.18 int PJFNK_DATA::linear_solver = -1

Flag to denote which linear solver to use - default = PJFNK Chooses.

4.34.2.19 bool PJFNK_DATA::LineSearch = false

True = use Backtracking Linesearch for global convergence.

4.34.2.20 double PJFNK_DATA::nl_bestres

Best found residual norm.

4.34.2.21 int PJFNK_DATA::nl_iter = 0

Number of non-linear iterations.

4.34.2.22 int PJFNK_DATA::nl_maxit = 0

Maximum allowable non-linear steps.

4.34.2.23 bool PJFNK_DATA::NL_Output = true

True = print PJFNK messages to console.

4.34.2.24 double PJFNK_DATA::nl_relres

Relative residual for the non-linear system.

4.34.2.25 double PJFNK_DATA::nl_res

Absolute redidual norm for the non-linear system.

4.34.2.26 double PJFNK_DATA::nl_res_base

Initial residual norm for the non-linear system.

4.34.2.27 double PJFNK_DATA::nl_tol_abs = 1e-6

Absolute Convergence tolerance for non-linear system - default = 1e-6.

4.34.2.28 double PJFNK_DATA::nl_tol_rel = 1e-6

Relative Convergence tol for the non-linear system - default = 1e-6.

4.34.2.29 PCG_DATA PJFNK_DATA::pcg_dat

Data structure for the PCG method.

4.34.2.30 int(∗ PJFNK_DATA::precon)(const Matrix< double > &r, Matrix< double > &p, const void ∗precon_data)

Function pointer for the user's preconditioning function for the linear system.

4.34.2.31 const void∗ PJFNK_DATA::precon_data

Data structure pointer for user's preconditioning data.

4.34.2.32 const void∗ PJFNK_DATA::res_data

Data structure pointer for user's residual data.

4.34.2.33 Matrix<double> PJFNK_DATA::v

Stored vector of x+eps∗v.

4.34.2.34 Matrix<double> PJFNK_DATA::x

Current solution vector for the non-linear system.

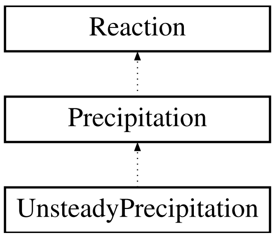
The documentation for this struct was generated from the following file:

- lark.h

4.35 Precipitation Class Reference

```
#include <shark.h>
```

Inheritance diagram for Precipitation:



Additional Inherited Members

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

- shark.h

4.36 PURE_GAS Struct Reference

```
#include <egret.h>
```

Public Attributes

- double molecular_weight
- double Sutherland_Temp
- double Sutherland_Const
- double Sutherland_Viscosity
- double specific_heat
- double molecular_diffusion
- double dynamic_viscosity
- double density
- double Schmidt

4.36.1 Member Data Documentation

4.36.1.1 double PURE_GAS::density

4.36.1.2 double PURE_GAS::dynamic_viscosity

4.36.1.3 double PURE_GAS::molecular_diffusion

4.36.1.4 double PURE_GAS::molecular_weight

4.36.1.5 double PURE_GAS::Schmidt

4.36.1.6 double PURE_GAS::specific_heat

4.36.1.7 double PURE_GAS::Sutherland_Const

4.36.1.8 double PURE_GAS::Sutherland_Temp

4.36.1.9 double PURE_GAS::Sutherland_Viscosity

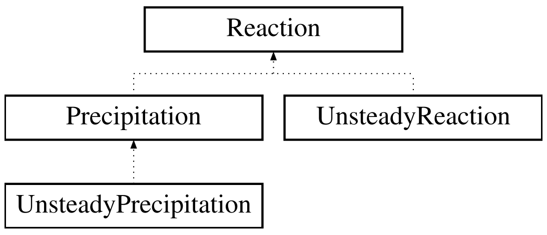
The documentation for this struct was generated from the following file:

- egret.h

4.37 Reaction Class Reference

```
#include <shark.h>
```

Inheritance diagram for Reaction:



Public Member Functions

- Reaction ()
- ∼Reaction ()
- void Initialize_List (MasterSpeciesList &List)
- void Display_Info ()
- void Set_Stoichiometric (int i, double v)
- void Set_Equilibrium (double v)
- void Set_Enthalpy (double H)
- void Set_Entropy (double S)
- void Set_EnthalpyANDEntropy (double H, double S)
- void Set_Energy (double G)
- void checkSpeciesEnergies ()
- void calculateEnergies ()
- void calculateEquilibrium (double T)
- bool haveEquilibrium ()
- double Get_Stoichiometric (int i)
- double Get_Equilibrium ()
- double Get_Enthalpy ()
- double Get_Entropy ()
- double Get_Energy ()
- double Eval_Residual (const Matrix< double > &x, const Matrix< double > &gama)

Protected Attributes

- MasterSpeciesList ∗ List
- std::vector< double > Stoichiometric
- double Equilibrium
- double enthalpy
- double entropy

- double energy
- bool CanCalcHS
- bool CanCalcG
- bool HaveHS
- bool HaveG
- bool HaveEquil

4.37.1 Constructor & Destructor Documentation

4.37.1.1 Reaction::Reaction ( )

4.37.1.2 Reaction::∼Reaction ( )

4.37.2 Member Function Documentation

4.37.2.1 void Reaction::calculateEnergies ( )

4.37.2.2 void Reaction::calculateEquilibrium (double *T*)

4.37.2.3 void Reaction::checkSpeciesEnergies ( )

4.37.2.4 void Reaction::Display_Info ( )

4.37.2.5 double Reaction::Eval_Residual (const Matrix< double > & *x*, const Matrix< double > & *gama*)

4.37.2.6 double Reaction::Get_Energy ( )

4.37.2.7 double Reaction::Get_Enthalpy ( )

4.37.2.8 double Reaction::Get_Entropy ( )

4.37.2.9 double Reaction::Get_Equilibrium ( )

4.37.2.10 double Reaction::Get_Stoichiometric (int *i*)

4.37.2.11 bool Reaction::haveEquilibrium ( )

4.37.2.12 void Reaction::Initialize_List (MasterSpeciesList & *List*)

4.37.2.13 void Reaction::Set_Energy (double *G*)

4.37.2.14 void Reaction::Set_Enthalpy (double *H*)

4.37.2.15 void Reaction::Set_EnthalpyANDEntropy (double *H*, double *S*)

4.37.2.16 void Reaction::Set_Entropy (double *S*)

4.37.2.17 void Reaction::Set_Equilibrium (double *v*)

4.37.2.18 void Reaction::Set_Stoichiometric (int *i*, double *v*)

4.37.3 Member Data Documentation

4.37.3.1 bool Reaction::CanCalcG [protected]

4.37.3.2 bool Reaction::CanCalcHS [protected]

4.37.3.3 double Reaction::energy [protected]

4.37.3.4 double Reaction::enthalpy [protected]

4.37.3.5 double Reaction::entropy [protected]

4.37.3.6 double Reaction::Equilibrium [protected]

4.37.3.7 bool Reaction::HaveEquil [protected]

4.37.3.8 bool Reaction::HaveG [protected]

4.37.3.9 bool Reaction::HaveHS [protected]

4.37.3.10 MasterSpeciesList* Reaction::List [protected]

4.37.3.11 std::vector<double> Reaction::Stoichiometric [protected]

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

- shark.h
- shark.cpp

4.38 SCOPSOWL_DATA Struct Reference

```
#include <scopsowl.h>
```

Public Attributes

- unsigned long int total_steps
- int coord_macro
- int coord_micro
- int level = 2
- double sim_time
- double t_old
- double t
- double t_counter = 0.0
- double t_print
- bool Print2File = true
- bool Print2Console = true
- bool SurfDiff = true
- bool Heterogeneous = true
- double gas_velocity
- double total_pressure
- double gas_temperature
- double pellet_radius
- double crystal_radius
- double char_macro
- double char_micro
- double binder_fraction
- double binder_porosity

- double binder_poresize
- double pellet_density
- bool DirichletBC = false
- bool NonLinear = true
- std::vector< double > y
- std::vector< double > tempy
- FILE ∗ OutputFile
- double(∗ eval_ads)(int i, int l, const void ∗user_data)
- double(∗ eval_retard)(int i, int l, const void ∗user_data)
- double(∗ eval_diff)(int i, int l, const void ∗user_data)
- double(∗ eval_surfDiff)(int i, int l, const void ∗user_data)
- double(∗ eval_kf)(int i, const void ∗user_data)
- const void ∗ user_data
- MIXED_GAS ∗ gas_dat
- MAGPIE_DATA magpie_dat
- std::vector< FINCH_DATA > finch_dat
- std::vector< SCOPSOWL_PARAM_DATA > param_dat
- std::vector< SKUA_DATA > skua_dat

4.38.1 Member Data Documentation

4.38.1.1 double SCOPSOWL_DATA::binder_fraction

4.38.1.2 double SCOPSOWL_DATA::binder_poresize

4.38.1.3 double SCOPSOWL_DATA::binder_porosity

4.38.1.4 double SCOPSOWL_DATA::char_macro

4.38.1.5 double SCOPSOWL_DATA::char_micro

4.38.1.6 int SCOPSOWL_DATA::coord_macro

4.38.1.7 int SCOPSOWL_DATA::coord_micro

4.38.1.8 double SCOPSOWL_DATA::crystal_radius

4.38.1.9 bool SCOPSOWL_DATA::DirichletBC = false

4.38.1.10 double(∗ SCOPSOWL_DATA::eval_ads)(int i, int l, const void ∗user_data)

4.38.1.11 double(∗ SCOPSOWL_DATA::eval_diff)(int i, int l, const void ∗user_data)

4.38.1.12 double(∗ SCOPSOWL_DATA::eval_kf)(int i, const void ∗user_data)

4.38.1.13 double(∗ SCOPSOWL_DATA::eval_retard)(int i, int l, const void ∗user_data)

4.38.1.14 double(∗ SCOPSOWL_DATA::eval_surfDiff)(int i, int l, const void ∗user_data)

4.38.1.15 std::vector<FINCH_DATA> SCOPSOWL_DATA::finch_dat

4.38.1.16 MIXED_GAS∗ SCOPSOWL_DATA::gas_dat

4.38.1.17 double SCOPSOWL_DATA::gas_temperature

4.38.1.18 double SCOPSOWL_DATA::gas_velocity

4.38.1.19 bool SCOPSOWL_DATA::Heterogeneous = true

4.38.1.20 int SCOPSOWL_DATA::level = 2

4.38.1.21 MAGPIE_DATA SCOPSOWL_DATA::magpie_dat

4.38.1.22 bool SCOPSOWL_DATA::NonLinear = true

4.38.1.23 FILE* SCOPSOWL_DATA::OutputFile

4.38.1.24 std::vector<SCOPSOWL_PARAM_DATA> SCOPSOWL_DATA::param_dat

4.38.1.25 double SCOPSOWL_DATA::pellet_density

4.38.1.26 double SCOPSOWL_DATA::pellet_radius

4.38.1.27 bool SCOPSOWL_DATA::Print2Console = true

4.38.1.28 bool SCOPSOWL_DATA::Print2File = true

4.38.1.29 double SCOPSOWL_DATA::sim_time

4.38.1.30 std::vector<SKUA_DATA> SCOPSOWL_DATA::skua_dat

4.38.1.31 bool SCOPSOWL_DATA::SurfDiff = true

4.38.1.32 double SCOPSOWL_DATA::t

4.38.1.33 double SCOPSOWL_DATA::t_counter = 0.0

4.38.1.34 double SCOPSOWL_DATA::t_old

4.38.1.35 double SCOPSOWL_DATA::t_print

4.38.1.36 std::vector<double> SCOPSOWL_DATA::tempy

4.38.1.37 double SCOPSOWL_DATA::total_pressure

4.38.1.38 unsigned long int SCOPSOWL_DATA::total_steps

4.38.1.39 const void* SCOPSOWL_DATA::user_data

4.38.1.40 std::vector<double> SCOPSOWL_DATA::y

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

- scopsowl.h

4.39 SCOPSOWL_OPT_DATA Struct Reference

```
#include <scopsowl_opt.h>
```

Public Attributes

- int num_curves
- int evaluation
- unsigned long int total_eval
- int current_points
- int num_params = 1
- int diffusion_type
- int adsorb_index
- int max_guess_iter = 20
- bool Optimize
- bool Rough
- double current_temp
- double current_press
- double current_equil
- double simulation_equil
- double max_bias
- double min_bias
- double e_norm
- double f_bias
- double e_norm_old
- double f_bias_old
- double param_guess
- double param_guess_old
- double rel_tol_norm = 0.01
- double abs_tol_bias = 1.0
- std::vector< double > y_base
- std::vector< double > q_data
- std::vector< double > q_sim
- std::vector< double > t
- FILE ∗ ParamFile
- FILE ∗ CompareFile
- SCOPSOWL_DATA owl_dat

4.39.1 Member Data Documentation

4.39.1.1 double SCOPSOWL_OPT_DATA::abs_tol_bias = 1.0

4.39.1.2 int SCOPSOWL_OPT_DATA::adsorb_index

4.39.1.3 FILE∗ SCOPSOWL_OPT_DATA::CompareFile

4.39.1.4 double SCOPSOWL_OPT_DATA::current_equil

4.39.1.5 int SCOPSOWL_OPT_DATA::current_points

4.39.1.6 double SCOPSOWL_OPT_DATA::current_press

4.39.1.7 double SCOPSOWL_OPT_DATA::current_temp

4.39.1.8 int SCOPSOWL_OPT_DATA::diffusion_type

4.39.1.9 double SCOPSOWL_OPT_DATA::e_norm

4.39.1.10 double SCOPSOWL_OPT_DATA::e_norm_old

4.39.1.11 int SCOPSOWL_OPT_DATA::evaluation

4.39.1.12 double SCOPSOWL_OPT_DATA::f_bias

4.39.1.13 double SCOPSOWL_OPT_DATA::f_bias_old

4.39.1.14 double SCOPSOWL_OPT_DATA::max_bias

4.39.1.15 int SCOPSOWL_OPT_DATA::max_guess_iter = 20

4.39.1.16 double SCOPSOWL_OPT_DATA::min_bias

4.39.1.17 int SCOPSOWL_OPT_DATA::num_curves

4.39.1.18 int SCOPSOWL_OPT_DATA::num_params = 1

4.39.1.19 bool SCOPSOWL_OPT_DATA::Optimize

4.39.1.20 SCOPSOWL_DATA SCOPSOWL_OPT_DATA::owl_dat

4.39.1.21 double SCOPSOWL_OPT_DATA::param_guess

4.39.1.22 double SCOPSOWL_OPT_DATA::param_guess_old

4.39.1.23 FILE* SCOPSOWL_OPT_DATA::ParamFile

4.39.1.24 std::vector<double> SCOPSOWL_OPT_DATA::q_data

4.39.1.25 std::vector<double> SCOPSOWL_OPT_DATA::q_sim

4.39.1.26 double SCOPSOWL_OPT_DATA::rel_tol_norm = 0.01

4.39.1.27 bool SCOPSOWL_OPT_DATA::Rough

4.39.1.28 double SCOPSOWL_OPT_DATA::simulation_equil

4.39.1.29 std::vector<double> SCOPSOWL_OPT_DATA::t

4.39.1.30 unsigned long int SCOPSOWL_OPT_DATA::total_eval

4.39.1.31 std::vector<double> SCOPSOWL_OPT_DATA::y_base

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

- scopsowl_opt.h

4.40 SCOPSOWL_PARAM_DATA Struct Reference

```
#include <scopsowl.h>
```

Public Attributes

- Matrix< double > qAvg

- Matrix< double > qAvg_old
- Matrix< double > Qst
- Matrix< double > Qst_old
- Matrix< double > dq_dc
- double xIC
- double qIntegralAvg
- double qIntegralAvg_old
- double QstAvg
- double QstAvg_old
- double qo
- double Qsto
- double dq_dco
- double pore_diffusion
- double film_transfer
- double activation_energy
- double ref_diffusion
- double ref_temperature
- double affinity
- double ref_pressure
- bool Adsorbable
- std::string speciesName

4.40.1 Member Data Documentation

4.40.1.1 double SCOPSOWL_PARAM_DATA::activation_energy

4.40.1.2 bool SCOPSOWL_PARAM_DATA::Adsorbable

4.40.1.3 double SCOPSOWL_PARAM_DATA::affinity

4.40.1.4 Matrix<double> SCOPSOWL_PARAM_DATA::dq_dc

4.40.1.5 double SCOPSOWL_PARAM_DATA::dq_dco

4.40.1.6 double SCOPSOWL_PARAM_DATA::film_transfer

4.40.1.7 double SCOPSOWL_PARAM_DATA::pore_diffusion

4.40.1.8 Matrix<double> SCOPSOWL_PARAM_DATA::qAvg

4.40.1.9 Matrix<double> SCOPSOWL_PARAM_DATA::qAvg_old

4.40.1.10 double SCOPSOWL_PARAM_DATA::qIntegralAvg

4.40.1.11 double SCOPSOWL_PARAM_DATA::qIntegralAvg_old

4.40.1.12 double SCOPSOWL_PARAM_DATA::qo

4.40.1.13 Matrix<double> SCOPSOWL_PARAM_DATA::Qst

4.40.1.14 Matrix<double> SCOPSOWL_PARAM_DATA::Qst_old

4.40.1.15 double SCOPSOWL_PARAM_DATA::QstAvg

4.40.1.16 double SCOPSOWL_PARAM_DATA::QstAvg_old

4.40.1.17 double SCOPSOWL_PARAM_DATA::Qsto

4.40.1.18 double SCOPSOWL_PARAM_DATA::ref_diffusion

4.40.1.19 double SCOPSOWL_PARAM_DATA::ref_pressure

4.40.1.20 double SCOPSOWL_PARAM_DATA::ref_temperature

4.40.1.21 std::string SCOPSOWL_PARAM_DATA::speciesName

4.40.1.22 double SCOPSOWL_PARAM_DATA::xIC

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

- scopsowl.h

4.41 SHARK_DATA Struct Reference

```
#include <shark.h>
```

Public Attributes

- MasterSpeciesList MasterList
- std::vector< Reaction > ReactionList
- std::vector< MassBalance > MassBalanceList
- std::vector< UnsteadyReaction > UnsteadyList
- std::vector< double(∗)(const Matrix< double > &x, SHARK_DATA ∗shark_dat, const void ∗data) > OtherList
- int numvar
- int num_ssr
- int num_mbe
- int num_usr
- int num_other = 0
- int act_fun = IDEAL
- int totalsteps = 0
- int timesteps = 0
- int pH_index = -1
- int pOH_index = -1
- double simulationtime = 0.0
- double dt = 0.1
- double dt_min = sqrt(DBL_EPSILON)
- double t_out = 0.0
- double t_count = 0.0
- double time = 0.0
- double time_old = 0.0
- double pH = 7.0
- double Norm = 0.0
- double dielectric_const = 78.325
- double temperature = 298.15

- bool steadystate = true
- bool TimeAdaptivity = false
- bool const_pH = false
- bool SpeciationCurve = false
- bool Console_Output = true
- bool File_Output = false
- bool Contains_pH = false
- bool Contains_pOH = false
- bool Converged = false
- Matrix< double > X_old
- Matrix< double > X_new
- Matrix< double > Conc_old
- Matrix< double > Conc_new
- Matrix< double > activity_new
- Matrix< double > activity_old
- int(∗ EvalActivity)(const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int(∗ Residual)(const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int(∗ lin_precon)(const Matrix< double > &r, Matrix< double > &p, const void ∗data)
- PJFNK_DATA Newton_data
- const void ∗ activity_data
- const void ∗ residual_data
- const void ∗ precon_data
- const void ∗ other_data
- FILE ∗ OutputFile
- yaml_cpp_class yaml_object

4.41.1 Member Data Documentation

4.41.1.1 int SHARK_DATA::act_fun = IDEAL

4.41.1.2 const void∗ SHARK_DATA::activity_data

4.41.1.3 Matrix<double> SHARK_DATA::activity_new

4.41.1.4 Matrix<double> SHARK_DATA::activity_old

4.41.1.5 Matrix<double> SHARK_DATA::Conc_new

4.41.1.6 Matrix<double> SHARK_DATA::Conc_old

4.41.1.7 bool SHARK_DATA::Console_Output = true

4.41.1.8 bool SHARK_DATA::const_pH = false

4.41.1.9 bool SHARK_DATA::Contains_pH = false

4.41.1.10 bool SHARK_DATA::Contains_pOH = false

4.41.1.11 bool SHARK_DATA::Converged = false

4.41.1.12 double SHARK_DATA::dielectric_const = 78.325

4.41.1.13 double SHARK_DATA::dt = 0.1

4.41.1.14 double SHARK_DATA::dt_min = sqrt(DBL_EPSILON)

4.41.1.15 int(∗ SHARK_DATA::EvalActivity)(const Matrix< double > &x, Matrix< double > &F, const void ∗data)

4.41.1.16 bool SHARK_DATA::File_Output = false

4.41.1.17 int(∗ SHARK_DATA::lin_precon)(const Matrix< double > &r, Matrix< double > &p, const void ∗data)

4.41.1.18 std::vector<MassBalance> SHARK_DATA::MassBalanceList

4.41.1.19 MasterSpeciesList SHARK_DATA::MasterList

4.41.1.20 PJFNK_DATA SHARK_DATA::Newton_data

4.41.1.21 double SHARK_DATA::Norm = 0.0

4.41.1.22 int SHARK_DATA::num_mbe

4.41.1.23 int SHARK_DATA::num_other = 0

4.41.1.24 int SHARK_DATA::num_ssr

4.41.1.25 int SHARK_DATA::num_usr

4.41.1.26 int SHARK_DATA::numvar

4.41.1.27 const void∗ SHARK_DATA::other_data

4.41.1.28 std::vector< double (∗) (const Matrix<double> &x, SHARK_DATA ∗shark_dat, const void ∗data) > SHARK_DATA::OtherList

4.41.1.29 FILE∗ SHARK_DATA::OutputFile

4.41.1.30 double SHARK_DATA::pH = 7.0

4.41.1.31 int SHARK_DATA::pH_index = -1

4.41.1.32 int SHARK_DATA::pOH_index = -1

4.41.1.33 const void∗ SHARK_DATA::precon_data

4.41.1.34 std::vector<Reaction> SHARK_DATA::ReactionList

4.41.1.35 int(∗ SHARK_DATA::Residual)(const Matrix< double > &x, Matrix< double > &F, const void ∗data)

4.41.1.36 const void∗ SHARK_DATA::residual_data

4.41.1.37 double SHARK_DATA::simulationtime = 0.0

4.41.1.38 bool SHARK_DATA::SpeciationCurve = false

4.41.1.39 bool SHARK_DATA::steadystate = true

4.41.1.40 double SHARK_DATA::t_count = 0.0

4.41.1.41 double SHARK_DATA::t_out = 0.0

4.41.1.42 double SHARK_DATA::temperature = 298.15

4.41.1.43 double SHARK_DATA::time = 0.0

4.41.1.44 double SHARK_DATA::time_old = 0.0

4.41.1.45 bool SHARK_DATA::TimeAdaptivity = false

4.41.1.46 int SHARK_DATA::timesteps = 0

4.41.1.47 int SHARK_DATA::totalsteps = 0

4.41.1.48 std::vector<UnsteadyReaction> SHARK_DATA::UnsteadyList

4.41.1.49 Matrix<double> SHARK_DATA::X_new

4.41.1.50 Matrix<double> SHARK_DATA::X_old

4.41.1.51 yaml_cpp_class SHARK_DATA::yaml_object

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

- shark.h

4.42 SKUA_DATA Struct Reference

```
#include <skua.h>
```

Public Attributes

- unsigned long int total_steps
- int coord
- double sim_time
- double t_old
- double t
- double t_counter = 0.0
- double t_print
- double qTn
- double qTnp1
- bool Print2File = true
- bool Print2Console = true
- double gas_velocity
- double pellet_radius
- double char_measure
- bool DirichletBC = true
- bool NonLinear = true
- std::vector< double > y
- FILE ∗ OutputFile
- double(∗ eval_diff)(int i, int l, const void ∗user_data)
- double(∗ eval_kf)(int i, const void ∗user_data)
- const void ∗ user_data
- MAGPIE_DATA magpie_dat
- MIXED_GAS ∗ gas_dat
- std::vector< FINCH_DATA > finch_dat
- std::vector< SKUA_PARAM > param_dat

4.42.1 Member Data Documentation

4.42.1.1 double SKUA_DATA::char_measure

4.42.1.2 int SKUA_DATA::coord

4.42.1.3 bool SKUA_DATA::DirichletBC = true

4.42.1.4 double(∗ SKUA_DATA::eval_diff)(int i, int l, const void ∗user_data)

4.42.1.5 double(∗ SKUA_DATA::eval_kf)(int i, const void ∗user_data)

4.42.1.6 std::vector<FINCH_DATA> SKUA_DATA::finch_dat

4.42.1.7 MIXED_GAS∗ SKUA_DATA::gas_dat

4.42.1.8 double SKUA_DATA::gas_velocity

4.42.1.9 MAGPIE_DATA SKUA_DATA::magpie_dat

4.42.1.10 bool SKUA_DATA::NonLinear = true

4.42.1.11 FILE∗ SKUA_DATA::OutputFile

4.42.1.12 std::vector<SKUA_PARAM> SKUA_DATA::param_dat

4.42.1.13 double SKUA_DATA::pellet_radius

4.42.1.14 bool SKUA_DATA::Print2Console = true

4.42.1.15 bool SKUA_DATA::Print2File = true

4.42.1.16 double SKUA_DATA::qTn

4.42.1.17 double SKUA_DATA::qTnp1

4.42.1.18 double SKUA_DATA::sim_time

4.42.1.19 double SKUA_DATA::t

4.42.1.20 double SKUA_DATA::t_counter = 0.0

4.42.1.21 double SKUA_DATA::t_old

4.42.1.22 double SKUA_DATA::t_print

4.42.1.23 unsigned long int SKUA_DATA::total_steps

4.42.1.24 const void∗ SKUA_DATA::user_data

4.42.1.25 std::vector<double> SKUA_DATA::y

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

- skua.h

4.43 SKUA_OPT_DATA Struct Reference

```
#include <skua_opt.h>
```

Public Attributes

- int num_curves
- int evaluation
- unsigned long int total_eval
- int current_points
- int num_params = 1
- int diffusion_type
- int adsorb_index
- int max_guess_iter = 20
- bool Optimize
- bool Rough
- double current_temp
- double current_press
- double current_equil
- double simulation_equil
- double max_bias
- double min_bias
- double e_norm
- double f_bias
- double e_norm_old
- double f_bias_old
- double param_guess
- double param_guess_old
- double rel_tol_norm = 0.1
- double abs_tol_bias = 0.1
- std::vector< double > y_base
- std::vector< double > q_data
- std::vector< double > q_sim
- std::vector< double > t
- FILE ∗ ParamFile
- FILE ∗ CompareFile
- SKUA_DATA skua_dat

4.43.1 Member Data Documentation

4.43.1.1 double SKUA_OPT_DATA::abs_tol_bias = 0.1

4.43.1.2 int SKUA_OPT_DATA::adsorb_index

4.43.1.3 FILE∗ SKUA_OPT_DATA::CompareFile

4.43.1.4 double SKUA_OPT_DATA::current_equil

4.43.1.5 int SKUA_OPT_DATA::current_points

4.43.1.6 double SKUA_OPT_DATA::current_press

4.43.1.7 double SKUA_OPT_DATA::current_temp

4.43.1.8 int SKUA_OPT_DATA::diffusion_type

4.43.1.9 double SKUA_OPT_DATA::e_norm

4.43.1.10 double SKUA_OPT_DATA::e_norm_old

4.43.1.11 int SKUA_OPT_DATA::evaluation

4.43.1.12 double SKUA_OPT_DATA::f_bias

4.43.1.13 double SKUA_OPT_DATA::f_bias_old

4.43.1.14 double SKUA_OPT_DATA::max_bias

4.43.1.15 int SKUA_OPT_DATA::max_guess_iter = 20

4.43.1.16 double SKUA_OPT_DATA::min_bias

4.43.1.17 int SKUA_OPT_DATA::num_curves

4.43.1.18 int SKUA_OPT_DATA::num_params = 1

4.43.1.19 bool SKUA_OPT_DATA::Optimize

4.43.1.20 double SKUA_OPT_DATA::param_guess

4.43.1.21 double SKUA_OPT_DATA::param_guess_old

4.43.1.22 FILE* SKUA_OPT_DATA::ParamFile

4.43.1.23 std::vector<double> SKUA_OPT_DATA::q_data

4.43.1.24 std::vector<double> SKUA_OPT_DATA::q_sim

4.43.1.25 double SKUA_OPT_DATA::rel_tol_norm = 0.1

4.43.1.26 bool SKUA_OPT_DATA::Rough

4.43.1.27 double SKUA_OPT_DATA::simulation_equil

4.43.1.28 SKUA_DATA SKUA_OPT_DATA::skua_dat

4.43.1.29 std::vector<double> SKUA_OPT_DATA::t

4.43.1.30 unsigned long int SKUA_OPT_DATA::total_eval

4.43.1.31 std::vector<double> SKUA_OPT_DATA::y_base

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

- skua_opt.h

4.44 SKUA_PARAM Struct Reference

```
#include <skua.h>
```

Public Attributes

- double activation_energy
- double ref_diffusion
- double ref_temperature
- double affinity
- double ref_pressure
- double film_transfer
- double xIC
- double y_eff
- double Qstn
- double Qstnp1
- double xn
- double xnp1
- bool Adsorbable
- std::string speciesName

4.44.1 Member Data Documentation

4.44.1.1 double SKUA_PARAM::activation_energy

4.44.1.2 bool SKUA_PARAM::Adsorbable

4.44.1.3 double SKUA_PARAM::affinity

4.44.1.4 double SKUA_PARAM::film_transfer

4.44.1.5 double SKUA_PARAM::Qstn

4.44.1.6 double SKUA_PARAM::Qstnp1

4.44.1.7 double SKUA_PARAM::ref_diffusion

4.44.1.8 double SKUA_PARAM::ref_pressure

4.44.1.9 double SKUA_PARAM::ref_temperature

4.44.1.10 std::string SKUA_PARAM::speciesName

4.44.1.11 double SKUA_PARAM::xIC

4.44.1.12 double SKUA_PARAM::xn

4.44.1.13 double SKUA_PARAM::xnp1

4.44.1.14 double SKUA_PARAM::y_eff

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

- skua.h

4.45 Speciation_Test01_Data Struct Reference

```
#include <sandbox.h>
```

Public Attributes

- int N = 4
- const double logKw = -14.0
- const double logKa1 = -6.35
- const double logKa2 = -10.33
- double CT = 0.1786
- double NaT = 0.1786
- std::vector< Molecule > x
- Matrix< double > Jacobian
- Matrix< double > NumJac
- Matrix< double > logC
- Matrix< double > C

4.45.1 Member Data Documentation

4.45.1.1 Matrix<double> Speciation_Test01_Data::C

4.45.1.2 double Speciation_Test01_Data::CT = 0.1786

4.45.1.3 Matrix<double> Speciation_Test01_Data::Jacobian

4.45.1.4 Matrix<double> Speciation_Test01_Data::logC

4.45.1.5 const double Speciation_Test01_Data::logKa1 = -6.35

4.45.1.6 const double Speciation_Test01_Data::logKa2 = -10.33

4.45.1.7 const double Speciation_Test01_Data::logKw = -14.0

4.45.1.8 int Speciation_Test01_Data::N = 4

4.45.1.9 double Speciation_Test01_Data::NaT = 0.1786

4.45.1.10 Matrix<double> Speciation_Test01_Data::NumJac

4.45.1.11 std::vector<Molecule> Speciation_Test01_Data::x

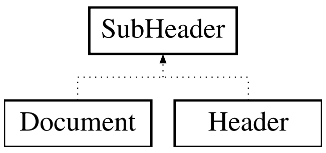
The documentation for this struct was generated from the following file:

- sandbox.h

4.46 SubHeader Class Reference

```
#include <yaml_wrapper.h>
```

Inheritance diagram for SubHeader:



Public Member Functions

- SubHeader ()
- ∼SubHeader ()
- SubHeader (const SubHeader &subheader)
- SubHeader (const KeyValueMap &map)
- SubHeader (std::string name)
- SubHeader (std::string name, const KeyValueMap &map)
- SubHeader & operator= (const SubHeader &sub)
- ValueTypePair & operator[] (const std::string key)
- ValueTypePair operator[] (const std::string key) const
- KeyValueMap & getMap ()
- void clear ()
- void addPair (std::string key, std::string val)
- void addPair (std::string key, std::string val, int type)
- void setName (std::string name)
- void setAlias (std::string alias)
- void setAlias (std::string alias, int state)
- void setNameAliasPair (std::string name, std::string alias, int state)
- void setState (int state)
- void DisplayContents ()
- std::string getName ()
- std::string getAlias ()
- bool isAlias ()
- bool isAnchor ()
- int getState ()

Protected Attributes

- KeyValueMap Data_Map
- std::string name
- std::string alias
- int state

4.46.1 Constructor & Destructor Documentation

4.46.1.1 SubHeader::SubHeader ( )

4.46.1.2 SubHeader::∼SubHeader ( )

4.46.1.3 SubHeader::SubHeader (const SubHeader & *subheader*)

4.46.1.4 SubHeader::SubHeader (const KeyValueMap & *map*)

4.46.1.5 SubHeader::SubHeader (std::string *name*)

4.46.1.6 SubHeader::SubHeader (std::string *name,* const KeyValueMap & *map*)

4.46.2 Member Function Documentation

4.46.2.1 void SubHeader::addPair (std::string *key,* std::string *val*)

4.46.2.2 void SubHeader::addPair (std::string *key,* std::string *val,* int *type*)

4.46.2.3 void SubHeader::clear ( )

4.46.2.4 void SubHeader::DisplayContents ( )

4.46.2.5 std::string SubHeader::getAlias ( )

4.46.2.6 KeyValueMap & SubHeader::getMap ( )

4.46.2.7 std::string SubHeader::getName ( )

4.46.2.8 int SubHeader::getState ( )

4.46.2.9 bool SubHeader::isAlias ( )

4.46.2.10 bool SubHeader::isAnchor ( )

4.46.2.11 SubHeader & SubHeader::operator= (const SubHeader & *sub*)

4.46.2.12 ValueTypePair & SubHeader::operator[] (const std::string *key*)

4.46.2.13 ValueTypePair SubHeader::operator[] (const std::string *key*) const

4.46.2.14 void SubHeader::setAlias (std::string *alias*)

4.46.2.15 void SubHeader::setAlias (std::string *alias,* int *state*)

4.46.2.16 void SubHeader::setName (std::string *name*)

4.46.2.17 void SubHeader::setNameAliasPair (std::string *name,* std::string *alias,* int *state*)

4.46.2.18 void SubHeader::setState (int *state*)

4.46.3 Member Data Documentation

4.46.3.1 std::string SubHeader::alias `[protected]`

4.46.3.2 KeyValueMap SubHeader::Data_Map `[protected]`

4.46.3.3 std::string SubHeader::name `[protected]`

4.46.3.4 int SubHeader::state `[protected]`

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

- yaml_wrapper.h
- yaml_wrapper.cpp

4.47 SYSTEM_DATA Struct Reference

```
#include <magpie.h>
```

Public Attributes

- double T

- double PT
- double qT
- double PI
- double pi
- double As
- int N
- int I
- int J
- int K
- unsigned long int total_eval
- double avg_norm
- double max_norm
- int Sys
- int Par
- bool Recover
- bool Carrier
- bool Ideal
- bool Output

4.47.1 Member Data Documentation

4.47.1.1 double SYSTEM_DATA::As

4.47.1.2 double SYSTEM_DATA::avg_norm

4.47.1.3 bool SYSTEM_DATA::Carrier

4.47.1.4 int SYSTEM_DATA::I

4.47.1.5 bool SYSTEM_DATA::Ideal

4.47.1.6 int SYSTEM_DATA::J

4.47.1.7 int SYSTEM_DATA::K

4.47.1.8 double SYSTEM_DATA::max_norm

4.47.1.9 int SYSTEM_DATA::N

4.47.1.10 bool SYSTEM_DATA::Output

4.47.1.11 int SYSTEM_DATA::Par

4.47.1.12 double SYSTEM_DATA::PI

4.47.1.13 double SYSTEM_DATA::pi

4.47.1.14 double SYSTEM_DATA::PT

4.47.1.15 double SYSTEM_DATA::qT

4.47.1.16 bool SYSTEM_DATA::Recover

4.47.1.17 int SYSTEM_DATA::Sys

4.47.1.18 double SYSTEM_DATA::T

4.47.1.19 unsigned long int SYSTEM_DATA::total_eval

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

- magpie.h

4.48 TRAJECTORY_DATA Struct Reference

```
#include <Trajectory.h>
```

Public Attributes

- double mu_0 = 12.57e-7
- double rho_f = 1000.0
- double eta = 0.001
- double Hamaker = 1.3e-21
- double Temp = 298
- double k = 1.38e-23
- double Rs = 0.0026925
- double L = 0.0611
- double porosity = 0.8979
- double V_separator
- double a = 33.0e-6
- double V_wire
- double L_wire
- double A_separator
- double A_wire
- double B0 = 1.0
- double H0
- double Ms = 0.6
- double b = 0.25e-6
- double chi_p = 3.87e-6
- double rho_p = 8700.0
- double Q_in
- double V0
- double Y_initial = 20.0
- double dt
- double M
- double mp
- double beta
- double q_bar
- double sigma_v
- double sigma_vz
- double sigma_z
- double sigma_n
- double sigma_m
- double n_rand
- double m_rand
- double s_rand
- double t_rand
- Matrix< double > POL

- Matrix< double > H
- Matrix< double > dX
- Matrix< double > dY
- Matrix< double > X
- Matrix< double > Y
- Matrix< int > Cap

4.48.1 Member Data Documentation

4.48.1.1 double TRAJECTORY_DATA::a = 33.0e-6

4.48.1.2 double TRAJECTORY_DATA::A_separator

4.48.1.3 double TRAJECTORY_DATA::A_wire

4.48.1.4 double TRAJECTORY_DATA::b = 0.25e-6

4.48.1.5 double TRAJECTORY_DATA::B0 = 1.0

4.48.1.6 double TRAJECTORY_DATA::beta

4.48.1.7 Matrix<int> TRAJECTORY_DATA::Cap

4.48.1.8 double TRAJECTORY_DATA::chi_p = 3.87e-6

4.48.1.9 double TRAJECTORY_DATA::dt

4.48.1.10 Matrix<double> TRAJECTORY_DATA::dX

4.48.1.11 Matrix<double> TRAJECTORY_DATA::dY

4.48.1.12 double TRAJECTORY_DATA::eta = 0.001

4.48.1.13 Matrix<double> TRAJECTORY_DATA::H

4.48.1.14 double TRAJECTORY_DATA::H0

4.48.1.15 double TRAJECTORY_DATA::Hamaker = 1.3e-21

4.48.1.16 double TRAJECTORY_DATA::k = 1.38e-23

4.48.1.17 double TRAJECTORY_DATA::L = 0.0611

4.48.1.18 double TRAJECTORY_DATA::L_wire

4.48.1.19 double TRAJECTORY_DATA::M

4.48.1.20 double TRAJECTORY_DATA::m_rand

4.48.1.21 double TRAJECTORY_DATA::mp

4.48.1.22 double TRAJECTORY_DATA::Ms = 0.6

4.48.1.23 double TRAJECTORY_DATA::mu_0 = 12.57e-7

4.48.1.24 double TRAJECTORY_DATA::n_rand

4.48.1.25 Matrix<double> TRAJECTORY_DATA::POL

4.48.1.26 double TRAJECTORY_DATA::porosity = 0.8979

4.48.1.27 double TRAJECTORY_DATA::q_bar

4.48.1.28 double TRAJECTORY_DATA::Q_in

4.48.1.29 double TRAJECTORY_DATA::rho_f = 1000.0

4.48.1.30 double TRAJECTORY_DATA::rho_p = 8700.0

4.48.1.31 double TRAJECTORY_DATA::Rs = 0.0026925

4.48.1.32 double TRAJECTORY_DATA::s_rand

4.48.1.33 double TRAJECTORY_DATA::sigma_m

4.48.1.34 double TRAJECTORY_DATA::sigma_n

4.48.1.35 double TRAJECTORY_DATA::sigma_v

4.48.1.36 double TRAJECTORY_DATA::sigma_vz

4.48.1.37 double TRAJECTORY_DATA::sigma_z

4.48.1.38 double TRAJECTORY_DATA::t_rand

4.48.1.39 double TRAJECTORY_DATA::Temp = 298

4.48.1.40 double TRAJECTORY_DATA::V0

4.48.1.41 double TRAJECTORY_DATA::V_separator

4.48.1.42 double TRAJECTORY_DATA::V_wire

4.48.1.43 Matrix<double> TRAJECTORY_DATA::X

4.48.1.44 Matrix<double> TRAJECTORY_DATA::Y

4.48.1.45 double TRAJECTORY_DATA::Y_initial = 20.0

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

- Trajectory.h

4.49 UI_DATA Struct Reference

Data structure holding the UI arguments.

`#include <ui.h>`

Public Attributes

- ValueTypePair value_type

    Data pair for input, tells what the input is and it's type.
- std::vector< std::string > user_input

    What is read in from the console at any point.
- std::vector< std::string > input_files

    A vector of input file names and directories given by user.
- std::string path

    Path to where input files are located.
- int count = 0

    Number of times a questing has been asked.
- int max = 3

    Maximum allowable recursions of a question.
- int option

    Current option choosen by the user.
- bool Path = false

    True if user gives path as an option.
- bool Files = false

    True if user gives input files as an option.
- bool MissingArg = true

    True if an input argument is missing; False if everything is ok.
- bool BasicUI = true

    True if using Basic UI; False if using Advanced UI.
- int argc

    Number of console arguments given on input.
- const char ∗ argv []

    Actual console arguments given at execution.

4.49.1 Detailed Description

Data structure holding the UI arguments.

C-Style object for interfacing with users request upon execution of the program. User input is stored in objects below and a series of booleans is used to determine how and what to execute.

4.49.2 Member Data Documentation

4.49.2.1 int UI_DATA::argc

Number of console arguments given on input.

4.49.2.2 const char∗ UI_DATA::argv[ ]

Actual console arguments given at execution.

4.49.2.3 bool UI_DATA::BasicUI = true

True if using Basic UI; False if using Advanced UI.

4.49.2.4 int UI_DATA::count = 0

Number of times a questing has been asked.

4.49.2.5 bool UI_DATA::Files = false

True if user gives input files as an option.

4.49.2.6 std::vector<std::string> UI_DATA::input_files

A vector of input file names and directories given by user.

4.49.2.7 int UI_DATA::max = 3

Maximum allowable recursions of a question.

4.49.2.8 bool UI_DATA::MissingArg = true

True if an input argument is missing; False if everything is ok.

4.49.2.9 int UI_DATA::option

Current option choosen by the user.

4.49.2.10 std::string UI_DATA::path

Path to where input files are located.

4.49.2.11 bool UI_DATA::Path = false

True if user gives path as an option.

4.49.2.12 std::vector<std::string> UI_DATA::user_input

What is read in from the console at any point.

4.49.2.13 ValueTypePair UI_DATA::value_type

Data pair for input, tells what the input is and it's type.

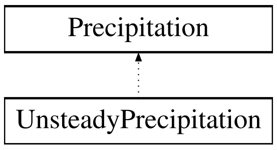
The documentation for this struct was generated from the following file:

- ui.h

4.50 UnsteadyPrecipitation Class Reference

```
#include <shark.h>
```

Inheritance diagram for UnsteadyPrecipitation:



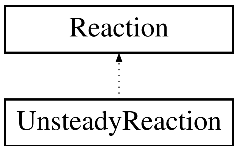
The documentation for this class was generated from the following file:

- shark.h

4.51 UnsteadyReaction Class Reference

```
#include <shark.h>
```

Inheritance diagram for UnsteadyReaction:



Public Member Functions

- UnsteadyReaction ()
- ∼UnsteadyReaction ()
- void Initialize_List (MasterSpeciesList &List)
- void Display_Info ()
- void Set_Species_Index (int i)
- void Set_Species_Index (std::string formula)
- void Set_Stoichiometric (int i, double v)
- void Set_Equilibrium (double v)
- void Set_Enthalpy (double H)
- void Set_Entropy (double S)
- void Set_EnthalpyANDEntropy (double H, double S)
- void Set_Energy (double G)
- void Set_InitialValue (double ic)
- void Set_MaximumValue (double max)
- void Set_Forward (double forward)
- void Set_Reverse (double reverse)
- void Set_ForwardRef (double Fref)
- void Set_ReverseRef (double Rref)
- void Set_ActivationEnergy (double E)
- void Set_Affinity (double b)
- void Set_TimeStep (double dt)
- void checkSpeciesEnergies ()
- void calculateEnergies ()
- void calculateEquilibrium (double T)
- void calculateRate (double T)
- bool haveEquilibrium ()
- bool haveRate ()
- int Get_Species_Index ()

- double Get_Stoichiometric (int i)
- double Get_Equilibrium ()
- double Get_Enthalpy ()
- double Get_Entropy ()
- double Get_Energy ()
- double Get_InitialValue ()
- double Get_MaximumValue ()
- double Get_Forward ()
- double Get_Reverse ()
- double Get_ForwardRef ()
- double Get_ReverseRef ()
- double Get_ActivationEnergy ()
- double Get_Affinity ()
- double Get_TimeStep ()
- double Eval_ReactionRate (const Matrix< double > &x, const Matrix< double > &gama)
- double Eval_Residual (const Matrix< double > &x_new, const Matrix< double > &x_old, const Matrix< double > &gama_new, const Matrix< double > &gama_old)
- double Eval_Residual (const Matrix< double > &x, const Matrix< double > &gama)
- double Eval_IC_Residual (const Matrix< double > &x)
- double Explicit_Eval (const Matrix< double > &x, const Matrix< double > &gama)

Protected Attributes

- double initial_value
- double max_value
- double forward_rate
- double reverse_rate
- double forward_ref_rate
- double reverse_ref_rate
- double activation_energy
- double temperature_affinity
- double time_step
- bool HaveForward
- bool HaveReverse
- bool HaveForRef
- bool HaveRevRef
- int species_index

Additional Inherited Members

4.51.1 Constructor & Destructor Documentation

4.51.1.1 UnsteadyReaction::UnsteadyReaction ( )

4.51.1.2 UnsteadyReaction::∼UnsteadyReaction ( )

4.51.2 Member Function Documentation

4.51.2.1 void UnsteadyReaction::calculateEnergies ( )

4.51.2.2 void UnsteadyReaction::calculateEquilibrium (double *T*)

4.51.2.3 void UnsteadyReaction::calculateRate (double *T*)

4.51.2.4 void UnsteadyReaction::checkSpeciesEnergies ( )

4.51.2.5 void UnsteadyReaction::Display_Info ( )

4.51.2.6 double UnsteadyReaction::Eval_IC_Residual (const Matrix< double > & *x*)

4.51.2.7 double UnsteadyReaction::Eval_ReactionRate (const Matrix< double > & *x*, const Matrix< double > & *gama*)

4.51.2.8 double UnsteadyReaction::Eval_Residual (const Matrix< double > & *x_new*, const Matrix< double > & *x_old*, const Matrix< double > & *gama_new*, const Matrix< double > & *gama_old*)

4.51.2.9 double UnsteadyReaction::Eval_Residual (const Matrix< double > & *x*, const Matrix< double > & *gama*)

4.51.2.10 double UnsteadyReaction::Explicit_Eval (const Matrix< double > & *x*, const Matrix< double > & *gama*)

4.51.2.11 double UnsteadyReaction::Get_ActivationEnergy ( )

4.51.2.12 double UnsteadyReaction::Get_Affinity ( )

4.51.2.13 double UnsteadyReaction::Get_Energy ( )

4.51.2.14 double UnsteadyReaction::Get_Enthalpy ( )

4.51.2.15 double UnsteadyReaction::Get_Entropy ( )

4.51.2.16 double UnsteadyReaction::Get_Equilibrium ( )

4.51.2.17 double UnsteadyReaction::Get_Forward ( )

4.51.2.18 double UnsteadyReaction::Get_ForwardRef ( )

4.51.2.19 double UnsteadyReaction::Get_InitialValue ( )

4.51.2.20 double UnsteadyReaction::Get_MaximumValue ( )

4.51.2.21 double UnsteadyReaction::Get_Reverse ( )

4.51.2.22 double UnsteadyReaction::Get_ReverseRef ( )

4.51.2.23 int UnsteadyReaction::Get_Species_Index ( )

4.51.2.24 double UnsteadyReaction::Get_Stoichiometric (int *i*)

4.51.2.25 double UnsteadyReaction::Get_TimeStep ( )

4.51.2.26 bool UnsteadyReaction::haveEquilibrium ( )

4.51.2.27 bool UnsteadyReaction::haveRate ( )

4.51.2.28 void UnsteadyReaction::Initialize_List (MasterSpeciesList & *List*)

4.51.2.29 void UnsteadyReaction::Set_ActivationEnergy (double *E*)

4.51.2.30 void UnsteadyReaction::Set_Affinity (double *b*)

4.51.2.31 void UnsteadyReaction::Set_Energy (double *G*)

4.51.2.32 void UnsteadyReaction::Set_Enthalpy (double *H*)

4.51.2.33 void UnsteadyReaction::Set_EnthalpyANDEntropy (double *H,* double *S*)

4.51.2.34 void UnsteadyReaction::Set_Entropy (double *S*)

4.51.2.35 void UnsteadyReaction::Set_Equilibrium (double *v*)

4.51.2.36 void UnsteadyReaction::Set_Forward (double *forward*)

4.51.2.37 void UnsteadyReaction::Set_ForwardRef (double *Fref*)

4.51.2.38 void UnsteadyReaction::Set_InitialValue (double *ic*)

4.51.2.39 void UnsteadyReaction::Set_MaximumValue (double *max*)

4.51.2.40 void UnsteadyReaction::Set_Reverse (double *reverse*)

4.51.2.41 void UnsteadyReaction::Set_ReverseRef (double *Rref*)

4.51.2.42 void UnsteadyReaction::Set_Species_Index (int *i*)

4.51.2.43 void UnsteadyReaction::Set_Species_Index (std::string *formula*)

4.51.2.44 void UnsteadyReaction::Set_Stoichiometric (int *i,* double *v*)

4.51.2.45 void UnsteadyReaction::Set_TimeStep (double *dt*)

4.51.3 Member Data Documentation

4.51.3.1 double UnsteadyReaction::activation_energy `[protected]`

4.51.3.2 double UnsteadyReaction::forward_rate `[protected]`

4.51.3.3 double UnsteadyReaction::forward_ref_rate `[protected]`

4.51.3.4 bool UnsteadyReaction::HaveForRef `[protected]`

4.51.3.5 bool UnsteadyReaction::HaveForward `[protected]`

4.51.3.6 bool UnsteadyReaction::HaveReverse `[protected]`

4.51.3.7 bool UnsteadyReaction::HaveRevRef `[protected]`

4.51.3.8 double UnsteadyReaction::initial_value `[protected]`

4.51.3.9 double UnsteadyReaction::max_value `[protected]`

4.51.3.10 double UnsteadyReaction::reverse_rate `[protected]`

4.51.3.11 double UnsteadyReaction::reverse_ref_rate `[protected]`

4.51.3.12 int UnsteadyReaction::species_index `[protected]`

4.51.3.13 double UnsteadyReaction::temperature_affinity `[protected]`

4.51.3.14 double UnsteadyReaction::time_step [protected]

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

- shark.h
- shark.cpp

4.52 ValueTypePair Class Reference

```
#include <yaml_wrapper.h>
```

Public Member Functions

- ValueTypePair ()
- ∼ValueTypePair ()
- ValueTypePair (const std::pair< std::string, int > &vt)
- ValueTypePair (std::string value, int type)
- ValueTypePair (const ValueTypePair &vt)
- ValueTypePair & operator= (const ValueTypePair &vt)
- void editValue (std::string value)
- void editPair (std::string value, int type)
- void findType ()
- void assertType (int type)
- void DisplayPair ()
- std::string getString ()
- bool getBool ()
- double getDouble ()
- int getInt ()
- std::string getValue ()
- int getType ()
- std::pair< std::string, int > & getPair ()

Private Attributes

- std::pair< std::string, int > Value_Type
- int type

4.52.1 Constructor & Destructor Documentation

4.52.1.1 ValueTypePair::ValueTypePair ( )

4.52.1.2 ValueTypePair::∼ValueTypePair ( )

4.52.1.3 ValueTypePair::ValueTypePair (const std::pair< std::string, int > & *vt*)

4.52.1.4 ValueTypePair::ValueTypePair (std::string *value,* int *type*)

4.52.1.5 ValueTypePair::ValueTypePair (const **ValueTypePair** & *vt*)

4.52.2 Member Function Documentation

4.52.2.1 void ValueTypePair::assertType (int *type*)

4.52.2.2 void ValueTypePair::DisplayPair ( )

4.52.2.3 void ValueTypePair::editPair (std::string *value,* int *type*)

4.52.2.4 void ValueTypePair::editValue (std::string *value*)

4.52.2.5 void ValueTypePair::findType ( )

4.52.2.6 bool ValueTypePair::getBool ( )

4.52.2.7 double ValueTypePair::getDouble ( )

4.52.2.8 int ValueTypePair::getInt ( )

4.52.2.9 std::pair< std::string, int > & ValueTypePair::getPair ( )

4.52.2.10 std::string ValueTypePair::getString ( )

4.52.2.11 int ValueTypePair::getType ( )

4.52.2.12 std::string ValueTypePair::getValue ( )

4.52.2.13 ValueTypePair & ValueTypePair::operator= (const ValueTypePair & *vt*)

4.52.3 Member Data Documentation

4.52.3.1 int ValueTypePair::type `[private]`

4.52.3.2 std::pair<std::string,int> ValueTypePair::Value_Type `[private]`

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

- yaml_wrapper.h
- yaml_wrapper.cpp

4.53 yaml_cpp_class Class Reference

```
#include <yaml_wrapper.h>
```

Public Member Functions

- yaml_cpp_class ()
- ∼yaml_cpp_class ()
- int setInputFile (const char ∗file)
- int readInputFile ()
- int cleanup ()
- int executeYamlRead (const char ∗file)
- YamlWrapper & getYamlWrapper ()
- void DisplayContents ()

Private Attributes

- YamlWrapper yaml_wrapper
- FILE ∗ input_file
- const char ∗ file_name
- yaml_parser_t token_parser
- yaml_token_t current_token
- yaml_token_t previous_token

4.53.1 Constructor & Destructor Documentation

4.53.1.1 yaml_cpp_class::yaml_cpp_class ( )

4.53.1.2 yaml_cpp_class::∼yaml_cpp_class ( )

4.53.2 Member Function Documentation

4.53.2.1 int yaml_cpp_class::cleanup ( )

4.53.2.2 void yaml_cpp_class::DisplayContents ( )

4.53.2.3 int yaml_cpp_class::executeYamlRead (const char ∗ *file*)

4.53.2.4 YamlWrapper & yaml_cpp_class::getYamlWrapper ( )

4.53.2.5 int yaml_cpp_class::readInputFile ( )

4.53.2.6 int yaml_cpp_class::setInputFile (const char ∗ *file*)

4.53.3 Member Data Documentation

4.53.3.1 yaml_token_t yaml_cpp_class::current_token `[private]`

4.53.3.2 const char∗ yaml_cpp_class::file_name `[private]`

4.53.3.3 FILE∗ yaml_cpp_class::input_file `[private]`

4.53.3.4 yaml_token_t yaml_cpp_class::previous_token `[private]`

4.53.3.5 yaml_parser_t yaml_cpp_class::token_parser `[private]`

4.53.3.6 YamlWrapper yaml_cpp_class::yaml_wrapper `[private]`

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

- yaml_wrapper.h
- yaml_wrapper.cpp

4.54 YamlWrapper Class Reference

```
#include <yaml_wrapper.h>
```

Public Member Functions

- YamlWrapper ()
- ∼YamlWrapper ()
- YamlWrapper (const YamlWrapper &yaml)
- YamlWrapper (std::string key, const Document &doc)
- YamlWrapper & operator= (const YamlWrapper &yaml)
- Document & operator() (const std::string key)
- Document operator() (const std::string key) const
- std::map< std::string, Document > & getDocMap ()
- Document & getDocument (std::string key)
- std::map< std::string,
  Document >::const_iterator end () const
- std::map< std::string,
  Document >::iterator end ()
- std::map< std::string,
  Document >::const_iterator begin () const
- std::map< std::string,
  Document >::iterator begin ()
- void clear ()
- void resetKeys ()
- void changeKey (std::string oldKey, std::string newKey)
- void revalidateAllKeys ()
- void DisplayContents ()
- void addDocKey (std::string key)
- void copyAnchor2Alias (std::string alias, Document &ref)
- int size ()
- Document & getAnchoredDoc (std::string alias)
- Document & getDocFromHeadAlias (std::string alias)
- Document & getDocFromSubAlias (std::string alias)

Private Attributes

- std::map< std::string, Document > Doc_Map

4.54.1 Constructor & Destructor Documentation

4.54.1.1 YamlWrapper::YamlWrapper ( )

4.54.1.2 YamlWrapper::∼YamlWrapper ( )

4.54.1.3 YamlWrapper::YamlWrapper (const YamlWrapper & *yaml*)

4.54.1.4 YamlWrapper::YamlWrapper (std::string *key,* const Document & *doc*)

4.54.2 Member Function Documentation

4.54.2.1 void YamlWrapper::addDocKey (std::string *key*)

4.54.2.2 std::map< std::string, Document >::const_iterator YamlWrapper::begin ( ) const

4.54.2.3 std::map< std::string, Document >::iterator YamlWrapper::begin ( )

4.54.2.4 void YamlWrapper::changeKey (std::string *oldKey,* std::string *newKey*)

4.54.2.5 void YamlWrapper::clear ( )

4.54.2.6 void YamlWrapper::copyAnchor2Alias (std::string *alias*, Document & *ref*)

4.54.2.7 void YamlWrapper::DisplayContents ( )

4.54.2.8 std::map< std::string, Document >::const_iterator YamlWrapper::end ( ) const

4.54.2.9 std::map< std::string, Document >::iterator YamlWrapper::end ( )

4.54.2.10 Document & YamlWrapper::getAnchoredDoc (std::string *alias*)

4.54.2.11 Document & YamlWrapper::getDocFromHeadAlias (std::string *alias*)

4.54.2.12 Document & YamlWrapper::getDocFromSubAlias (std::string *alias*)

4.54.2.13 std::map< std::string, Document > & YamlWrapper::getDocMap ( )

4.54.2.14 Document & YamlWrapper::getDocument (std::string *key*)

4.54.2.15 Document & YamlWrapper::operator() (const std::string *key*)

4.54.2.16 Document YamlWrapper::operator() (const std::string *key*) const

4.54.2.17 YamlWrapper & YamlWrapper::operator= (const YamlWrapper & *yaml*)

4.54.2.18 void YamlWrapper::resetKeys ( )

4.54.2.19 void YamlWrapper::revalidateAllKeys ( )

4.54.2.20 int YamlWrapper::size ( )

4.54.3 Member Data Documentation

4.54.3.1 std::map<std::string, Document> YamlWrapper::Doc_Map [private]

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

- yaml_wrapper.h
- yaml_wrapper.cpp

Chapter 5

File Documentation

5.1 dogfish.cpp File Reference

```
#include "dogfish.h"
```

Functions

- void print2file_species_header (FILE ∗Output, DOGFISH_DATA ∗dog_dat, int i)
- void print2file_DOGFISH_header (DOGFISH_DATA ∗dog_dat)
- void print2file_DOGFISH_result_old (DOGFISH_DATA ∗dog_dat)
- void print2file_DOGFISH_result_new (DOGFISH_DATA ∗dog_dat)
- double default_Retardation (int i, int l, const void ∗data)
- double default_IntraDiffusion (int i, int l, const void ∗data)
- double default_FilmMTCoeff (int i, const void ∗data)
- double default_SurfaceConcentration (int i, const void ∗data)
- int setup_DOGFISH_DATA (FILE ∗file, double(∗eval_R)(int i, int l, const void ∗user_data), double(∗eval_-DI)(int i, int l, const void ∗user_data), double(∗eval_kf)(int i, const void ∗user_data), double(∗eval_qs)(int i, const void ∗user_data), const void ∗user_data, DOGFISH_DATA ∗dog_dat)
- int DOGFISH_Executioner (DOGFISH_DATA ∗dog_dat)
- int set_DOGFISH_ICs (DOGFISH_DATA ∗dog_dat)
- int set_DOGFISH_timestep (DOGFISH_DATA ∗dog_dat)
- int DOGFISH_preprocesses (DOGFISH_DATA ∗dog_dat)
- int set_DOGFISH_params (const void ∗user_data)
- int DOGFISH_postprocesses (DOGFISH_DATA ∗dog_dat)
- int DOGFISH_reset (DOGFISH_DATA ∗dog_dat)
- int DOGFISH (DOGFISH_DATA ∗dog_dat)
- int DOGFISH_TESTS ()

5.1.1 Function Documentation

5.1.1.1 double default_FilmMTCoeff (int *i*, const void ∗ *data*)

5.1.1.2 double default_IntraDiffusion (int *i*, int *l*, const void ∗ *data*)

5.1.1.3 double default_Retardation (int *i*, int *l*, const void ∗ *data*)

5.1.1.4 double default_SurfaceConcentration (int *i*, const void ∗ *data*)

5.1.1.5 int DOGFISH (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.6 int DOGFISH_Executioner (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.7 int DOGFISH_postprocesses (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.8 int DOGFISH_preprocesses (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.9 int DOGFISH_reset (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.10 int DOGFISH_TESTS ( )

5.1.1.11 void print2file_DOGFISH_header (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.12 void print2file_DOGFISH_result_new (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.13 void print2file_DOGFISH_result_old (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.14 void print2file_species_header (FILE ∗ *Output,* DOGFISH_DATA ∗ *dog_dat,* int *i*)

5.1.1.15 int set_DOGFISH_ICs (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.16 int set_DOGFISH_params (const void ∗ *user_data*)

5.1.1.17 int set_DOGFISH_timestep (DOGFISH_DATA ∗ *dog_dat*)

5.1.1.18 int setup_DOGFISH_DATA (FILE ∗ *file,* double(∗)(int i, int l, const void ∗user_data) *eval_R,* double(∗)(int i, int l, const void ∗user_data) *eval_DI,* double(∗)(int i, const void ∗user_data) *eval_kf,* double(∗)(int i, const void ∗user_data) *eval_qs,* const void ∗ *user_data,* DOGFISH_DATA ∗ *dog_dat*)

5.2 dogfish.h File Reference

```
#include "finch.h"
#include "mola.h"
```

Classes

- struct DOGFISH_PARAM
- struct DOGFISH_DATA

Functions

- void print2file_species_header (FILE ∗Output, DOGFISH_DATA ∗dog_dat, int i)
- void print2file_DOGFISH_header (DOGFISH_DATA ∗dog_dat)
- void print2file_DOGFISH_result_old (DOGFISH_DATA ∗dog_dat)
- void print2file_DOGFISH_result_new (DOGFISH_DATA ∗dog_dat)
- double default_Retardation (int i, int l, const void ∗data)
- double default_IntraDiffusion (int i, int l, const void ∗data)
- double default_FilmMTCoeff (int i, const void ∗data)
- double default_SurfaceConcentration (int i, const void ∗data)
- int setup_DOGFISH_DATA (FILE ∗file, double(∗eval_R)(int i, int l, const void ∗user_data), double(∗eval_DI)(int i, int l, const void ∗user_data), double(∗eval_kf)(int i, const void ∗user_data), double(∗eval_qs)(int i, const void ∗user_data), const void ∗user_data, DOGFISH_DATA ∗dog_dat)

- int DOGFISH_Executioner (DOGFISH_DATA ∗dog_dat)
- int set_DOGFISH_ICs (DOGFISH_DATA ∗dog_dat)
- int set_DOGFISH_timestep (DOGFISH_DATA ∗dog_dat)
- int DOGFISH_preprocesses (DOGFISH_DATA ∗dog_dat)
- int set_DOGFISH_params (const void ∗user_data)
- int DOGFISH_postprocesses (DOGFISH_DATA ∗dog_dat)
- int DOGFISH_reset (DOGFISH_DATA ∗dog_dat)
- int DOGFISH (DOGFISH_DATA ∗dog_dat)
- int DOGFISH_TESTS ()

5.2.1 Function Documentation

5.2.1.1 double default_FilmMTCoeff (int *i,* const void ∗ *data*)

5.2.1.2 double default_IntraDiffusion (int *i,* int *l,* const void ∗ *data*)

5.2.1.3 double default_Retardation (int *i,* int *l,* const void ∗ *data*)

5.2.1.4 double default_SurfaceConcentration (int *i,* const void ∗ *data*)

5.2.1.5 int DOGFISH (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.6 int DOGFISH_Executioner (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.7 int DOGFISH_postprocesses (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.8 int DOGFISH_preprocesses (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.9 int DOGFISH_reset (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.10 int DOGFISH_TESTS ( )

5.2.1.11 void print2file_DOGFISH_header (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.12 void print2file_DOGFISH_result_new (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.13 void print2file_DOGFISH_result_old (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.14 void print2file_species_header (FILE ∗ *Output,* DOGFISH_DATA ∗ *dog_dat,* int *i*)

5.2.1.15 int set_DOGFISH_ICs (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.16 int set_DOGFISH_params (const void ∗ *user_data*)

5.2.1.17 int set_DOGFISH_timestep (DOGFISH_DATA ∗ *dog_dat*)

5.2.1.18 int setup_DOGFISH_DATA (FILE ∗ *file,* double(∗)(int i, int l, const void ∗user_data) *eval_R,* double(∗)(int i, int l, const void ∗user_data) *eval_DI,* double(∗)(int i, const void ∗user_data) *eval_kf,* double(∗)(int i, const void ∗user_data) *eval_qs,* const void ∗ *user_data,* DOGFISH_DATA ∗ *dog_dat*)

5.3 eel.cpp File Reference

```
#include "eel.h"
```

Functions

- int EEL_TESTS ()

5.3.1 Function Documentation

5.3.1.1 int EEL_TESTS ( )

5.4 eel.h File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include "error.h"
```

Classes

- class Atom
- class PeriodicTable

Functions

- int EEL_TESTS ()

5.4.1 Function Documentation

5.4.1.1 int EEL_TESTS ( )

5.5 egret.cpp File Reference

```
#include "egret.h"
```

Functions

- int initialize_data (int N, MIXED_GAS ∗gas_dat)
- int set_variables (double PT, double T, double us, double L, std::vector< double > &y, MIXED_GAS ∗gas_dat)
- int calculate_properties (MIXED_GAS ∗gas_dat)
- int EGRET_TESTS ()

5.5.1 Function Documentation

5.5.1.1 int calculate_properties (**MIXED_GAS** ∗ *gas_dat*)

5.5.1.2 int EGRET_TESTS ( )

5.5.1.3 int initialize_data (int *N*, MIXED_GAS ∗ *gas_dat*)

5.5.1.4 int set_variables (double *PT*, double *T*, double *us*, double *L*, std::vector< double > & *y*, MIXED_GAS ∗ *gas_dat*)

5.6 egret.h File Reference

```
#include "macaw.h"
```

Classes

- struct PURE_GAS
- struct MIXED_GAS

Macros

- #define Rstd 8.3144621
- #define RE3 8.3144621E+3
- #define Po 100.0
- #define Cstd(p, T) ((p)/(Rstd∗T))
- #define CE3(p, T) ((p)/(RE3∗T))
- #define Pstd(c, T) ((c)∗Rstd∗T)
- #define PE3(c, T) ((c)∗RE3∗T)
- #define Nu(mu, rho) ((mu)/(rho))
- #define PSI(T) (0.873143 + (0.000072375∗T))
- #define Dp_ij(Dij, PT) ((PT∗Dij)/Po)
- #define D_ij(MWi, MWj, rhoi, rhoj, mui, muj) ((4.0 / sqrt(2.0)) ∗ pow(((1/MWi)+(1/MWj)),0.5)) / pow((pow((pow((rhoi/(1.385∗mui)),2.0)/MWi),0.25)+ pow((pow((rhoj/(1.385∗muj)),2.0)/MWj),0.25)),2.0)
- #define Mu(muo, To, C, T) (muo ∗ ((To + C)/(T + C)) ∗ pow((T/To),1.5))
- #define D_ii(rhoi, mui) (1.385∗mui/rhoi)
- #define ReNum(u, L, nu) (u∗L/nu)
- #define ScNum(nu, D) (nu/D)
- #define FilmMTCoeff(D, L, Re, Sc) ((D/L)∗(2.0 + (1.1∗pow(Re,0.6)∗pow(Sc,0.3))))

Functions

- int initialize_data (int N, MIXED_GAS ∗gas_dat)
- int set_variables (double PT, double T, double us, double L, std::vector< double > &y, MIXED_GAS ∗gas_dat)
- int calculate_properties (MIXED_GAS ∗gas_dat)
- int EGRET_TESTS ()

5.6.1 Macro Definition Documentation

5.6.1.1 #define CE3(*p*, *T*) ((p)/(RE3∗T))

5.6.1.2 #define Cstd(*p*, *T*) ((p)/(Rstd∗T))

5.6.1.3 #define D_ii(*rhoi*, *mui*) (1.385∗mui/rhoi)

5.6.1.4 #define D_ij(*MWi, MWj, rhoi, rhoj, mui, muj*) ((4.0 / sqrt(2.0)) ∗ pow(((1/MWi)+(1/MWj)),0.5)) / pow((pow((pow((rhoi/(1.385∗mui)),2.0)/MWi),0.25)+ pow((pow((rhoj/(1.385∗muj)),2.0)/MWj),0.25)),2.0)

5.6.1.5 #define Dp_ij(*Dij, PT*) ((PT∗Dij)/Po)

5.6.1.6 #define FilmMTCoeff(*D, L, Re, Sc*) ((D/L)∗(2.0 + (1.1∗pow(Re,0.6)∗pow(Sc,0.3))))

5.6.1.7 #define Mu(*muo, To, C, T*) (muo ∗ ((To + C)/(T + C)) ∗ pow((T/To),1.5))

5.6.1.8 #define Nu(*mu, rho*) ((mu)/(rho))

5.6.1.9 #define PE3(*c, T*) ((c)∗RE3∗T)

5.6.1.10 #define Po 100.0

5.6.1.11 #define PSI(*T*) (0.873143 + (0.000072375∗T))

5.6.1.12 #define Pstd(*c, T*) ((c)∗Rstd∗T)

5.6.1.13 #define RE3 8.3144621E+3

5.6.1.14 #define ReNum(*u, L, nu*) (u∗L/nu)

5.6.1.15 #define Rstd 8.3144621

5.6.1.16 #define ScNum(*nu, D*) (nu/D)

5.6.2 Function Documentation

5.6.2.1 int calculate_properties (MIXED_GAS ∗ *gas_dat*)

5.6.2.2 int EGRET_TESTS ( )

5.6.2.3 int initialize_data (int *N*, MIXED_GAS ∗ *gas_dat*)

5.6.2.4 int set_variables (double *PT*, double *T*, double *us*, double *L*, std::vector< double > & *y*, MIXED_GAS ∗ *gas_dat*)

5.7 error.cpp File Reference

```
#include "error.h"
```

Functions

- void error (int flag)

5.7.1 Function Documentation

5.7.1.1 void error (int *flag*)

5.8 error.h File Reference

```
#include <iostream>
```

Macros

- #define mError(i)

Enumerations

- enum error_type {
  generic_error, file_dne, indexing_error, magpie_reverse_error,
  simulation_fail, invalid_components, invalid_boolean, invalid_molefraction,
  invalid_gas_sum, invalid_solid_sum, scenario_fail, out_of_bounds,
  non_square_matrix, dim_mis_match, empty_matrix, opt_no_support,
  invalid_fraction, ortho_check_fail, unstable_matrix, no_diffusion,
  negative_mass, negative_time, matvec_mis_match, arg_matrix_same,
  singular_matrix, matrix_too_small, invalid_size, nullptr_func,
  invalid_norm, vector_out_of_bounds, zero_vector, tensor_out_of_bounds,
  non_real_edge, nullptr_error, invalid_atom, invalid_proton,
  invalid_neutron, invalid_electron, invalid_valence, string_parse_error,
  unregistered_name, rxn_rate_error, invalid_species, duplicate_variable,
  missing_information, invalid_type, key_not_found, anchor_alias_dne,
  initial_error, not_a_token, read_error, invalid_console_input }

Functions

- void error (int flag)

5.8.1 Macro Definition Documentation

5.8.1.1 #define mError(*i*)

Value:

```
{error(i);          \
std::cout << "Source: " << __FILE__ << "\nLine: " << __LINE__ << std::endl;}
```

5.8.2 Enumeration Type Documentation

5.8.2.1 enum error_type

Enumerator

generic_error

file_dne

indexing_error

magpie_reverse_error

simulation_fail

invalid_components

invalid_boolean

invalid_molefraction

invalid_gas_sum

invalid_solid_sum

scenario_fail

out_of_bounds

non_square_matrix

dim_mis_match

empty_matrix

opt_no_support

invalid_fraction

ortho_check_fail

unstable_matrix

no_diffusion

negative_mass

negative_time

matvec_mis_match

arg_matrix_same

singular_matrix

matrix_too_small

invalid_size

nullptr_func

invalid_norm

vector_out_of_bounds

zero_vector

tensor_out_of_bounds

non_real_edge

nullptr_error

invalid_atom

invalid_proton

invalid_neutron

invalid_electron

invalid_valence

string_parse_error

unregistered_name

rxn_rate_error

invalid_species

duplicate_variable

missing_information

invalid_type

key_not_found

anchor_alias_dne

initial_error

not_a_token

read_error

invalid_console_input

5.8.3 Function Documentation

5.8.3.1 void error (int *flag*)

5.9 finch.cpp File Reference

```
#include "finch.h"
```

Functions

- double max (std::vector< double > &values)
- double min (std::vector< double > &values)
- double minmod (std::vector< double > &values)
- int uTotal (FINCH_DATA ∗dat)
- int uAverage (FINCH_DATA ∗dat)
- int check_Mass (FINCH_DATA ∗dat)
- int l_direct (FINCH_DATA ∗dat)
- int lark_picard_step (const Matrix< double > &x, Matrix< double > &G, const void ∗data)
- int nl_picard (FINCH_DATA ∗dat)
- int setup_FINCH_DATA (int(∗user_callroutine)(const void ∗user_data), int(∗user_setic)(const void ∗user_data), int(∗user_timestep)(const void ∗user_data), int(∗user_preprocess)(const void ∗user_data), int(∗user_solve)(const void ∗user_data), int(∗user_setparams)(const void ∗user_data), int(∗user_discretize)(const void ∗user_data), int(∗user_bcs)(const void ∗user_data), int(∗user_res)(const Matrix< double > &x, Matrix< double > &res, const void ∗user_data), int(∗user_precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗user_data), int(∗user_postprocess)(const void ∗user_data), int(∗user_reset)(const void ∗user_data), FINCH_DATA ∗dat, const void ∗param_data)
- void print2file_dim_header (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_time_header (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_result_old (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_result_new (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_newline (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_tab (FILE ∗Output, FINCH_DATA ∗dat)
- int default_execution (const void ∗user_data)
- int default_ic (const void ∗user_data)
- int default_timestep (const void ∗user_data)
- int default_preprocess (const void ∗user_data)
- int default_solve (const void ∗user_data)
- int default_params (const void ∗user_data)
- int minmod_discretization (const void ∗user_data)
- int vanAlbada_discretization (const void ∗user_data)
- int ospre_discretization (const void ∗user_data)
- int default_bcs (const void ∗user_data)
- int default_res (const Matrix< double > &x, Matrix< double > &res, const void ∗user_data)
- int default_precon (const Matrix< double > &b, Matrix< double > &p, const void ∗user_data)
- int default_postprocess (const void ∗user_data)
- int default_reset (const void ∗user_data)
- int buckley_leverett_ic (const void ∗user_data)
- int buckley_leverett_params (const void ∗user_data)
- int burgers_ic (const void ∗user_data)
- int burgers_params (const void ∗user_data)
- int burgers_bcs (const void ∗user_data)
- int FINCH_TESTS ()

5.9.1 Function Documentation

5.9.1.1 int buckley_leverett_ic (const void ∗ *user_data*)

5.9.1.2 int buckley_leverett_params (const void ∗ *user_data*)

5.9.1.3 int burgers_bcs (const void ∗ *user_data*)

5.9.1.4 int burgers_ic (const void ∗ *user_data*)

5.9.1.5 int burgers_params (const void ∗ *user_data*)

5.9.1.6 int check_Mass (FINCH_DATA ∗ *dat*)

5.9.1.7 int default_bcs (const void ∗ *user_data*)

5.9.1.8 int default_execution (const void ∗ *user_data*)

5.9.1.9 int default_ic (const void ∗ *user_data*)

5.9.1.10 int default_params (const void ∗ *user_data*)

5.9.1.11 int default_postprocess (const void ∗ *user_data*)

5.9.1.12 int default_precon (const Matrix< double > & *b*, Matrix< double > & *p*, const void ∗ *user_data*)

5.9.1.13 int default_preprocess (const void ∗ *user_data*)

5.9.1.14 int default_res (const Matrix< double > & *x*, Matrix< double > & *res*, const void ∗ *user_data*)

5.9.1.15 int default_reset (const void ∗ *user_data*)

5.9.1.16 int default_solve (const void ∗ *user_data*)

5.9.1.17 int default_timestep (const void ∗ *user_data*)

5.9.1.18 int FINCH_TESTS ( )

5.9.1.19 int l_direct (FINCH_DATA ∗ *dat*)

5.9.1.20 int lark_picard_step (const Matrix< double > & *x*, Matrix< double > & *G*, const void ∗ *data*)

5.9.1.21 double max (std::vector< double > & *values*)

5.9.1.22 double min (std::vector< double > & *values*)

5.9.1.23 double minmod (std::vector< double > & *values*)

5.9.1.24 int minmod_discretization (const void ∗ *user_data*)

5.9.1.25 int nl_picard (FINCH_DATA ∗ *dat*)

5.9.1.26 int ospre_discretization (const void ∗ *user_data*)

5.9.1.27 void print2file_dim_header (FILE ∗ *Output*, FINCH_DATA ∗ *dat*)

5.9.1.28 void print2file_newline (FILE ∗ *Output*, FINCH_DATA ∗ *dat*)

5.9.1.29 void print2file_result_new (FILE ∗ *Output*, FINCH_DATA ∗ *dat*)

5.9.1.30 void print2file_result_old (FILE ∗ *Output*, FINCH_DATA ∗ *dat*)

5.9.1.31 void print2file_tab (FILE ∗ *Output*, FINCH_DATA ∗ *dat*)

5.9.1.32 void print2file_time_header (FILE ∗ *Output*, FINCH_DATA ∗ *dat*)

5.9.1.33 int setup_FINCH_DATA (int(∗)(const void ∗user_data) *user_callroutine,* int(∗)(const void ∗user_data) *user_setic,* int(∗)(const void ∗user_data) *user_timestep,* int(∗)(const void ∗user_data) *user_preprocess,* int(∗)(const void ∗user_data) *user_solve,* int(∗)(const void ∗user_data) *user_setparams,* int(∗)(const void ∗user_data) *user_discretize,* int(∗)(const void ∗user_data) *user_bcs,* int(∗)(const Matrix< double > &x, Matrix< double > &res, const void ∗user_data) *user_res,* int(∗)(const Matrix< double > &b, Matrix< double > &p, const void ∗user_data) *user_precon,* int(∗)(const void ∗user_data) *user_postprocess,* int(∗)(const void ∗user_data) *user_reset,* FINCH_DATA ∗ *dat,* const void ∗ *param_data*)

5.9.1.34 int uAverage (FINCH_DATA ∗ *dat*)

5.9.1.35 int uTotal (FINCH_DATA ∗ *dat*)

5.9.1.36 int vanAlbada_discretization (const void ∗ *user_data*)

5.10 finch.h File Reference

```
#include "macaw.h"
#include "lark.h"
```

Classes

- struct FINCH_DATA

Macros

- #define FINCH_Picard 0
- #define LARK_Picard 1
- #define LARK_PJFNK 2
- #define Cartesian 0
- #define Cylindrical 1
- #define Spherical 2

Functions

- double max (std::vector< double > &values)
- double min (std::vector< double > &values)
- double minmod (std::vector< double > &values)
- int uTotal (FINCH_DATA ∗dat)
- int uAverage (FINCH_DATA ∗dat)
- int check_Mass (FINCH_DATA ∗dat)
- int l_direct (FINCH_DATA ∗dat)
- int lark_picard_step (const Matrix< double > &x, Matrix< double > &G, const void ∗data)
- int nl_picard (FINCH_DATA ∗dat)
- int setup_FINCH_DATA (int(∗user_callroutine)(const void ∗user_data), int(∗user_setic)(const void ∗user_-data), int(∗user_timestep)(const void ∗user_data), int(∗user_preprocess)(const void ∗user_data), int(∗user_-solve)(const void ∗user_data), int(∗user_setparams)(const void ∗user_data), int(∗user_discretize)(const void ∗user_data), int(∗user_bcs)(const void ∗user_data), int(∗user_res)(const Matrix< double > &x, Matrix< double > &res, const void ∗user_data), int(∗user_precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗user_data), int(∗user_postprocess)(const void ∗user_data), int(∗user_reset)(const void ∗user_-data), FINCH_DATA ∗dat, const void ∗param_data)
- void print2file_dim_header (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_time_header (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_result_old (FILE ∗Output, FINCH_DATA ∗dat)

- void print2file_result_new (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_newline (FILE ∗Output, FINCH_DATA ∗dat)
- void print2file_tab (FILE ∗Output, FINCH_DATA ∗dat)
- int default_execution (const void ∗user_data)
- int default_ic (const void ∗user_data)
- int default_timestep (const void ∗user_data)
- int default_preprocess (const void ∗user_data)
- int default_solve (const void ∗user_data)
- int default_params (const void ∗user_data)
- int minmod_discretization (const void ∗user_data)
- int vanAlbada_discretization (const void ∗user_data)
- int ospre_discretization (const void ∗user_data)
- int default_bcs (const void ∗user_data)
- int default_res (const Matrix< double > &x, Matrix< double > &res, const void ∗user_data)
- int default_precon (const Matrix< double > &b, Matrix< double > &p, const void ∗user_data)
- int default_postprocess (const void ∗user_data)
- int default_reset (const void ∗user_data)
- int buckley_leverett_ic (const void ∗user_data)
- int buckley_leverett_params (const void ∗user_data)
- int burgers_ic (const void ∗user_data)
- int burgers_params (const void ∗user_data)
- int burgers_bcs (const void ∗user_data)
- int FINCH_TESTS ()

5.10.1 Macro Definition Documentation

5.10.1.1 #define Cartesian 0

5.10.1.2 #define Cylindrical 1

5.10.1.3 #define FINCH_Picard 0

5.10.1.4 #define LARK_Picard 1

5.10.1.5 #define LARK_PJFNK 2

5.10.1.6 #define Spherical 2

5.10.2 Function Documentation

5.10.2.1 int buckley_leverett_ic (const void ∗ *user_data*)

5.10.2.2 int buckley_leverett_params (const void ∗ *user_data*)

5.10.2.3 int burgers_bcs (const void ∗ *user_data*)

5.10.2.4 int burgers_ic (const void ∗ *user_data*)

5.10.2.5 int burgers_params (const void ∗ *user_data*)

5.10.2.6 int check_Mass (FINCH_DATA ∗ *dat*)

5.10.2.7 int default_bcs (const void ∗ *user_data*)

5.10.2.8 int default_execution (const void $*$ *user_data*)

5.10.2.9 int default_ic (const void $*$ *user_data*)

5.10.2.10 int default_params (const void $*$ *user_data*)

5.10.2.11 int default_postprocess (const void $*$ *user_data*)

5.10.2.12 int default_precon (const Matrix$<$ double $>$ & *b,* Matrix$<$ double $>$ & *p,* const void $*$ *user_data*)

5.10.2.13 int default_preprocess (const void $*$ *user_data*)

5.10.2.14 int default_res (const Matrix$<$ double $>$ & *x,* Matrix$<$ double $>$ & *res,* const void $*$ *user_data*)

5.10.2.15 int default_reset (const void $*$ *user_data*)

5.10.2.16 int default_solve (const void $*$ *user_data*)

5.10.2.17 int default_timestep (const void $*$ *user_data*)

5.10.2.18 int FINCH_TESTS ( )

5.10.2.19 int l_direct (FINCH_DATA $*$ *dat*)

5.10.2.20 int lark_picard_step (const Matrix$<$ double $>$ & *x,* Matrix$<$ double $>$ & *G,* const void $*$ *data*)

5.10.2.21 double max (std::vector$<$ double $>$ & *values*)

5.10.2.22 double min (std::vector$<$ double $>$ & *values*)

5.10.2.23 double minmod (std::vector$<$ double $>$ & *values*)

5.10.2.24 int minmod_discretization (const void $*$ *user_data*)

5.10.2.25 int nl_picard (FINCH_DATA $*$ *dat*)

5.10.2.26 int ospre_discretization (const void $*$ *user_data*)

5.10.2.27 void print2file_dim_header (FILE $*$ *Output,* FINCH_DATA $*$ *dat*)

5.10.2.28 void print2file_newline (FILE $*$ *Output,* FINCH_DATA $*$ *dat*)

5.10.2.29 void print2file_result_new (FILE $*$ *Output,* FINCH_DATA $*$ *dat*)

5.10.2.30 void print2file_result_old (FILE $*$ *Output,* FINCH_DATA $*$ *dat*)

5.10.2.31 void print2file_tab (FILE $*$ *Output,* FINCH_DATA $*$ *dat*)

5.10.2.32 void print2file_time_header (FILE $*$ *Output,* FINCH_DATA $*$ *dat*)

5.10.2.33 int setup_FINCH_DATA (int(∗)(const void ∗user_data) *user_callroutine*, int(∗)(const void ∗user_data) *user_setic*, int(∗)(const void ∗user_data) *user_timestep*, int(∗)(const void ∗user_data) *user_preprocess*, int(∗)(const void ∗user_data) *user_solve*, int(∗)(const void ∗user_data) *user_setparams*, int(∗)(const void ∗user_data) *user_discretize*, int(∗)(const void ∗user_data) *user_bcs*, int(∗)(const **Matrix**< double > &x, **Matrix**< double > &res, const void ∗user_data) *user_res*, int(∗)(const **Matrix**< double > &b, **Matrix**< double > &p, const void ∗user_data) *user_precon*, int(∗)(const void ∗user_data) *user_postprocess*, int(∗)(const void ∗user_data) *user_reset*, **FINCH_DATA** ∗ *dat*, const void ∗ *param_data*)

5.10.2.34 int uAverage (**FINCH_DATA** ∗ *dat*)

5.10.2.35 int uTotal (**FINCH_DATA** ∗ *dat*)

5.10.2.36 int vanAlbada_discretization (const void ∗ *user_data*)

5.11 flock.h File Reference

```
#include "macaw.h"
#include "egret.h"
#include "finch.h"
#include "lark.h"
#include "skua.h"
#include "scopsowl.h"
#include "gsta_opt.h"
#include "magpie.h"
#include "skua_opt.h"
#include "scopsowl_opt.h"
#include "yaml_wrapper.h"
```

5.12 gsta_opt.cpp File Reference

```
#include "gsta_opt.h"
```

Functions

- int roundIt (double d)
- int twoFifths (int m)
- int orderMag (double x)
- int minValue (std::vector< int > array)
- int minIndex (std::vector< double > array)
- int avgPar (std::vector< int > array)
- double avgValue (std::vector< double > array)
- double weightedAvg (double ∗enorm, double ∗x, int n)
- double rSq (double ∗x, double ∗y, double slope, double vint, int m_dat)
- bool isSmooth (double ∗par, void ∗data)
- void orthoLinReg (double ∗x, double ∗y, double ∗par, int m_dat, int n_par)
- void eduGuess (double ∗P, double ∗q, double ∗par, int k, int m_dat, void ∗data)
- double gstaFunc (double p, const double ∗K, double qmax, int n_par)
- double gstaObjFunc (double ∗t, double ∗y, double ∗par, int m_dat, void ∗data)
- void eval_GSTA (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int gsta_optimize (const char ∗fileName)

5.12.1 Function Documentation

5.12.1.1 int avgPar (std::vector< int > *array*)

5.12.1.2 double avgValue (std::vector< double > *array*)

5.12.1.3 void eduGuess (double ∗ *P,* double ∗ *q,* double ∗ *par,* int *k,* int *m_dat,* void ∗ *data*)

5.12.1.4 void eval_GSTA (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.12.1.5 int gsta_optimize (const char ∗ *fileName*)

5.12.1.6 double gstaFunc (double *p,* const double ∗ *K,* double *qmax,* int *n_par*)

5.12.1.7 double gstaObjFunc (double ∗ *t,* double ∗ *y,* double ∗ *par,* int *m_dat,* void ∗ *data*)

5.12.1.8 bool isSmooth (double ∗ *par,* void ∗ *data*)

5.12.1.9 int minIndex (std::vector< double > *array*)

5.12.1.10 int minValue (std::vector< int > *array*)

5.12.1.11 int orderMag (double *x*)

5.12.1.12 void orthoLinReg (double ∗ *x,* double ∗ *y,* double ∗ *par,* int *m_dat,* int *n_par*)

5.12.1.13 int roundIt (double *d*)

5.12.1.14 double rSq (double ∗ *x,* double ∗ *y,* double *slope,* double *vint,* int *m_dat*)

5.12.1.15 int twoFifths (int *m*)

5.12.1.16 double weightedAvg (double ∗ *enorm,* double ∗ *x,* int *n*)

5.13 gsta_opt.h File Reference

```
#include "lmcurve.h"
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include "error.h"
```

Classes

- struct GSTA_OPT_DATA

Macros

- #define Po 100.0
- #define R 8.3144621
- #define Na 6.0221413E+23

Functions

- void error ()
- int roundIt (double d)
- int twoFifths (int m)
- int orderMag (double x)
- int minValue (std::vector< int > array)
- int minIndex (std::vector< double > array)
- int avgPar (std::vector< int > array)
- double avgValue (std::vector< double > array)
- double weightedAvg (double ∗enorm, double ∗x, int n)
- double rSq (double ∗x, double ∗y, double slope, double vint, int m_dat)
- bool isSmooth (double ∗par, void ∗data)
- void orthoLinReg (double ∗x, double ∗y, double ∗par, int m_dat, int n_par)
- void eduGuess (double ∗P, double ∗q, double ∗par, int k, int m_dat, void ∗data)
- double gstaFunc (double p, const double ∗K, double qmax, int n_par)
- double gstaObjFunc (double ∗t, double ∗y, double ∗par, int m_dat, void ∗data)
- void eval_GSTA (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int gsta_optimize (const char ∗fileName)

5.13.1 Macro Definition Documentation

5.13.1.1 #define Na 6.0221413E+23

5.13.1.2 #define Po 100.0

5.13.1.3 #define R 8.3144621

5.13.2 Function Documentation

5.13.2.1 int avgPar (std::vector< int > *array*)

5.13.2.2 double avgValue (std::vector< double > *array*)

5.13.2.3 void eduGuess (double ∗ *P,* double ∗ *q,* double ∗ *par,* int *k,* int *m_dat,* void ∗ *data*)

5.13.2.4 void error ( )

5.13.2.5 void eval_GSTA (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.13.2.6 int gsta_optimize (const char ∗ *fileName*)

5.13.2.7 double gstaFunc (double *p,* const double ∗ *K,* double *qmax,* int *n_par*)

5.13.2.8 double gstaObjFunc (double ∗ *t,* double ∗ *y,* double ∗ *par,* int *m_dat,* void ∗ *data*)

5.13.2.9 bool isSmooth (double ∗ *par,* void ∗ *data*)

5.13.2.10 int minIndex (std::vector< double > *array*)

5.13.2.11 int minValue (std::vector< int > *array*)

5.13.2.12 int orderMag (double *x*)

5.13.2.13 void orthoLinReg (double ∗ *x*, double ∗ *y*, double ∗ *par*, int *m_dat*, int *n_par*)

5.13.2.14 int roundIt (double *d*)

5.13.2.15 double rSq (double ∗ *x*, double ∗ *y*, double *slope*, double *vint*, int *m_dat*)

5.13.2.16 int twoFifths (int *m*)

5.13.2.17 double weightedAvg (double ∗ *enorm*, double ∗ *x*, int *n*)

5.14 lark.cpp File Reference

Linear Algebra Residual Kernels.

```
#include "lark.h"
```

Functions

- int update_arnoldi_solution (Matrix< double > &x, Matrix< double > &x0, ARNOLDI_DATA ∗arnoldi_dat)
- int arnoldi (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗data), Matrix< double > &r0, ARNOLDI_DATA ∗arnoldi_dat, const void ∗matvec_data, const void ∗precon_data)
- int gmresLeftPreconditioned (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &P, const void ∗data), Matrix< double > &b, GMRESLP_DATA ∗gmreslp_dat, const void ∗matvec_data, const void ∗precon_data)
- int fom (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &P, const void ∗data), Matrix< double > &b, GMRESLP_DATA ∗gmreslp_dat, const void ∗matvec_data, const void ∗precon_data)
- int gmresRightPreconditioned (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗data), Matrix< double > &b, GMRESRP_DATA ∗gmresrp_dat, const void ∗matvec_data, const void ∗precon_data)
- int pcg (int(∗matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &z, const void ∗data), Matrix< double > &b, PCG_DATA ∗pcg_dat, const void ∗matvec_data, const void ∗precon_data)
- int bicgstab (int(∗matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &z, const void ∗data), Matrix< double > &b, BiCGSTAB_DATA ∗bicg_dat, const void ∗matvec_data, const void ∗precon_data)
- int cgs (int(∗matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &z, const void ∗data), Matrix< double > &b, CGS_DATA ∗cgs_dat, const void ∗matvec_data, const void ∗precon_data)
- int operatorTranspose (int(∗matvec)(const Matrix< double > &v, Matrix< double > &Av, const void ∗data), Matrix< double > &r, Matrix< double > &u, OPTRANS_DATA ∗transpose_dat, const void ∗matvec_data)
- int gcr (int(∗matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &Mr, const void ∗data), Matrix< double > &b, GCR_DATA ∗gcr_dat, const void ∗matvec_data, const void ∗precon_data)
- int gmresPreconditioner (const Matrix< double > &r, Matrix< double > &Mr, const void ∗data)
- int gmresr (int(∗matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗data), int(∗terminal_precon)(const Matrix< double > &r, Matrix< double > &Mr, const void ∗data), Matrix< double > &b, GMRESR_DATA ∗gmresr_dat, const void ∗matvec_data, const void ∗term_precon_data)

- int picard (int(∗res)(const Matrix< double > &x, Matrix< double > &r, const void ∗data), int(∗evalx)(const Matrix< double > &x0, Matrix< double > &x, const void ∗data), Matrix< double > &x, PICARD_DATA ∗picard_dat, const void ∗res_data, const void ∗evalx_data)
- int jacvec (const Matrix< double > &v, Matrix< double > &Jv, const void ∗data)
- int backtrackLineSearch (int(∗feval)(const Matrix< double > &x, Matrix< double > &F, const void ∗data), Matrix< double > &Fkp1, Matrix< double > &xkp1, Matrix< double > &pk, double normFk, BACKTRACK-_DATA ∗backtrack_dat, const void ∗feval_data)
- int pjfnk (int(∗res)(const Matrix< double > &x, Matrix< double > &F, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &p, const void ∗data), Matrix< double > &x, PJFNK_DATA ∗pjfnk-_dat, const void ∗res_data, const void ∗precon_data)
- int NumericalJacobian (int(∗Func)(const Matrix< double > &x, Matrix< double > &F, const void ∗user_-data), const Matrix< double > &x, Matrix< double > &J, int Nx, int Nf, NUM_JAC_DATA ∗jac_dat, const void ∗user_data)
- int LARK_TESTS ()

5.14.1 Detailed Description

Linear Algebra Residual Kernels. lark.h

Author

Austin Ladshaw

Version

0.0 beta

Date

10/14/2014

Copyright

5.14.2 Function Documentation

5.14.2.1 int arnoldi (int(∗)(const Matrix< double > &v, Matrix< double > &w, const void ∗data) *matvec,* int(∗)(const Matrix< double > &b, Matrix< double > &p, const void ∗data) *precon,* Matrix< double > & *r0,* ARNOLDI_DATA ∗ *arnoldi_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.2 int backtrackLineSearch (int(∗)(const Matrix< double > &x, Matrix< double > &F, const void ∗data) *feval,* Matrix< double > & *Fkp1,* Matrix< double > & *xkp1,* Matrix< double > & *pk,* double *normFk,* BACKTRACK_DATA ∗ *backtrack_dat,* const void ∗ *feval_data*)

5.14.2.3 int bicgstab (int(∗)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &z, const void ∗data) *precon,* Matrix< double > & *b,* BiCGSTAB_DATA ∗ *bicg_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.4 int cgs (int(∗)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data) *matvec,* int(∗)(const Matrix< double > &r, Matrix< double > &z, const void ∗data) *precon,* Matrix< double > & *b,* CGS_DATA ∗ *cgs_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.5 int fom (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &P, const void ∗data) *precon,* **Matrix**< double > & *b,* **GMRESLP_DATA** ∗ *gmreslp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.6 int gcr (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &Ax, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &Mr, const void ∗data) *precon,* **Matrix**< double > & *b,* **GCR_DATA** ∗ *gcr_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.7 int gmresLeftPreconditioned (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &P, const void ∗data) *precon,* **Matrix**< double > & *b,* **GMRESLP_DATA** ∗ *gmreslp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.8 int gmresPreconditioner (const **Matrix**< double > & *r,* **Matrix**< double > & *Mr,* const void ∗ *data*)

5.14.2.9 int gmresr (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &Ax, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &Mr, const void ∗data) *terminal_precon,* **Matrix**< double > & *b,* **GMRESR_DATA** ∗ *gmresr_dat,* const void ∗ *matvec_data,* const void ∗ *term_precon_data*)

5.14.2.10 int gmresRightPreconditioned (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *b,* **GMRESRP_DATA** ∗ *gmresrp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.11 int jacvec (const **Matrix**< double > & *v,* **Matrix**< double > & *Jv,* const void ∗ *data*)

5.14.2.12 int LARK_TESTS ( )

5.14.2.13 int NumericalJacobian (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &F, const void ∗user_data) *Func,* const **Matrix**< double > & *x,* **Matrix**< double > & *J,* int *Nx,* int *Nf,* **NUM_JAC_DATA** ∗ *jac_dat,* const void ∗ *user_data*)

5.14.2.14 int operatorTranspose (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &Av, const void ∗data) *matvec,* **Matrix**< double > & *r,* **Matrix**< double > & *u,* **OPTRANS_DATA** ∗ *transpose_dat,* const void ∗ *matvec_data*)

5.14.2.15 int pcg (int(∗)(const **Matrix**< double > &p, **Matrix**< double > &Ap, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &z, const void ∗data) *precon,* **Matrix**< double > & *b,* **PCG_DATA** ∗ *pcg_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.14.2.16 int picard (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &r, const void ∗data) *res,* int(∗)(const **Matrix**< double > &x0, **Matrix**< double > &x, const void ∗data) *evalx,* **Matrix**< double > & *x,* **PICARD_DATA** ∗ *picard_dat,* const void ∗ *res_data,* const void ∗ *evalx_data*)

5.14.2.17 int pjfnk (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &F, const void ∗data) *res,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *x,* **PJFNK_DATA** ∗ *pjfnk_dat,* const void ∗ *res_data,* const void ∗ *precon_data*)

5.14.2.18 int update_arnoldi_solution (**Matrix**< double > & *x,* **Matrix**< double > & *x0,* **ARNOLDI_DATA** ∗ *arnoldi_dat*)

5.15 lark.h File Reference

Linear Algebra Residual Kernels.

```
#include "macaw.h"
#include <float.h>
```

Classes

- struct ARNOLDI_DATA

    Data structure for the construction of the Krylov subspaces for a linear system.
- struct GMRESLP_DATA

    Data structure for implementation of the Restarted GMRES algorithm with Left Preconditioning.
- struct GMRESRP_DATA

    Data structure for the Restarted GMRES algorithm with Right Preconditioning.
- struct PCG_DATA

    Data structure for implementation of the PCG algorithms for symmetric linear systems.
- struct BiCGSTAB_DATA

    Data structure for the implementation of the BiCGSTAB algorithm for non-symmetric linear systems.
- struct CGS_DATA

    Data structure for the implementation of the CGS algorithm for non-symmetric linear systems.
- struct OPTRANS_DATA

    Data structure for implementation of linear operator transposition.
- struct GCR_DATA

    Data structure for the implementation of the GCR algorithm for non-symmetric linear systems.
- struct GMRESR_DATA

    Data structure for the implementation of GCR with Nested GMRES preconditioning (i.e., GMRESR)
- struct PICARD_DATA

    Data structure for the implementation of a Picard or Fixed-Point iteration for non-linear systems.
- struct BACKTRACK_DATA

    Data structure for the implementation of Backtracking Linesearch.
- struct PJFNK_DATA

    Data structure for the implementation of the PJFNK algorithm for non-linear systems.
- struct NUM_JAC_DATA

    Data structure to form a numerical jacobian matrix with finite differences.

Enumerations

- enum krylov_method {
  GMRESLP, PCG, BiCGSTAB, CGS,
  FOM, GMRESRP, GCR, GMRESR }

    Enum of definitions for linear solver types in PJFNK.

Functions

- int update_arnoldi_solution (Matrix< double > &x, Matrix< double > &x0, ARNOLDI_DATA ∗arnoldi_dat)
- int arnoldi (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗data), Matrix< double > &r0, ARNOLDI_DATA ∗arnoldi_dat, const void ∗matvec_data, const void ∗precon_data)
- int gmresLeftPreconditioned (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗data), Matrix< double > &b, GMRESLP_DATA ∗gmreslp_dat, const void ∗matvec_data, const void ∗precon_data)
- int fom (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗data), Matrix< double > &b, GMRESLP_DATA ∗gmreslp_dat, const void ∗matvec_data, const void ∗precon_data)
- int gmresRightPreconditioned (int(∗matvec)(const Matrix< double > &v, Matrix< double > &w, const void ∗data), int(∗precon)(const Matrix< double > &b, Matrix< double > &p, const void ∗data), Matrix< double > &b, GMRESRP_DATA ∗gmresrp_dat, const void ∗matvec_data, const void ∗precon_data)

- int pcg (int(∗matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &z, const void ∗data), Matrix< double > &b, PCG_DATA ∗pcg_dat, const void ∗matvec_data, const void ∗precon_data)
- int bicgstab (int(∗matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &z, const void ∗data), Matrix< double > &b, BiCGSTAB_DATA ∗bicg_dat, const void ∗matvec_data, const void ∗precon_data)
- int cgs (int(∗matvec)(const Matrix< double > &p, Matrix< double > &Ap, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &z, const void ∗data), Matrix< double > &b, CGS_DATA ∗cgs_dat, const void ∗matvec_data, const void ∗precon_data)
- int operatorTranspose (int(∗matvec)(const Matrix< double > &v, Matrix< double > &Av, const void ∗data), Matrix< double > &r, Matrix< double > &u, OPTRANS_DATA ∗transpose_dat, const void ∗matvec_data)
- int gcr (int(∗matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &Mr, const void ∗data), Matrix< double > &b, GCR_DATA ∗gcr_dat, const void ∗matvec_data, const void ∗precon_data)
- int gmresPreconditioner (const Matrix< double > &r, Matrix< double > &Mr, const void ∗data)
- int gmresr (int(∗matvec)(const Matrix< double > &x, Matrix< double > &Ax, const void ∗data), int(∗terminal-_precon)(const Matrix< double > &r, Matrix< double > &Mr, const void ∗data), Matrix< double > &b, GMRESR_DATA ∗gmresr_dat, const void ∗matvec_data, const void ∗term_precon_data)
- int picard (int(∗res)(const Matrix< double > &x, Matrix< double > &r, const void ∗data), int(∗evalx)(const Matrix< double > &x0, Matrix< double > &x, const void ∗data), Matrix< double > &x, PICARD_DATA ∗picard_dat, const void ∗res_data, const void ∗evalx_data)
- int jacvec (const Matrix< double > &v, Matrix< double > &Jv, const void ∗data)
- int backtrackLineSearch (int(∗feval)(const Matrix< double > &x, Matrix< double > &F, const void ∗data), Matrix< double > &Fkp1, Matrix< double > &xkp1, Matrix< double > &pk, double normFk, BACKTRACK-_DATA ∗backtrack_dat, const void ∗feval_data)
- int pjfnk (int(∗res)(const Matrix< double > &x, Matrix< double > &F, const void ∗data), int(∗precon)(const Matrix< double > &r, Matrix< double > &p, const void ∗data), Matrix< double > &x, PJFNK_DATA ∗pjfnk-_dat, const void ∗res_data, const void ∗precon_data)
- int NumericalJacobian (int(∗Func)(const Matrix< double > &x, Matrix< double > &F, const void ∗user_-data), const Matrix< double > &x, Matrix< double > &J, int Nx, int Nf, NUM_JAC_DATA ∗jac_dat, const void ∗user_data)
- int LARK_TESTS ()

5.15.1 Detailed Description

Linear Algebra Residual Kernels. lark.cpp

The functions contained within are designed to solve generic linear and non-linear square systems of equations given a function argument and data from the user. Optionally, the user can also provide a function to return a preconditioning result that will be applied to the system.

Having the user define how the preconditioning is carried out provides two major advantages: (1) we do not need to store and large, sparse preconditioning matrices and instead only store the preconditioned vector result and (2) this allows the user to use any kind of preconditioner they see fit for their problem.

The Arnoldi function is typically not called by the user, but can be if desired. It accepts the function arguments and a residual vector to form an orthonormal basis of the Krylov subspace using the Modified Gram-Schmidt process (aka Arnoldi Iteration). This function is called by GMRES to iteratively solve a linear system of equations. Note that you can use this function to directly solve the linear system as long as that system is not too large. Construction of the basis is expensive, which is why this is used as a sub-function of an iterative method.

The Restarted GMRES function will accept function arguments for a linear system and attempt to solve said system iteratively by constructing an orthonormal basis from the Krylov function. Note that this GMRES function does support restarting and will use restarting by default if the linear system is too large.

Also included is a GMRES algorithm without restarting. This will directly solve the linear system within residual tolerance using a Full Orthogonal basis set of that system. It is equivalent to calling the Krylov method with the k parameter equal to N (i.e. the number of equations). This method is nick-named the Full Othogonalization Method (FOM), although the true FOM algorithm in literature is slightly different.

The PJFNK function will accept function arguments for a square, non-linear system of equations and attempt to solve it iteratively using both the GMRES and Krylov functions with Newton's method to convert the non-linear system into a linear system.

Also built here is a PCG implementation for solving symmetric linear systems. Can also be called by PJFNK if we know that the linear system (i.e. the Jacobian) is symmetric. This algorithm is significantly more efficient than GMRES, but is only valid if the system of equations is symmetric.

Other linear solvers implemented in this work are the BiCGSTAB and CGS algorithms for non-symmetric, positive definite matrices. These algorithms are significantly more computationally efficient than GMRES or FOM. However, they can both break down if the linear system is poorly conditioned. In general, you only want to use these methods if you have preconditioning available and your linear system is very, very large. Otherwise, you will be better suited to using GMRES or FOM.

There is also an implementation of the Generalized Conjugate Residual (GCR) method with and without restarting. This is a GMRES-like method that should give the exact solution within N iterations, where N is the original size of the matrix. Built ontop of the GCR method is a GMRESR (or GMRES Recursive) algorithm that uses GCR as the base method and performs GMRESRP iterations as a preconditioner at each iteration of GCR. This is the only linear solver that has built-in preconditioning. As a result, it may be slower than other algorithms for simple problems, but generally will have much better convergence behavior and will almost always give better residual reduction, even for hard to solve problems.

NOTE: There are three GMRES implementations: (i) gmresLP, (ii) fom, and (iii) gmresRP. GMRESLP is a restarted GMRES implementation that is left preconditioned and only checks the residual on the outer loops. This may be less efficient than GMRESRP, which can check both outer and inner loop residuals. However, GMRESRP has to use right preconditioning, which also slightly changes the convergence behavior of the linear system. GMRES with left preconditioning and without restarting will just build the full subspace by default, thus solving the system exactly, but may require too much memory. You can do a GMRESRP unrestarted by specifying that the restart parameter be equal to the size of the problem.

Author

Austin Ladshaw

Version

0.0 beta

Date

10/14/2014

Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

5.15.2 Enumeration Type Documentation

5.15.2.1 enum krylov_method

Enum of definitions for linear solver types in PJFNK.

Enum delineates the available Krylov Subspace methods that can be used to solve the linear sub-problem at each non-linear iteration in a Newton method.

Enumerator

GMRESLP

PCG

BiCGSTAB

CGS

FOM

GMRESRP

GCR

GMRESR

5.15.3 Function Documentation

5.15.3.1 int arnoldi (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *r0,* **ARNOLDI_DATA** ∗ *arnoldi_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.2 int backtrackLineSearch (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &F, const void ∗data) *feval,* **Matrix**< double > & *Fkp1,* **Matrix**< double > & *xkp1,* **Matrix**< double > & *pk,* double *normFk,* **BACKTRACK_DATA** ∗ *backtrack_dat,* const void ∗ *feval_data*)

5.15.3.3 int bicgstab (int(∗)(const **Matrix**< double > &p, **Matrix**< double > &Ap, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &z, const void ∗data) *precon,* **Matrix**< double > & *b,* **BiCGSTAB_DATA** ∗ *bicg_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.4 int cgs (int(∗)(const **Matrix**< double > &p, **Matrix**< double > &Ap, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &z, const void ∗data) *precon,* **Matrix**< double > & *b,* **CGS_DATA** ∗ *cgs_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.5 int fom (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *b,* **GMRESLP_DATA** ∗ *gmreslp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.6 int gcr (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &Ax, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &Mr, const void ∗data) *precon,* **Matrix**< double > & *b,* **GCR_DATA** ∗ *gcr_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.7 int gmresLeftPreconditioned (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *b,* **GMRESLP_DATA** ∗ *gmreslp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.8 int gmresPreconditioner (const **Matrix**< double > & *r,* **Matrix**< double > & *Mr,* const void ∗ *data*)

5.15.3.9 int gmresr (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &Ax, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &Mr, const void ∗data) *terminal_precon,* **Matrix**< double > & *b,* **GMRESR_DATA** ∗ *gmresr_dat,* const void ∗ *matvec_data,* const void ∗ *term_precon_data*)

5.15.3.10 int gmresRightPreconditioned (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &w, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &b, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *b,* **GMRESRP_DATA** ∗ *gmresrp_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.11 int jacvec (const **Matrix**< double > & *v,* **Matrix**< double > & *Jv,* const void ∗ *data*)

5.15.3.12 int LARK_TESTS ( )

5.15.3.13 int NumericalJacobian (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &F, const void ∗user_data) *Func,* const **Matrix**< double > & *x,* **Matrix**< double > & *J,* int *Nx,* int *Nf,* **NUM_JAC_DATA** ∗ *jac_dat,* const void ∗ *user_data*)

5.15.3.14 int operatorTranspose (int(∗)(const **Matrix**< double > &v, **Matrix**< double > &Av, const void ∗data) *matvec,* **Matrix**< double > & *r,* **Matrix**< double > & *u,* **OPTRANS_DATA** ∗ *transpose_dat,* const void ∗ *matvec_data*)

5.15.3.15 int pcg (int(∗)(const **Matrix**< double > &p, **Matrix**< double > &Ap, const void ∗data) *matvec,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &z, const void ∗data) *precon,* **Matrix**< double > & *b,* **PCG_DATA** ∗ *pcg_dat,* const void ∗ *matvec_data,* const void ∗ *precon_data*)

5.15.3.16 int picard (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &r, const void ∗data) *res,* int(∗)(const **Matrix**< double > &x0, **Matrix**< double > &x, const void ∗data) *evalx,* **Matrix**< double > & *x,* **PICARD_DATA** ∗ *picard_dat,* const void ∗ *res_data,* const void ∗ *evalx_data*)

5.15.3.17 int pjfnk (int(∗)(const **Matrix**< double > &x, **Matrix**< double > &F, const void ∗data) *res,* int(∗)(const **Matrix**< double > &r, **Matrix**< double > &p, const void ∗data) *precon,* **Matrix**< double > & *x,* **PJFNK_DATA** ∗ *pjfnk_dat,* const void ∗ *res_data,* const void ∗ *precon_data*)

5.15.3.18 int update_arnoldi_solution (**Matrix**< double > & *x,* **Matrix**< double > & *x0,* **ARNOLDI_DATA** ∗ *arnoldi_dat*)

5.16 macaw.cpp File Reference

```
#include "macaw.h"
```

Functions

- int MACAW_TESTS ()

5.16.1 Function Documentation

5.16.1.1 int MACAW_TESTS ( )

5.17 macaw.h File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include <exception>
#include "error.h"
```

Classes

- class Matrix< T >

Macros

- #define M_PI 3.14159265358979323846264338327950288 /∗ pi ∗/

Functions

- int MACAW_TESTS ()

5.17.1 Macro Definition Documentation

5.17.1.1 #define M_PI 3.14159265358979323846264338327950288 /∗ pi ∗/

5.17.2 Function Documentation

5.17.2.1 int MACAW_TESTS ( )

5.18 magpie.cpp File Reference

```
#include "magpie.h"
```

Functions

- double qo (double po, const void ∗data, int i)
- double dq_dp (double p, const void ∗data, int i)
- double q_p (double p, const void ∗data, int i)
- double PI (double po, const void ∗data, int i)
- double eMax (const void ∗data, int i)
- double Qst (double po, const void ∗data, int i)
- double lnact_mSPD (const double ∗par, const void ∗data, int i, volatile double PI)
- double grad_mSPD (const double ∗par, const void ∗data, int i)
- double qT (const double ∗par, const void ∗data)
- void initialGuess_mSPD (double ∗par, const void ∗data)
- void eval_po_PI (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_po_qo (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_po (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_eta (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_GPAST (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int MAGPIE (const void ∗data)
- int MAGPIE_SCENARIOS (const char ∗inputFileName, const char ∗sceneFileName)

5.18.1 Function Documentation

5.18.1.1 double dq_dp (double *p,* const void ∗ *data,* int *i*)

5.18.1.2 double eMax (const void ∗ *data,* int *i*)

5.18.1.3 void eval_eta (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.18.1.4 void eval_GPAST (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.18.1.5 void eval_po (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.18.1.6 void eval_po_PI (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.18.1.7 void eval_po_qo (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.18.1.8 double grad_mSPD (const double ∗ *par,* const void ∗ *data,* int *i*)

5.18.1.9 void initialGuess_mSPD (double ∗ *par,* const void ∗ *data*)

5.18.1.10 double lnact_mSPD (const double ∗ *par,* const void ∗ *data,* int *i,* volatile double *PI*)

5.18.1.11 int MAGPIE (const void ∗ *data*)

5.18.1.12 int MAGPIE_SCENARIOS (const char ∗ *inputFileName,* const char ∗ *sceneFileName*)

5.18.1.13 double PI (double *po,* const void ∗ *data,* int *i*)

5.18.1.14 double q_p (double *p,* const void ∗ *data,* int *i*)

5.18.1.15 double qo (double *po,* const void ∗ *data,* int *i*)

5.18.1.16 double Qst (double *po,* const void ∗ *data,* int *i*)

5.18.1.17 double qT (const double ∗ *par,* const void ∗ *data*)

5.19 magpie.h File Reference

```
#include "lmcurve.h"
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <vector>
#include <time.h>
#include <float.h>
#include <string>
#include "error.h"
```

Classes

- struct GSTA_DATA
- struct mSPD_DATA
- struct GPAST_DATA
- struct SYSTEM_DATA
- struct MAGPIE_DATA

Macros

- #define DBL_EPSILON 2.2204460492503131e-016
- #define Z 10.0
- #define A 3.13E+09
- #define V 18.92
- #define Po 100.0
- #define R 8.3144621
- #define Na 6.0221413E+23
- #define kB 1.3806488E-23
- #define shapeFactor(v_i) (((Z - 2) ∗ v_i) / (Z ∗ V)) + (2 / Z)

- #define lnKo(H, S, T) -(H / (R ∗ T)) + (S / R)
- #define He(qm, K1, m) (qm ∗ K1) / (m ∗ Po)

Functions

- double qo (double po, const void ∗data, int i)
- double dq_dp (double p, const void ∗data, int i)
- double q_p (double p, const void ∗data, int i)
- double PI (double po, const void ∗data, int i)
- double Qst (double po, const void ∗data, int i)
- double eMax (const void ∗data, int i)
- double lnact_mSPD (const double ∗par, const void ∗data, int i, volatile double PI)
- double grad_mSPD (const double ∗par, const void ∗data, int i)
- double qT (const double ∗par, const void ∗data)
- void initialGuess_mSPD (double ∗par, const void ∗data)
- void eval_po_PI (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_po_qo (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_po (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_eta (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- void eval_GPAST (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int MAGPIE (const void ∗data)
- int MAGPIE_SCENARIOS (const char ∗inputFileName, const char ∗sceneFileName)

5.19.1 Macro Definition Documentation

5.19.1.1 #define A 3.13E+09

5.19.1.2 #define DBL_EPSILON 2.2204460492503131e-016

5.19.1.3 #define He(*qm*, *K1*, *m*) (qm ∗ K1) / (m ∗ Po)

5.19.1.4 #define kB 1.3806488E-23

5.19.1.5 #define lnKo(*H*, *S*, *T*) -(H / (R ∗ T)) + (S / R)

5.19.1.6 #define Na 6.0221413E+23

5.19.1.7 #define Po 100.0

5.19.1.8 #define R 8.3144621

5.19.1.9 #define shapeFactor(*v_i*) (((Z - 2) ∗ v_i) / (Z ∗ V)) + (2 / Z)

5.19.1.10 #define V 18.92

5.19.1.11 #define Z 10.0

5.19.2 Function Documentation

5.19.2.1 double dq_dp (double *p*, const void ∗ *data*, int *i*)

5.19.2.2 double eMax (const void ∗ *data*, int *i*)

5.19.2.3 void eval_eta (const double ∗ *par*, int *m_dat*, const void ∗ *data*, double ∗ *fvec*, int ∗ *info*)

5.19.2.4 void eval_GPAST (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.19.2.5 void eval_po (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.19.2.6 void eval_po_PI (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.19.2.7 void eval_po_qo (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.19.2.8 double grad_mSPD (const double ∗ *par,* const void ∗ *data,* int *i*)

5.19.2.9 void initialGuess_mSPD (double ∗ *par,* const void ∗ *data*)

5.19.2.10 double lnact_mSPD (const double ∗ *par,* const void ∗ *data,* int *i,* volatile double *PI*)

5.19.2.11 int MAGPIE (const void ∗ *data*)

5.19.2.12 int MAGPIE_SCENARIOS (const char ∗ *inputFileName,* const char ∗ *sceneFileName*)

5.19.2.13 double PI (double *po,* const void ∗ *data,* int *i*)

5.19.2.14 double q_p (double *p,* const void ∗ *data,* int *i*)

5.19.2.15 double qo (double *po,* const void ∗ *data,* int *i*)

5.19.2.16 double Qst (double *po,* const void ∗ *data,* int *i*)

5.19.2.17 double qT (const double ∗ *par,* const void ∗ *data*)

5.20 main.cpp File Reference

Main Function.

`#include "ui.h"`

Functions

- int main (int argc, const char ∗argv[])

5.20.1 Detailed Description

Main Function. User input provided at time of execution is used to call the ui functions

Author

Austin Ladshaw

Version

0.0 beta

Date

08/25/2015

Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

5.20.2 Function Documentation

5.20.2.1 int main (int *argc*, const char ∗ *argv[]*)

Command Line Interface: To Override, comment out this line and replace with your own run function

5.21 mola.cpp File Reference

```
#include "mola.h"
```

Functions

- int MOLA_TESTS ()

5.21.1 Function Documentation

5.21.1.1 int MOLA_TESTS ( )

5.22 mola.h File Reference

```
#include <ctype.h>
#include "eel.h"
```

Classes

- class Molecule

Functions

- int MOLA_TESTS ()

5.22.1 Function Documentation

5.22.1.1 int MOLA_TESTS ( )

5.23 monkfish.cpp File Reference

```
#include "monkfish.h"
```

Functions

- double default_porosity (int i, int l, const void ∗user_data)
- double default_density (int i, int l, const void ∗user_data)
- double default_interparticle_diffusion (int i, int l, const void ∗user_data)
- double default_monk_adsorption (int i, int l, const void ∗user_data)
- double default_monk_equilibrium (int i, int l, const void ∗user_data)
- double default_monkfish_retardation (int i, int l, const void ∗user_data)
- double default_exterior_concentration (int i, const void ∗user_data)
- double default_film_transfer (int i, const void ∗user_data)
- int MONKFISH_TESTS ()

5.23.1 Function Documentation

5.23.1.1 double default_density (int *i*, int *l*, const void ∗ *user_data*)

5.23.1.2 double default_exterior_concentration (int *i*, const void ∗ *user_data*)

5.23.1.3 double default_film_transfer (int *i*, const void ∗ *user_data*)

5.23.1.4 double default_interparticle_diffusion (int *i*, int *l*, const void ∗ *user_data*)

5.23.1.5 double default_monk_adsorption (int *i*, int *l*, const void ∗ *user_data*)

5.23.1.6 double default_monk_equilibrium (int *i*, int *l*, const void ∗ *user_data*)

5.23.1.7 double default_monkfish_retardation (int *i*, int *l*, const void ∗ *user_data*)

5.23.1.8 double default_porosity (int *i*, int *l*, const void ∗ *user_data*)

5.23.1.9 int MONKFISH_TESTS ( )

5.24 monkfish.h File Reference

```
#include "dogfish.h"
```

Classes

- struct MONKFISH_PARAM
- struct MONKFISH_DATA

Functions

- double default_porosity (int i, int l, const void ∗user_data)
- double default_density (int i, int l, const void ∗user_data)
- double default_interparticle_diffusion (int i, int l, const void ∗user_data)
- double default_monk_adsorption (int i, int l, const void ∗user_data)
- double default_monk_equilibrium (int i, int l, const void ∗user_data)
- double default_monkfish_retardation (int i, int l, const void ∗user_data)
- double default_exterior_concentration (int i, const void ∗user_data)
- double default_film_transfer (int i, const void ∗user_data)

- int setup_MONKFISH_DATA (FILE ∗file, double(∗eval_porosity)(int i, int l, const void ∗user_data), double(∗eval_density)(int i, int l, const void ∗user_data), double(∗eval_ext_diff)(int i, int l, const void ∗user-_data), double(∗eval_adsorb)(int i, int l, const void ∗user_data), double(∗eval_retard)(int i, int l, const void ∗user_data), double(∗eval_ext_conc)(int i, const void ∗user_data), double(∗eval_ext_film)(int i, const void ∗user_data), double(∗dog_diffusion)(int i, int l, const void ∗user_data), double(∗dog_ext_film)(int i, const void ∗user_data), double(∗dog_surf_conc)(int i, const void ∗user_data), const void ∗user_data, MONKFISH_DA-TA ∗monk_dat)
- int MONKFISH_TESTS ()

5.24.1 Function Documentation

5.24.1.1 double default_density (int *i,* int *l,* const void ∗ *user_data*)

5.24.1.2 double default_exterior_concentration (int *i,* const void ∗ *user_data*)

5.24.1.3 double default_film_transfer (int *i,* const void ∗ *user_data*)

5.24.1.4 double default_interparticle_diffusion (int *i,* int *l,* const void ∗ *user_data*)

5.24.1.5 double default_monk_adsorption (int *i,* int *l,* const void ∗ *user_data*)

5.24.1.6 double default_monk_equilibrium (int *i,* int *l,* const void ∗ *user_data*)

5.24.1.7 double default_monkfish_retardation (int *i,* int *l,* const void ∗ *user_data*)

5.24.1.8 double default_porosity (int *i,* int *l,* const void ∗ *user_data*)

5.24.1.9 int MONKFISH_TESTS ( )

5.24.1.10 int setup_MONKFISH_DATA (FILE ∗ *file,* double(∗)(int i, int l, const void ∗user_data) *eval_porosity,* double(∗)(int i, int l, const void ∗user_data) *eval_density,* double(∗)(int i, int l, const void ∗user_data) *eval_ext_diff,* double(∗)(int i, int l, const void ∗user_data) *eval_adsorb,* double(∗)(int i, int l, const void ∗user_data) *eval_retard,* double(∗)(int i, const void ∗user_data) *eval_ext_conc,* double(∗)(int i, const void ∗user_data) *eval_ext_film,* double(∗)(int i, int l, const void ∗user_data) *dog_diffusion,* double(∗)(int i, const void ∗user_data) *dog_ext_film,* double(∗)(int i, const void ∗user_data) *dog_surf_conc,* const void ∗ *user_data,* MONKFISH_DATA ∗ *monk_dat*)

5.25 sandbox.cpp File Reference

```
#include "sandbox.h"
```

Functions

- int Speciation_Test01_Function (const Matrix< double > &x, Matrix< double > &F, const void ∗res_data)
- int Speciation_Test01_Jacobian (const Matrix< double > &x, Matrix< double > &J, const void ∗precon_data)
- int Speciation_Test01_Guess (const void ∗user_data)
- int Speciation_Test01_MatVec (const Matrix< double > &x, Matrix< double > &Ax, const void ∗matvec_-data)
- int RUN_SANDBOX ()

5.25.1 Function Documentation

5.25.1.1 int RUN_SANDBOX ( )

5.25.1.2 int Speciation_Test01_Function (const **Matrix**< double > & *x*, **Matrix**< double > & *F*, const void ∗ *res_data*)

5.25.1.3 int Speciation_Test01_Guess (const void ∗ *user_data*)

5.25.1.4 int Speciation_Test01_Jacobian (const **Matrix**< double > & *x*, **Matrix**< double > & *J*, const void ∗ *precon_data*)

5.25.1.5 int Speciation_Test01_MatVec (const **Matrix**< double > & *x*, **Matrix**< double > & *Ax*, const void ∗ *matvec_data*)

5.26 sandbox.h File Reference

```
#include "flock.h"
#include "school.h"
```

Classes

- struct Speciation_Test01_Data

Functions

- int Speciation_Test01_Function (const Matrix< double > &x, Matrix< double > &F, const void ∗res_data)
- int Speciation_Test01_Jacobian (const Matrix< double > &x, Matrix< double > &J, const void ∗precon_data)
- int Speciation_Test01_Guess (const void ∗user_data)
- int Speciation_Test01_MatVec (const Matrix< double > &x, Matrix< double > &Ax, const void ∗matvec_-data)
- int RUN_SANDBOX ()

5.26.1 Function Documentation

5.26.1.1 int RUN_SANDBOX ( )

5.26.1.2 int Speciation_Test01_Function (const **Matrix**< double > & *x*, **Matrix**< double > & *F*, const void ∗ *res_data*)

5.26.1.3 int Speciation_Test01_Guess (const void ∗ *user_data*)

5.26.1.4 int Speciation_Test01_Jacobian (const **Matrix**< double > & *x*, **Matrix**< double > & *J*, const void ∗ *precon_data*)

5.26.1.5 int Speciation_Test01_MatVec (const **Matrix**< double > & *x*, **Matrix**< double > & *Ax*, const void ∗ *matvec_data*)

5.27 school.h File Reference

```
#include "eel.h"
#include "mola.h"
#include "shark.h"
#include "dogfish.h"
#include "monkfish.h"
#include "yaml_wrapper.h"
```

5.28 scopsowl.cpp File Reference

```
#include "scopsowl.h"
```

Functions

- void print2file_species_header (FILE ∗Output, SCOPSOWL_DATA ∗owl_dat, int i)
- void print2file_SCOPSOWL_time_header (FILE ∗Output, SCOPSOWL_DATA ∗owl_dat, int i)
- void print2file_SCOPSOWL_header (SCOPSOWL_DATA ∗owl_dat)
- void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA ∗owl_dat)
- void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA ∗owl_dat)
- double default_adsorption (int i, int l, const void ∗user_data)
- double default_retardation (int i, int l, const void ∗user_data)
- double default_pore_diffusion (int i, int l, const void ∗user_data)
- double default_surf_diffusion (int i, int l, const void ∗user_data)
- double default_effective_diffusion (int i, int l, const void ∗user_data)
- double const_pore_diffusion (int i, int l, const void ∗user_data)
- double default_filmMassTransfer (int i, const void ∗user_data)
- double const_filmMassTransfer (int i, const void ∗user_data)
- int setup_SCOPSOWL_DATA (FILE ∗file, double(∗eval_sorption)(int i, int l, const void ∗user_data), double(∗eval_retardation)(int i, int l, const void ∗user_data), double(∗eval_pore_diff)(int i, int l, const void ∗user_data), double(∗eval_filmMT)(int i, const void ∗user_data), double(∗eval_surface_diff)(int i, int l, const void ∗user_data), const void ∗user_data, MIXED_GAS ∗gas_data, SCOPSOWL_DATA ∗owl_data)
- int SCOPSOWL_Executioner (SCOPSOWL_DATA ∗owl_dat)
- int set_SCOPSOWL_ICs (SCOPSOWL_DATA ∗owl_dat)
- int set_SCOPSOWL_timestep (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL_preprocesses (SCOPSOWL_DATA ∗owl_dat)
- int set_SCOPSOWL_params (const void ∗user_data)
- int SCOPSOWL_postprocesses (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL_reset (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL (SCOPSOWL_DATA ∗owl_dat)
- int LARGE_CYCLE_TEST01 (SCOPSOWL_DATA ∗owl_dat)
- int SMALL_CYCLE_TEST02 (SCOPSOWL_DATA ∗owl_dat)
- int CURVE_TEST03 (SCOPSOWL_DATA ∗owl_dat)
- int CURVE_TEST04 (SCOPSOWL_DATA ∗owl_dat)
- int CURVE_TEST05 (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL_SCENARIOS (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate)
- int SCOPSOWL_TESTS ()

5.28.1 Function Documentation

5.28.1.1 double const_filmMassTransfer (int *i*, const void ∗ *user_data*)

5.28.1.2 double const_pore_diffusion (int *i*, int *l*, const void ∗ *user_data*)

5.28.1.3 int CURVE_TEST03 (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.4 int CURVE_TEST04 (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.5 int CURVE_TEST05 (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.6 double default_adsorption (int *i,* int *l,* const void ∗ *user_data*)

5.28.1.7 double default_effective_diffusion (int *i,* int *l,* const void ∗ *user_data*)

5.28.1.8 double default_filmMassTransfer (int *i,* const void ∗ *user_data*)

5.28.1.9 double default_pore_diffusion (int *i,* int *l,* const void ∗ *user_data*)

5.28.1.10 double default_retardation (int *i,* int *l,* const void ∗ *user_data*)

5.28.1.11 double default_surf_diffusion (int *i,* int *l,* const void ∗ *user_data*)

5.28.1.12 int LARGE_CYCLE_TEST01 (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.13 void print2file_SCOPSOWL_header (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.14 void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.15 void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.16 void print2file_SCOPSOWL_time_header (FILE ∗ *Output,* SCOPSOWL_DATA ∗ *owl_dat,* int *i*)

5.28.1.17 void print2file_species_header (FILE ∗ *Output,* SCOPSOWL_DATA ∗ *owl_dat,* int *i*)

5.28.1.18 int SCOPSOWL (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.19 int SCOPSOWL_Executioner (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.20 int SCOPSOWL_postprocesses (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.21 int SCOPSOWL_preprocesses (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.22 int SCOPSOWL_reset (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.23 int SCOPSOWL_SCENARIOS (const char ∗ *scene,* const char ∗ *sorbent,* const char ∗ *comp,* const char ∗ *sorbate*)

5.28.1.24 int SCOPSOWL_TESTS ( )

5.28.1.25 int set_SCOPSOWL_ICs (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.26 int set_SCOPSOWL_params (const void ∗ *user_data*)

5.28.1.27 int set_SCOPSOWL_timestep (SCOPSOWL_DATA ∗ *owl_dat*)

5.28.1.28 int setup_SCOPSOWL_DATA (FILE ∗ *file,* double(∗)(int i, int l, const void ∗user_data) *eval_sorption,* double(∗)(int i, int l, const void ∗user_data) *eval_retardation,* double(∗)(int i, int l, const void ∗user_data) *eval_pore_diff,* double(∗)(int i, const void ∗user_data) *eval_filmMT,* double(∗)(int i, int l, const void ∗user_data) *eval_surface_diff,* const void ∗ *user_data,* MIXED_GAS ∗ *gas_data,* SCOPSOWL_DATA ∗ *owl_data*)

5.28.1.29 int SMALL_CYCLE_TEST02 (SCOPSOWL_DATA ∗ *owl_dat*)

5.29 scopsowl.h File Reference

```
#include "egret.h"
#include "skua.h"
```

Classes

- struct SCOPSOWL_PARAM_DATA
- struct SCOPSOWL_DATA

Macros

- #define SCOPSOWL_HPP_
- #define Dp(Dm, ep) (ep∗ep∗Dm)
- #define Dk(rp, T, MW) (9700.0∗rp∗pow((T/MW),0.5))
- #define avgDp(Dp, Dk) (pow(((1/Dp)+(1/Dk)),-1.0))

Functions

- void print2file_species_header (FILE ∗Output, SCOPSOWL_DATA ∗owl_dat, int i)
- void print2file_SCOPSOWL_time_header (FILE ∗Output, SCOPSOWL_DATA ∗owl_dat, int i)
- void print2file_SCOPSOWL_header (SCOPSOWL_DATA ∗owl_dat)
- void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA ∗owl_dat)
- void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA ∗owl_dat)
- double default_adsorption (int i, int l, const void ∗user_data)
- double default_retardation (int i, int l, const void ∗user_data)
- double default_pore_diffusion (int i, int l, const void ∗user_data)
- double default_surf_diffusion (int i, int l, const void ∗user_data)
- double default_effective_diffusion (int i, int l, const void ∗user_data)
- double const_pore_diffusion (int i, int l, const void ∗user_data)
- double default_filmMassTransfer (int i, const void ∗user_data)
- double const_filmMassTransfer (int i, const void ∗user_data)
- int setup_SCOPSOWL_DATA (FILE ∗file, double(∗eval_sorption)(int i, int l, const void ∗user_data), double(∗eval_retardation)(int i, int l, const void ∗user_data), double(∗eval_pore_diff)(int i, int l, const void ∗user_data), double(∗eval_filmMT)(int i, const void ∗user_data), double(∗eval_surface_diff)(int i, int l, const void ∗user_data), const void ∗user_data, MIXED_GAS ∗gas_data, SCOPSOWL_DATA ∗owl_data)
- int SCOPSOWL_Executioner (SCOPSOWL_DATA ∗owl_dat)
- int set_SCOPSOWL_ICs (SCOPSOWL_DATA ∗owl_dat)
- int set_SCOPSOWL_timestep (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL_preprocesses (SCOPSOWL_DATA ∗owl_dat)
- int set_SCOPSOWL_params (const void ∗user_data)
- int SCOPSOWL_postprocesses (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL_reset (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL (SCOPSOWL_DATA ∗owl_dat)
- int LARGE_CYCLE_TEST01 (SCOPSOWL_DATA ∗owl_dat)
- int SMALL_CYCLE_TEST02 (SCOPSOWL_DATA ∗owl_dat)
- int CURVE_TEST03 (SCOPSOWL_DATA ∗owl_dat)
- int CURVE_TEST04 (SCOPSOWL_DATA ∗owl_dat)
- int CURVE_TEST05 (SCOPSOWL_DATA ∗owl_dat)
- int SCOPSOWL_SCENARIOS (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate)
- int SCOPSOWL_TESTS ()

5.29.1 Macro Definition Documentation

5.29.1.1 #define avgDp(Dp, Dk) (pow(((1/Dp)+(1/Dk)),-1.0))

5.29.1.2 #define Dk(*rp*, *T*, *MW*) (9700.0∗rp∗pow((T/MW),0.5))

5.29.1.3 #define Dp(*Dm*, *ep*) (ep∗ep∗Dm)

5.29.1.4 #define SCOPSOWL_HPP_

5.29.2 Function Documentation

5.29.2.1 double const_filmMassTransfer (int *i*, const void ∗ *user_data*)

5.29.2.2 double const_pore_diffusion (int *i*, int *l*, const void ∗ *user_data*)

5.29.2.3 int CURVE_TEST03 (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.4 int CURVE_TEST04 (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.5 int CURVE_TEST05 (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.6 double default_adsorption (int *i*, int *l*, const void ∗ *user_data*)

5.29.2.7 double default_effective_diffusion (int *i*, int *l*, const void ∗ *user_data*)

5.29.2.8 double default_filmMassTransfer (int *i*, const void ∗ *user_data*)

5.29.2.9 double default_pore_diffusion (int *i*, int *l*, const void ∗ *user_data*)

5.29.2.10 double default_retardation (int *i*, int *l*, const void ∗ *user_data*)

5.29.2.11 double default_surf_diffusion (int *i*, int *l*, const void ∗ *user_data*)

5.29.2.12 int LARGE_CYCLE_TEST01 (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.13 void print2file_SCOPSOWL_header (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.14 void print2file_SCOPSOWL_result_new (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.15 void print2file_SCOPSOWL_result_old (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.16 void print2file_SCOPSOWL_time_header (FILE ∗ *Output*, SCOPSOWL_DATA ∗ *owl_dat*, int *i*)

5.29.2.17 void print2file_species_header (FILE ∗ *Output*, SCOPSOWL_DATA ∗ *owl_dat*, int *i*)

5.29.2.18 int SCOPSOWL (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.19 int SCOPSOWL_Executioner (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.20 int SCOPSOWL_postprocesses (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.21 int SCOPSOWL_preprocesses (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.22 int SCOPSOWL_reset (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.23 int SCOPSOWL_SCENARIOS (const char ∗ *scene,* const char ∗ *sorbent,* const char ∗ *comp,* const char ∗ *sorbate*)

5.29.2.24 int SCOPSOWL_TESTS ( )

5.29.2.25 int set_SCOPSOWL_ICs (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.26 int set_SCOPSOWL_params (const void ∗ *user_data*)

5.29.2.27 int set_SCOPSOWL_timestep (SCOPSOWL_DATA ∗ *owl_dat*)

5.29.2.28 int setup_SCOPSOWL_DATA (FILE ∗ *file,* double(∗)(int i, int l, const void ∗user_data) *eval_sorption,* double(∗)(int i, int l, const void ∗user_data) *eval_retardation,* double(∗)(int i, int l, const void ∗user_data) *eval_pore_diff,* double(∗)(int i, const void ∗user_data) *eval_filmMT,* double(∗)(int i, int l, const void ∗user_data) *eval_surface_diff,* const void ∗ *user_data,* MIXED_GAS ∗ *gas_data,* SCOPSOWL_DATA ∗ *owl_data*)

5.29.2.29 int SMALL_CYCLE_TEST02 (SCOPSOWL_DATA ∗ *owl_dat*)

5.30 scopsowl_opt.cpp File Reference

```
#include "scopsowl_opt.h"
```

Functions

- int SCOPSOWL_OPT_set_y (SCOPSOWL_OPT_DATA ∗owl_opt)
- int initial_guess_SCOPSOWL (SCOPSOWL_OPT_DATA ∗owl_opt)
- void eval_SCOPSOWL_Uptake (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int SCOPSOWL_OPTIMIZE (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate, const char ∗data)

5.30.1 Function Documentation

5.30.1.1 void eval_SCOPSOWL_Uptake (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.30.1.2 int initial_guess_SCOPSOWL (SCOPSOWL_OPT_DATA ∗ *owl_opt*)

5.30.1.3 int SCOPSOWL_OPT_set_y (SCOPSOWL_OPT_DATA ∗ *owl_opt*)

5.30.1.4 int SCOPSOWL_OPTIMIZE (const char ∗ *scene,* const char ∗ *sorbent,* const char ∗ *comp,* const char ∗ *sorbate,* const char ∗ *data*)

5.31 scopsowl_opt.h File Reference

```
#include "scopsowl.h"
```

Classes

- struct SCOPSOWL_OPT_DATA

Functions

- int SCOPSOWL_OPT_set_y (SCOPSOWL_OPT_DATA ∗owl_opt)
- int initial_guess_SCOPSOWL (SCOPSOWL_OPT_DATA ∗owl_opt)
- void eval_SCOPSOWL_Uptake (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int SCOPSOWL_OPTIMIZE (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate, const char ∗data)

5.31.1 Function Documentation

5.31.1.1 void eval_SCOPSOWL_Uptake (const double ∗ *par,* int *m_dat,* const void ∗ *data,* double ∗ *fvec,* int ∗ *info*)

5.31.1.2 int initial_guess_SCOPSOWL (SCOPSOWL_OPT_DATA ∗ *owl_opt*)

5.31.1.3 int SCOPSOWL_OPT_set_y (SCOPSOWL_OPT_DATA ∗ *owl_opt*)

5.31.1.4 int SCOPSOWL_OPTIMIZE (const char ∗ *scene,* const char ∗ *sorbent,* const char ∗ *comp,* const char ∗ *sorbate,* const char ∗ *data*)

5.32 shark.cpp File Reference

```
#include "shark.h"
```

Functions

- void print2file_shark_info (SHARK_DATA ∗shark_dat)
- void print2file_shark_header (SHARK_DATA ∗shark_dat)
- void print2file_shark_results_new (SHARK_DATA ∗shark_dat)
- void print2file_shark_results_old (SHARK_DATA ∗shark_dat)
- int ideal_solution (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int Davies_equation (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int DebyeHuckel_equation (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int DaviesLadshaw_equation (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int act_choice (const std::string &input)
- bool linesearch_choice (const std::string &input)
- int linearsolve_choice (const std::string &input)
- int Convert2LogConcentration (const Matrix< double > &x, Matrix< double > &logx)
- int Convert2Concentration (const Matrix< double > &logx, Matrix< double > &x)
- int read_scenario (SHARK_DATA ∗shark_dat)
- int read_options (SHARK_DATA ∗shark_dat)
- int read_species (SHARK_DATA ∗shark_dat)
- int read_massbalance (SHARK_DATA ∗shark_dat)
- int read_equilrxn (SHARK_DATA ∗shark_dat)
- int read_unsteadyrxn (SHARK_DATA ∗shark_dat)
- int setup_SHARK_DATA (FILE ∗file, int(∗residual)(const Matrix< double > &x, Matrix< double > &res, const void ∗data), int(∗activity)(const Matrix< double > &x, Matrix< double > &gama, const void ∗data), int(∗precond)(const Matrix< double > &r, Matrix< double > &p, const void ∗data), SHARK_DATA ∗dat, const void ∗activity_data, const void ∗residual_data, const void ∗precon_data, const void ∗other_data)
- int shark_add_customResidual (int i, double(∗other_res)(const Matrix< double > &x, SHARK_DATA ∗shark-_dat, const void ∗other_data), SHARK_DATA ∗shark_dat)
- int shark_parameter_check (SHARK_DATA ∗shark_dat)
- int shark_energy_calculations (SHARK_DATA ∗shark_dat)
- int shark_temperature_calculations (SHARK_DATA ∗shark_dat)

- int shark_pH_finder (SHARK_DATA ∗shark_dat)
- int shark_guess (SHARK_DATA ∗shark_dat)
- int shark_initial_conditions (SHARK_DATA ∗shark_dat)
- int shark_executioner (SHARK_DATA ∗shark_dat)
- int shark_timestep_const (SHARK_DATA ∗shark_dat)
- int shark_timestep_adapt (SHARK_DATA ∗shark_dat)
- int shark_preprocesses (SHARK_DATA ∗shark_dat)
- int shark_solver (SHARK_DATA ∗shark_dat)
- int shark_postprocesses (SHARK_DATA ∗shark_dat)
- int shark_reset (SHARK_DATA ∗shark_dat)
- int shark_residual (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int SHARK (SHARK_DATA ∗shark_dat)
- int SHARK_SCENARIO (const char ∗yaml_input)
- int SHARK_TESTS ()

5.32.1 Function Documentation

5.32.1.1 int act_choice (const std::string & *input*)

5.32.1.2 int Convert2Concentration (const **Matrix**< double > & *logx*, **Matrix**< double > & *x*)

5.32.1.3 int Convert2LogConcentration (const **Matrix**< double > & *x*, **Matrix**< double > & *logx*)

5.32.1.4 int Davies_equation (const **Matrix**< double > & *x*, **Matrix**< double > & *F*, const void ∗ *data*)

5.32.1.5 int DaviesLadshaw_equation (const **Matrix**< double > & *x*, **Matrix**< double > & *F*, const void ∗ *data*)

5.32.1.6 int DebyeHuckel_equation (const **Matrix**< double > & *x*, **Matrix**< double > & *F*, const void ∗ *data*)

5.32.1.7 int ideal_solution (const **Matrix**< double > & *x*, **Matrix**< double > & *F*, const void ∗ *data*)

5.32.1.8 int linearsolve_choice (const std::string & *input*)

5.32.1.9 bool linesearch_choice (const std::string & *input*)

5.32.1.10 void print2file_shark_header (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.11 void print2file_shark_info (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.12 void print2file_shark_results_new (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.13 void print2file_shark_results_old (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.14 int read_equilrxn (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.15 int read_massbalance (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.16 int read_options (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.17 int read_scenario (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.18 int read_species (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.19 int read_unsteadyrxn (**SHARK_DATA** ∗ *shark_dat*)

5.32.1.20 int setup_SHARK_DATA (FILE ∗ *file,* int(∗)(const Matrix< double > &x, Matrix< double > &res, const void ∗data) *residual,* int(∗)(const Matrix< double > &x, Matrix< double > &gama, const void ∗data) *activity,* int(∗)(const Matrix< double > &r, Matrix< double > &p, const void ∗data) *precond,* SHARK_DATA ∗ *dat,* const void ∗ *activity_data,* const void ∗ *residual_data,* const void ∗ *precon_data,* const void ∗ *other_data*)

5.32.1.21 int SHARK (SHARK_DATA ∗ *shark_dat*)

5.32.1.22 int shark_add_customResidual (int *i,* double(∗)(const Matrix< double > &x, SHARK_DATA ∗shark_dat, const void ∗other_data) *other_res,* SHARK_DATA ∗ *shark_dat*)

5.32.1.23 int shark_energy_calculations (SHARK_DATA ∗ *shark_dat*)

5.32.1.24 int shark_executioner (SHARK_DATA ∗ *shark_dat*)

5.32.1.25 int shark_guess (SHARK_DATA ∗ *shark_dat*)

5.32.1.26 int shark_initial_conditions (SHARK_DATA ∗ *shark_dat*)

5.32.1.27 int shark_parameter_check (SHARK_DATA ∗ *shark_dat*)

5.32.1.28 int shark_pH_finder (SHARK_DATA ∗ *shark_dat*)

5.32.1.29 int shark_postprocesses (SHARK_DATA ∗ *shark_dat*)

5.32.1.30 int shark_preprocesses (SHARK_DATA ∗ *shark_dat*)

5.32.1.31 int shark_reset (SHARK_DATA ∗ *shark_dat*)

5.32.1.32 int shark_residual (const Matrix< double > & *x,* Matrix< double > & *F,* const void ∗ *data*)

5.32.1.33 int SHARK_SCENARIO (const char ∗ *yaml_input*)

5.32.1.34 int shark_solver (SHARK_DATA ∗ *shark_dat*)

5.32.1.35 int shark_temperature_calculations (SHARK_DATA ∗ *shark_dat*)

5.32.1.36 int SHARK_TESTS ( )

5.32.1.37 int shark_timestep_adapt (SHARK_DATA ∗ *shark_dat*)

5.32.1.38 int shark_timestep_const (SHARK_DATA ∗ *shark_dat*)

5.33 shark.h File Reference

```
#include "mola.h"
#include "macaw.h"
#include "lark.h"
#include "yaml_wrapper.h"
```

Classes

- class MasterSpeciesList
- class Reaction
- class MassBalance

- class UnsteadyReaction
- class Mechanism
- class Precipitation
- class UnsteadyPrecipitation
- struct SHARK_DATA

Macros

- #define Rstd 8.3144621

Typedefs

- typedef struct SHARK_DATA SHARK_DATA

Enumerations

- enum valid_act {
  IDEAL, DAVIES, DEBYE_HUCKEL, DAVIES_LADSHAW,
  SIT, PITZER }

Functions

- void print2file_shark_info (SHARK_DATA ∗shark_dat)
- void print2file_shark_header (SHARK_DATA ∗shark_dat)
- void print2file_shark_results_new (SHARK_DATA ∗shark_dat)
- void print2file_shark_results_old (SHARK_DATA ∗shark_dat)
- int ideal_solution (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int Davies_equation (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int DebyeHuckel_equation (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int DaviesLadshaw_equation (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int act_choice (const std::string &input)
- bool linesearch_choice (const std::string &input)
- int linearsolve_choice (const std::string &input)
- int Convert2LogConcentration (const Matrix< double > &x, Matrix< double > &logx)
- int Convert2Concentration (const Matrix< double > &logx, Matrix< double > &x)
- int read_scenario (SHARK_DATA ∗shark_dat)
- int read_options (SHARK_DATA ∗shark_dat)
- int read_species (SHARK_DATA ∗shark_dat)
- int read_massbalance (SHARK_DATA ∗shark_dat)
- int read_equilrxn (SHARK_DATA ∗shark_dat)
- int read_unsteadyrxn (SHARK_DATA ∗shark_dat)
- int setup_SHARK_DATA (FILE ∗file, int(∗residual)(const Matrix< double > &x, Matrix< double > &res, const void ∗data), int(∗activity)(const Matrix< double > &x, Matrix< double > &gama, const void ∗data), int(∗precond)(const Matrix< double > &r, Matrix< double > &p, const void ∗data), SHARK_DATA ∗dat, const void ∗activity_data, const void ∗residual_data, const void ∗precon_data, const void ∗other_data)
- int shark_add_customResidual (int i, double(∗other_res)(const Matrix< double > &x, SHARK_DATA ∗shark-_dat, const void ∗other_data), SHARK_DATA ∗shark_dat)
- int shark_parameter_check (SHARK_DATA ∗shark_dat)
- int shark_energy_calculations (SHARK_DATA ∗shark_dat)
- int shark_temperature_calculations (SHARK_DATA ∗shark_dat)
- int shark_pH_finder (SHARK_DATA ∗shark_dat)
- int shark_guess (SHARK_DATA ∗shark_dat)
- int shark_initial_conditions (SHARK_DATA ∗shark_dat)

- int shark_executioner (SHARK_DATA ∗shark_dat)
- int shark_timestep_const (SHARK_DATA ∗shark_dat)
- int shark_timestep_adapt (SHARK_DATA ∗shark_dat)
- int shark_preprocesses (SHARK_DATA ∗shark_dat)
- int shark_solver (SHARK_DATA ∗shark_dat)
- int shark_postprocesses (SHARK_DATA ∗shark_dat)
- int shark_reset (SHARK_DATA ∗shark_dat)
- int shark_residual (const Matrix< double > &x, Matrix< double > &F, const void ∗data)
- int SHARK (SHARK_DATA ∗shark_dat)
- int SHARK_SCENARIO (const char ∗yaml_input)
- int SHARK_TESTS ()

5.33.1 Macro Definition Documentation

5.33.1.1 #define Rstd 8.3144621

5.33.2 Typedef Documentation

5.33.2.1 typedef struct SHARK_DATA SHARK_DATA

5.33.3 Enumeration Type Documentation

5.33.3.1 enum valid_act

Enumerator

IDEAL

DAVIES

DEBYE_HUCKEL

DAVIES_LADSHAW

SIT

PITZER

5.33.4 Function Documentation

5.33.4.1 int act_choice (const std::string & *input*)

5.33.4.2 int Convert2Concentration (const Matrix< double > & *logx*, Matrix< double > & *x*)

5.33.4.3 int Convert2LogConcentration (const Matrix< double > & *x*, Matrix< double > & *logx*)

5.33.4.4 int Davies_equation (const Matrix< double > & *x*, Matrix< double > & *F*, const void ∗ *data*)

5.33.4.5 int DaviesLadshaw_equation (const Matrix< double > & *x*, Matrix< double > & *F*, const void ∗ *data*)

5.33.4.6 int DebyeHuckel_equation (const Matrix< double > & *x*, Matrix< double > & *F*, const void ∗ *data*)

5.33.4.7 int ideal_solution (const Matrix< double > & *x*, Matrix< double > & *F*, const void ∗ *data*)

5.33.4.8 int linearsolve_choice (const std::string & *input*)

5.33.4.9 bool linesearch_choice (const std::string & *input*)

5.33.4.10 void print2file_shark_header (SHARK_DATA ∗ *shark_dat*)

5.33.4.11 void print2file_shark_info (SHARK_DATA ∗ *shark_dat*)

5.33.4.12 void print2file_shark_results_new (SHARK_DATA ∗ *shark_dat*)

5.33.4.13 void print2file_shark_results_old (SHARK_DATA ∗ *shark_dat*)

5.33.4.14 int read_equilrxn (SHARK_DATA ∗ *shark_dat*)

5.33.4.15 int read_massbalance (SHARK_DATA ∗ *shark_dat*)

5.33.4.16 int read_options (SHARK_DATA ∗ *shark_dat*)

5.33.4.17 int read_scenario (SHARK_DATA ∗ *shark_dat*)

5.33.4.18 int read_species (SHARK_DATA ∗ *shark_dat*)

5.33.4.19 int read_unsteadyrxn (SHARK_DATA ∗ *shark_dat*)

5.33.4.20 int setup_SHARK_DATA (FILE ∗ *file,* int(∗)(const Matrix< double > &x, Matrix< double > &res, const void ∗data) *residual,* int(∗)(const Matrix< double > &x, Matrix< double > &gama, const void ∗data) *activity,* int(∗)(const Matrix< double > &r, Matrix< double > &p, const void ∗data) *precond,* SHARK_DATA ∗ *dat,* const void ∗ *activity_data,* const void ∗ *residual_data,* const void ∗ *precon_data,* const void ∗ *other_data*)

5.33.4.21 int SHARK (SHARK_DATA ∗ *shark_dat*)

5.33.4.22 int shark_add_customResidual (int *i,* double(∗)(const Matrix< double > &x, SHARK_DATA ∗shark_dat, const void ∗other_data) *other_res,* SHARK_DATA ∗ *shark_dat*)

5.33.4.23 int shark_energy_calculations (SHARK_DATA ∗ *shark_dat*)

5.33.4.24 int shark_executioner (SHARK_DATA ∗ *shark_dat*)

5.33.4.25 int shark_guess (SHARK_DATA ∗ *shark_dat*)

5.33.4.26 int shark_initial_conditions (SHARK_DATA ∗ *shark_dat*)

5.33.4.27 int shark_parameter_check (SHARK_DATA ∗ *shark_dat*)

5.33.4.28 int shark_pH_finder (SHARK_DATA ∗ *shark_dat*)

5.33.4.29 int shark_postprocesses (SHARK_DATA ∗ *shark_dat*)

5.33.4.30 int shark_preprocesses (SHARK_DATA ∗ *shark_dat*)

5.33.4.31 int shark_reset (SHARK_DATA ∗ *shark_dat*)

5.33.4.32 int shark_residual (const Matrix< double > & *x,* Matrix< double > & *F,* const void ∗ *data*)

5.33.4.33 int SHARK_SCENARIO (const char ∗ *yaml_input*)

5.33.4.34 int shark_solver (SHARK_DATA ∗ *shark_dat*)

5.33.4.35 int shark_temperature_calculations (SHARK_DATA ∗ *shark_dat*)

5.33.4.36 int SHARK_TESTS ( )

5.33.4.37 int shark_timestep_adapt (SHARK_DATA ∗ *shark_dat*)

5.33.4.38 int shark_timestep_const (SHARK_DATA ∗ *shark_dat*)

5.34 skua.cpp File Reference

```
#include "skua.h"
```

Functions

- void print2file_species_header (FILE ∗Output, SKUA_DATA ∗skua_dat, int i)
- void print2file_SKUA_time_header (FILE ∗Output, SKUA_DATA ∗skua_dat, int i)
- void print2file_SKUA_header (SKUA_DATA ∗skua_dat)
- void print2file_SKUA_results_old (SKUA_DATA ∗skua_dat)
- void print2file_SKUA_results_new (SKUA_DATA ∗skua_dat)
- double default_Dc (int i, int l, const void ∗data)
- double default_kf (int i, const void ∗data)
- double const_Dc (int i, int l, const void ∗data)
- double simple_darken_Dc (int i, int l, const void ∗data)
- double theoretical_darken_Dc (int i, int l, const void ∗data)
- double empirical_kf (int i, const void ∗data)
- double const_kf (int i, const void ∗data)
- int molefractionCheck (SKUA_DATA ∗skua_dat)
- int setup_SKUA_DATA (FILE ∗file, double(∗eval_Dc)(int i, int l, const void ∗user_data), double(∗eval_Kf)(int i, const void ∗user_data), const void ∗user_data, MIXED_GAS ∗gas_data, SKUA_DATA ∗skua_dat)
- int SKUA_Executioner (SKUA_DATA ∗skua_dat)
- int set_SKUA_ICs (SKUA_DATA ∗skua_dat)
- int set_SKUA_timestep (SKUA_DATA ∗skua_dat)
- int SKUA_preprocesses (SKUA_DATA ∗skua_dat)
- int set_SKUA_params (const void ∗user_data)
- int SKUA_postprocesses (SKUA_DATA ∗skua_dat)
- int SKUA_reset (SKUA_DATA ∗skua_dat)
- int SKUA (SKUA_DATA ∗skua_dat)
- int SKUA_CYCLE_TEST01 (SKUA_DATA ∗skua_dat)
- int SKUA_CYCLE_TEST02 (SKUA_DATA ∗skua_dat)
- int SKUA_LOW_TEST03 (SKUA_DATA ∗skua_dat)
- int SKUA_MID_TEST04 (SKUA_DATA ∗skua_dat)
- int SKUA_SCENARIOS (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate)
- int SKUA_TESTS ()

5.34.1 Function Documentation

5.34.1.1 double const_Dc (int *i,* int *l,* const void ∗ *data*)

5.34.1.2 double const_kf (int *i,* const void ∗ *data*)

5.34.1.3 double default_Dc (int *i,* int *l,* const void ∗ *data*)

5.34.1.4 double default_kf (int *i,* const void ∗ *data*)

5.34.1.5 double empirical_kf (int *i*, const void ∗ *data*)

5.34.1.6 int molefractionCheck (SKUA_DATA ∗ *skua_dat*)

5.34.1.7 void print2file_SKUA_header (SKUA_DATA ∗ *skua_dat*)

5.34.1.8 void print2file_SKUA_results_new (SKUA_DATA ∗ *skua_dat*)

5.34.1.9 void print2file_SKUA_results_old (SKUA_DATA ∗ *skua_dat*)

5.34.1.10 void print2file_SKUA_time_header (FILE ∗ *Output*, SKUA_DATA ∗ *skua_dat*, int *i*)

5.34.1.11 void print2file_species_header (FILE ∗ *Output*, SKUA_DATA ∗ *skua_dat*, int *i*)

5.34.1.12 int set_SKUA_ICs (SKUA_DATA ∗ *skua_dat*)

5.34.1.13 int set_SKUA_params (const void ∗ *user_data*)

5.34.1.14 int set_SKUA_timestep (SKUA_DATA ∗ *skua_dat*)

5.34.1.15 int setup_SKUA_DATA (FILE ∗ *file*, double(∗)(int i, int l, const void ∗user_data) *eval_Dc*, double(∗)(int i, const void ∗user_data) *eval_Kf*, const void ∗ *user_data*, MIXED_GAS ∗ *gas_data*, SKUA_DATA ∗ *skua_dat*)

5.34.1.16 double simple_darken_Dc (int *i*, int *l*, const void ∗ *data*)

5.34.1.17 int SKUA (SKUA_DATA ∗ *skua_dat*)

5.34.1.18 int SKUA_CYCLE_TEST01 (SKUA_DATA ∗ *skua_dat*)

5.34.1.19 int SKUA_CYCLE_TEST02 (SKUA_DATA ∗ *skua_dat*)

5.34.1.20 int SKUA_Executioner (SKUA_DATA ∗ *skua_dat*)

5.34.1.21 int SKUA_LOW_TEST03 (SKUA_DATA ∗ *skua_dat*)

5.34.1.22 int SKUA_MID_TEST04 (SKUA_DATA ∗ *skua_dat*)

5.34.1.23 int SKUA_postprocesses (SKUA_DATA ∗ *skua_dat*)

5.34.1.24 int SKUA_preprocesses (SKUA_DATA ∗ *skua_dat*)

5.34.1.25 int SKUA_reset (SKUA_DATA ∗ *skua_dat*)

5.34.1.26 int SKUA_SCENARIOS (const char ∗ *scene*, const char ∗ *sorbent*, const char ∗ *comp*, const char ∗ *sorbate*)

5.34.1.27 int SKUA_TESTS ( )

5.34.1.28 double theoretical_darken_Dc (int *i*, int *l*, const void ∗ *data*)

5.35 skua.h File Reference

```
#include "finch.h"
#include "magpie.h"
#include "egret.h"
```

Classes

- struct SKUA_PARAM
- struct SKUA_DATA

Macros

- #define SKUA_HPP_
- #define D_inf(Dref, Tref, B, p, T) (Dref ∗ pow(p+sqrt(DBL_EPSILON),(Tref/T)-B))
- #define D_o(Diff, E, T) (Diff ∗ exp(-E/(Rstd∗T)))
- #define D_c(Diff, phi) (Diff ∗ (1.0/((1.0+1.1E-6)-phi)))

Functions

- void print2file_species_header (FILE ∗Output, SKUA_DATA ∗skua_dat, int i)
- void print2file_SKUA_time_header (FILE ∗Output, SKUA_DATA ∗skua_dat, int i)
- void print2file_SKUA_header (SKUA_DATA ∗skua_dat)
- void print2file_SKUA_results_old (SKUA_DATA ∗skua_dat)
- void print2file_SKUA_results_new (SKUA_DATA ∗skua_dat)
- double default_Dc (int i, int l, const void ∗data)
- double default_kf (int i, const void ∗data)
- double const_Dc (int i, int l, const void ∗data)
- double simple_darken_Dc (int i, int l, const void ∗data)
- double theoretical_darken_Dc (int i, int l, const void ∗data)
- double empirical_kf (int i, const void ∗data)
- double const_kf (int i, const void ∗data)
- int molefractionCheck (SKUA_DATA ∗skua_dat)
- int setup_SKUA_DATA (FILE ∗file, double(∗eval_Dc)(int i, int l, const void ∗user_data), double(∗eval_Kf)(int i, const void ∗user_data), const void ∗user_data, MIXED_GAS ∗gas_data, SKUA_DATA ∗skua_dat)
- int SKUA_Executioner (SKUA_DATA ∗skua_dat)
- int set_SKUA_ICs (SKUA_DATA ∗skua_dat)
- int set_SKUA_timestep (SKUA_DATA ∗skua_dat)
- int SKUA_preprocesses (SKUA_DATA ∗skua_dat)
- int set_SKUA_params (const void ∗user_data)
- int SKUA_postprocesses (SKUA_DATA ∗skua_dat)
- int SKUA_reset (SKUA_DATA ∗skua_dat)
- int SKUA (SKUA_DATA ∗skua_dat)
- int SKUA_CYCLE_TEST01 (SKUA_DATA ∗skua_dat)
- int SKUA_CYCLE_TEST02 (SKUA_DATA ∗skua_dat)
- int SKUA_LOW_TEST03 (SKUA_DATA ∗skua_dat)
- int SKUA_MID_TEST04 (SKUA_DATA ∗skua_dat)
- int SKUA_SCENARIOS (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate)
- int SKUA_TESTS ()

5.35.1 Macro Definition Documentation

5.35.1.1 #define D_c(*Diff, phi*) (Diff ∗ (1.0/((1.0+1.1E-6)-phi)))

5.35.1.2 #define D_inf(*Dref, Tref, B, p, T*) (Dref ∗ pow(p+sqrt(DBL_EPSILON),(Tref/T)-B))

5.35.1.3 #define D_o(*Diff, E, T*) (Diff ∗ exp(-E/(Rstd∗T)))

5.35.1.4 #define SKUA_HPP_

5.35.2 Function Documentation

5.35.2.1 double const_Dc (int *i*, int *l*, const void * *data*)

5.35.2.2 double const_kf (int *i*, const void * *data*)

5.35.2.3 double default_Dc (int *i*, int *l*, const void * *data*)

5.35.2.4 double default_kf (int *i*, const void * *data*)

5.35.2.5 double empirical_kf (int *i*, const void * *data*)

5.35.2.6 int molefractionCheck (SKUA_DATA * *skua_dat*)

5.35.2.7 void print2file_SKUA_header (SKUA_DATA * *skua_dat*)

5.35.2.8 void print2file_SKUA_results_new (SKUA_DATA * *skua_dat*)

5.35.2.9 void print2file_SKUA_results_old (SKUA_DATA * *skua_dat*)

5.35.2.10 void print2file_SKUA_time_header (FILE * *Output*, SKUA_DATA * *skua_dat*, int *i*)

5.35.2.11 void print2file_species_header (FILE * *Output*, SKUA_DATA * *skua_dat*, int *i*)

5.35.2.12 int set_SKUA_ICs (SKUA_DATA * *skua_dat*)

5.35.2.13 int set_SKUA_params (const void * *user_data*)

5.35.2.14 int set_SKUA_timestep (SKUA_DATA * *skua_dat*)

5.35.2.15 int setup_SKUA_DATA (FILE * *file*, double(*)(int i, int l, const void *user_data) *eval_Dc*, double(*)(int i, const void *user_data) *eval_Kf*, const void * *user_data*, MIXED_GAS * *gas_data*, SKUA_DATA * *skua_dat*)

5.35.2.16 double simple_darken_Dc (int *i*, int *l*, const void * *data*)

5.35.2.17 int SKUA (SKUA_DATA * *skua_dat*)

5.35.2.18 int SKUA_CYCLE_TEST01 (SKUA_DATA * *skua_dat*)

5.35.2.19 int SKUA_CYCLE_TEST02 (SKUA_DATA * *skua_dat*)

5.35.2.20 int SKUA_Executioner (SKUA_DATA * *skua_dat*)

5.35.2.21 int SKUA_LOW_TEST03 (SKUA_DATA * *skua_dat*)

5.35.2.22 int SKUA_MID_TEST04 (SKUA_DATA * *skua_dat*)

5.35.2.23 int SKUA_postprocesses (SKUA_DATA * *skua_dat*)

5.35.2.24 int SKUA_preprocesses (SKUA_DATA * *skua_dat*)

5.35.2.25 int SKUA_reset (SKUA_DATA * *skua_dat*)

5.35.2.26 int SKUA_SCENARIOS (const char * *scene*, const char * *sorbent*, const char * *comp*, const char * *sorbate*)

5.35.2.27 int SKUA_TESTS ( )

5.35.2.28 double theoretical_darken_Dc (int *i*, int *l*, const void ∗ *data*)

5.36 skua_opt.cpp File Reference

```
#include "skua_opt.h"
```

Functions

- int SKUA_OPT_set_y (SKUA_OPT_DATA ∗skua_opt)
- int initial_guess_SKUA (SKUA_OPT_DATA ∗skua_opt)
- void eval_SKUA_Uptake (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int SKUA_OPTIMIZE (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate, const char ∗data)

5.36.1 Function Documentation

5.36.1.1 void eval_SKUA_Uptake (const double ∗ *par*, int *m_dat*, const void ∗ *data*, double ∗ *fvec*, int ∗ *info*)

5.36.1.2 int initial_guess_SKUA (SKUA_OPT_DATA ∗ *skua_opt*)

5.36.1.3 int SKUA_OPT_set_y (SKUA_OPT_DATA ∗ *skua_opt*)

5.36.1.4 int SKUA_OPTIMIZE (const char ∗ *scene*, const char ∗ *sorbent*, const char ∗ *comp*, const char ∗ *sorbate*, const char ∗ *data*)

5.37 skua_opt.h File Reference

```
#include "skua.h"
```

Classes

- struct SKUA_OPT_DATA

Functions

- int SKUA_OPT_set_y (SKUA_OPT_DATA ∗skua_opt)
- int initial_guess_SKUA (SKUA_OPT_DATA ∗skua_opt)
- void eval_SKUA_Uptake (const double ∗par, int m_dat, const void ∗data, double ∗fvec, int ∗info)
- int SKUA_OPTIMIZE (const char ∗scene, const char ∗sorbent, const char ∗comp, const char ∗sorbate, const char ∗data)

5.37.1 Function Documentation

5.37.1.1 void eval_SKUA_Uptake (const double ∗ *par*, int *m_dat*, const void ∗ *data*, double ∗ *fvec*, int ∗ *info*)

5.37.1.2 int initial_guess_SKUA (SKUA_OPT_DATA ∗ *skua_opt*)

5.37.1.3 int SKUA_OPT_set_y (SKUA_OPT_DATA ∗ *skua_opt*)

5.37.1.4 int SKUA_OPTIMIZE (const char ∗ *scene,* const char ∗ *sorbent,* const char ∗ *comp,* const char ∗ *sorbate,* const char ∗ *data*)

5.38 Trajectory.cpp File Reference

```
#include "Trajectory.h"
```

Functions

- double Magnetic_R (const Matrix< double > &dX, const Matrix< double > &dY, int i, double b, double mu_0, double chi_p, double M, double H0, double a)
- double Magnetic_T (const Matrix< double > &dX, const Matrix< double > &dY, int i, double b, double mu_0, double chi_p, double M, double H0, double a)
- double Grav_R (const Matrix< double > &dX, int i, double b, double rho_p, double rho_f)
- double Grav_T (const Matrix< double > &dX, int i, double b, double rho_p, double rho_f)
- double Van_R (const Matrix< double > &dX, const Matrix< double > &dY, int i, double Hamaker, double b, double a)
- double V_RAD (const Matrix< double > &dX, const Matrix< double > &dY, int i, double V0, double rho_f, double a, double eta)
- double V_THETA (const Matrix< double > &dX, const Matrix< double > &dY, int i, double V0, double rho_f, double a, double eta)
- double Brown_RAD (double n_rand, double m_rand, double sigma_n, double sigma_m)
- double Brown_THETA (double s_rand, double t_rand, double sigma_n, double sigma_m)
- int POLAR (Matrix< double > &POL, const Matrix< double > &dX, const Matrix< double > &dY, const void ∗data, int i)
- double RADIAL_FORCE (const Matrix< double > &POL, double eta, double b, double mp, double t, double a)
- double TANGENTIAL_FORCE (const Matrix< double > &POL, const Matrix< double > &dY, double eta, double b, double mp, double t, double a, int i)
- int CARTESIAN (const Matrix< double > &POL, Matrix< double > &H, const Matrix< double > &dY, double i, const void ∗data)
- int DISPLACEMENT (Matrix< double > &dX, Matrix< double > &dY, const Matrix< double > &H, int i)
- int LOCATION (const Matrix< double > &dY, const Matrix< double > &dX, Matrix< double > &X, Matrix< double > &Y, int i)
- double Removal_Efficiency (double Sum_Cap, const void ∗data)
- int Trajectory_SetupConstants (TRAJECTORY_DATA ∗dat)
- int Number_Generator (TRAJECTORY_DATA ∗dat)
- int Run_Trajectory ()

5.38.1 Function Documentation

5.38.1.1 double Brown_RAD (double *n_rand,* double *m_rand,* double *sigma_n,* double *sigma_m*)

5.38.1.2 double Brown_THETA (double *s_rand,* double *t_rand,* double *sigma_n,* double *sigma_m*)

5.38.1.3 int CARTESIAN (const **Matrix**< double > & *POL,* **Matrix**< double > & *H,* const **Matrix**< double > & *dY,* double *i,* const void ∗ *data*)

5.38.1.4 int DISPLACEMENT (**Matrix**< double > & *dX,* **Matrix**< double > & *dY,* const **Matrix**< double > & *H,* int *i*)

5.38.1.5 double Grav_R (const **Matrix**< double > & *dX,* int *i,* double *b,* double *rho_p,* double *rho_f*)

5.38.1.6 double Grav_T (const **Matrix**< double > & *dX*, int *i*, double *b*, double *rho_p*, double *rho_f*)

5.38.1.7 int LOCATION (const **Matrix**< double > & *dY*, const **Matrix**< double > & *dX*, **Matrix**< double > & *X*, **Matrix**< double > & *Y*, int *i*)

5.38.1.8 double Magnetic_R (const **Matrix**< double > & *dX*, const **Matrix**< double > & *dY*, int *i*, double *b*, double *mu_0*, double *chi_p*, double *M*, double *H0*, double *a*)

5.38.1.9 double Magnetic_T (const **Matrix**< double > & *dX*, const **Matrix**< double > & *dY*, int *i*, double *b*, double *mu_0*, double *chi_p*, double *M*, double *H0*, double *a*)

5.38.1.10 int Number_Generator (**TRAJECTORY_DATA** ∗ *dat*)

5.38.1.11 int POLAR (**Matrix**< double > & *POL*, const **Matrix**< double > & *dX*, const **Matrix**< double > & *dY*, const void ∗ *data*, int *i*)

5.38.1.12 double RADIAL_FORCE (const **Matrix**< double > & *POL*, double *eta*, double *b*, double *mp*, double *t*, double *a*)

5.38.1.13 double Removal_Efficiency (double *Sum_Cap*, const void ∗ *data*)

5.38.1.14 int Run_Trajectory ( )

5.38.1.15 double TANGENTIAL_FORCE (const **Matrix**< double > & *POL*, const **Matrix**< double > & *dY*, double *eta*, double *b*, double *mp*, double *t*, double *a*, int *i*)

5.38.1.16 int Trajectory_SetupConstants (**TRAJECTORY_DATA** ∗ *dat*)

5.38.1.17 double V_RAD (const **Matrix**< double > & *dX*, const **Matrix**< double > & *dY*, int *i*, double *V0*, double *rho_f*, double *a*, double *eta*)

5.38.1.18 double V_THETA (const **Matrix**< double > & *dX*, const **Matrix**< double > & *dY*, int *i*, double *V0*, double *rho_f*, double *a*, double *eta*)

5.38.1.19 double Van_R (const **Matrix**< double > & *dX*, const **Matrix**< double > & *dY*, int *i*, double *Hamaker*, double *b*, double *a*)

5.39 Trajectory.h File Reference

```
#include "macaw.h"
#include <random>
#include <chrono>
```

Classes

- struct TRAJECTORY_DATA

Functions

- double Magnetic_R (const Matrix< double > &dX, const Matrix< double > &dY, int i, double b, double mu_0, double chi_p, double M, double H0, double a)
- double Magnetic_T (const Matrix< double > &dX, const Matrix< double > &dY, int i, double b, double mu_0, double chi_p, double M, double H0, double a)
- double Grav_R (const Matrix< double > &dX, int i, double b, double rho_p, double rho_f)
- double Grav_T (const Matrix< double > &dX, int i, double b, double rho_p, double rho_f)

- double Van_R (const Matrix< double > &dX, const Matrix< double > &dY, int i, double Hamaker, double b, double a)
- double V_RAD (const Matrix< double > &dX, const Matrix< double > &dY, int i, double V0, double rho_f, double a, double eta)
- double V_THETA (const Matrix< double > &dX, const Matrix< double > &dY, int i, double V0, double rho_f, double a, double eta)
- double Brown_RAD (double n_rand, double m_rand, double sigma_n, double sigma_m)
- double Brown_THETA (double s_rand, double t_rand, double sigma_n, double sigma_m)
- int POLAR (Matrix< double > &POL, const Matrix< double > &dX, const Matrix< double > &dY, const void ∗data, int i)
- double RADIAL_FORCE (const Matrix< double > &POL, double eta, double b, double mp, double t, double a)
- double TANGENTIAL_FORCE (const Matrix< double > &POL, const Matrix< double > &dY, double eta, double b, double mp, double t, double a, int i)
- int CARTESIAN (const Matrix< double > &POL, Matrix< double > &H, const Matrix< double > &dY, double i, const void ∗data)
- int DISPLACEMENT (Matrix< double > &dX, Matrix< double > &dY, const Matrix< double > &H, int i)
- int LOCATION (const Matrix< double > &dY, const Matrix< double > &dX, Matrix< double > &X, Matrix< double > &Y, int i)
- double Removal_Efficiency (double Sum_Cap, const void ∗data)
- int Trajectory_SetupConstants (TRAJECTORY_DATA ∗dat)
- int Number_Generator (TRAJECTORY_DATA ∗dat)
- int Run_Trajectory ()

5.39.1 Function Documentation

5.39.1.1 double Brown_RAD (double *n_rand,* double *m_rand,* double *sigma_n,* double *sigma_m*)

5.39.1.2 double Brown_THETA (double *s_rand,* double *t_rand,* double *sigma_n,* double *sigma_m*)

5.39.1.3 int CARTESIAN (const Matrix< double > & *POL,* Matrix< double > & *H,* const Matrix< double > & *dY,* double *i,* const void ∗ *data*)

5.39.1.4 int DISPLACEMENT (Matrix< double > & *dX,* Matrix< double > & *dY,* const Matrix< double > & *H,* int *i*)

5.39.1.5 double Grav_R (const Matrix< double > & *dX,* int *i,* double *b,* double *rho_p,* double *rho_f*)

5.39.1.6 double Grav_T (const Matrix< double > & *dX,* int *i,* double *b,* double *rho_p,* double *rho_f*)

5.39.1.7 int LOCATION (const Matrix< double > & *dY,* const Matrix< double > & *dX,* Matrix< double > & *X,* Matrix< double > & *Y,* int *i*)

5.39.1.8 double Magnetic_R (const Matrix< double > & *dX,* const Matrix< double > & *dY,* int *i,* double *b,* double *mu_0,* double *chi_p,* double *M,* double *H0,* double *a*)

5.39.1.9 double Magnetic_T (const Matrix< double > & *dX,* const Matrix< double > & *dY,* int *i,* double *b,* double *mu_0,* double *chi_p,* double *M,* double *H0,* double *a*)

5.39.1.10 int Number_Generator (TRAJECTORY_DATA ∗ *dat*)

5.39.1.11 int POLAR (Matrix< double > & *POL,* const Matrix< double > & *dX,* const Matrix< double > & *dY,* const void ∗ *data,* int *i*)

5.39.1.12 double RADIAL_FORCE (const Matrix< double > & *POL,* double *eta,* double *b,* double *mp,* double *t,* double *a*)

5.39.1.13 double Removal_Efficiency (double *Sum_Cap*, const void ∗ *data*)

5.39.1.14 int Run_Trajectory ( )

5.39.1.15 double TANGENTIAL_FORCE (const Matrix< double > & *POL*, const Matrix< double > & *dY*, double *eta*, double *b*, double *mp*, double *t*, double *a*, int *i*)

5.39.1.16 int Trajectory_SetupConstants (TRAJECTORY_DATA ∗ *dat*)

5.39.1.17 double V_RAD (const Matrix< double > & *dX*, const Matrix< double > & *dY*, int *i*, double *V0*, double *rho_f*, double *a*, double *eta*)

5.39.1.18 double V_THETA (const Matrix< double > & *dX*, const Matrix< double > & *dY*, int *i*, double *V0*, double *rho_f*, double *a*, double *eta*)

5.39.1.19 double Van_R (const Matrix< double > & *dX*, const Matrix< double > & *dY*, int *i*, double *Hamaker*, double *b*, double *a*)

5.40 ui.cpp File Reference

User Interface for Ecosystem.

```
#include "ui.h"
```

Functions

- void aui_help ()

  Function to display help for Advanced User Interface.
- void bui_help ()

  Function to display help for Basic User Interface.
- std::string allLower (const std::string &input)

  Function to return an all lower case string based on the passed argument.
- bool exit (const std::string &input)

  Function returns true if user requests exit.
- bool help (const std::string &input)

  Function returns trun if the user requests help.
- bool version (const std::string &input)

  Function returns true if user requests to know the executable version.
- bool test (const std::string &input)

  Function returns true if user requests to run a test.
- bool exec (const std::string &input)

  Function returns true if the user requests to run a simulation/executable.
- bool path (const std::string &input)

  Function returns true if the user indicates that input files share a common path.
- bool input (const std::string &input)

  Function returns true if the user indicates that the next arguments are input files.
- bool valid_test_string (const std::string &input, UI_DATA ∗ui_dat)

  Function returns true if the user gave a valid test option.
- bool valid_exec_string (const std::string &input, UI_DATA ∗ui_dat)

  Function returns true if the user gave a valid execution option.
- int number_files (UI_DATA ∗ui_dat)

  Function returns the number of expected input files for the user's run option.

- bool valid_addon_options (UI_DATA ∗ui_dat)

  Function returns true if the user has choosen a valid additional runtime option.
- void display_help (UI_DATA ∗ui_dat)

  Function to call the appropriate help menu based on type of interface.
- void display_version (UI_DATA ∗ui_dat)

  Function to display ecosystem version information to the console.
- int invalid_input (int count, int max)

  Function returns a CONTINUE or EXIT when invalid input is given.
- bool valid_input_main (UI_DATA ∗ui_dat)

  Function returns true if user gave valid input in Basic UI.
- bool valid_input_tests (UI_DATA ∗ui_dat)

  Function returns true if user gave a valid test function to run.
- bool valid_input_execute (UI_DATA ∗ui_dat)

  Function returns true if user gave a valid executable function to run.
- int test_loop (UI_DATA ∗ui_dat)

  Function that loops the Basic UI until a valid test option was selected.
- int exec_loop (UI_DATA ∗ui_dat)

  Function that loops the Basic UI until a valid executable option was selected.
- int run_test (UI_DATA ∗ui_dat)

  Function will call the user requested test function.
- int run_exec (UI_DATA ∗ui_dat)

  Function will call the user requested executable function.
- int run_executable (int argc, const char ∗argv[ ])

  Function called by the main and runs both user interfaces for the program.

5.40.1 Detailed Description

User Interface for Ecosystem. ui.h

Author

Austin Ladshaw

Version

0.0 beta

Date

08/25/2015

Copyright

This software was designed and built at the Georgia Institute of Technology by Austin Ladshaw for PhD research in the area of adsorption and surface science. Copyright (c) 2015, all rights reserved.

5.40.2 Function Documentation

5.40.2.1 std::string allLower (const std::string & *input*)

Function to return an all lower case string based on the passed argument.

This function will copy the input paramter and convert that copy to all lower case. The copy is then returned and can be checked against valid or allowed strings.

Parameters

input	string to copy and convert to lower case

5.40.2.2 void aui_help ( )

Function to display help for Advanced User Interface.

The Advanved User Interface help screen is accessed by including run option -h or –help when executing the program from command line.

5.40.2.3 void bui_help ( )

Function to display help for Basic User Interface.

The Basic User Interface help screen is accessed by running the executable, then typing "help" at any point during the console prompts. Exception to this occurs when the console prompts you to provide input files for your choosen routine. In this circumstance, the executable always assumes that what the user types in will be an input file.

5.40.2.4 void display_help (UI_DATA ∗ *ui_dat*)

Function to call the appropriate help menu based on type of interface.

This function looks at the ui_dat structure and the user's OS files to determine what help menu to display and how to display it. There are two different types of help menus that can be displayed: (i) Advanced Help and (ii) Basic Help. Additionally, this function checks the OS file system for the existence of installed help files. If it finds those files, then it instructs the command terminal to read the contents of those files with the "less" command. Otherwise, it will just print the appropriate help menu to the console window.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.5 void display_version (UI_DATA ∗ *ui_dat*)

Function to display ecosystem version information to the console.

This function will check the ui_dat structure to see which type of interface the user is using, then print out the version information for the executable being run.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.6 bool exec (const std::string & *input*)

Function returns true if the user requests to run a simulation/executable.

This function will check the input string for "-e" or "&ndash;execute" and determine whether or not the user requests to run an ecosystem executable function.

Parameters

input	input string the user gives to the console

5.40.2.7 int exec_loop (UI_DATA * *ui_dat*)

Function that loops the Basic UI until a valid executable option was selected.

This function loops the Basic UI menu for running an executable until a valid executable is selected by the user. If a valid executable is not selected, and the maximum number of loops has been reached, then this function will cause the program to force quit.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.8 bool exit (const std::string & *input*)

Function returns true if user requests exit.

This function will check the input string for "exit" or "quit" and terminate the executable. Only checked if using the Basic User Interface.

Parameters

input	input string user gives to the console

5.40.2.9 bool help (const std::string & *input*)

Function returns trun if the user requests help.

This function will check the input string for "help", "-h", or "&ndash;help" and will tell the executable to display the help menu. The help menu that gets displayed depends on how the executable was run to begin with.

Parameters

input	input string user gives to the console

5.40.2.10 bool input (const std::string & *input*)

Function returns true if the user indicates that the next arguments are input files.

This function will check the input string for "-i" or "&ndash;input" and determine whether or not the user's next arguments are input files for a specific simulation. Only used in Advanced User Interface.

Parameters

input	input string the user gives to the console

5.40.2.11 int invalid_input (int *count,* int *max*)

Function returns a CONTINUE or EXIT when invalid input is given.

This function looks at the current count and the max iterations and determines whether or not to force the executable to terminate. If the user provides too many incorrect options during the Basic User Interface, then the executable will force quit.

Parameters

count	number of times the user has provided a bad option
max	maximum allowable bad options before force quit

5.40.2.12 int number_files (UI_DATA * *ui_dat*)

Function returns the number of expected input files for the user's run option.

This function will check the option variable in the ui_dat structure to determine the number of input files that is expected to be given. Running different executable functions in ecosystem may require various number of input files.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.13 bool path (const std::string & *input*)

Function returns true if the user indicates that input files share a common path.

This function will check the input string for "-p" or "&ndash;path" and determine whether or not the user will give a common path to all input files needed for the specified simulation. Only used in Advanced User Interface.

Parameters

input	input string the user gives to the console

5.40.2.14 int run_exec (UI_DATA * *ui_dat*)

Function will call the user requested executable function.

This function checks the option variable of the ui_dat structure and runs the corresponding executable function.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.15 int run_executable (int *argc*, const char * *argv[]*)

Function called by the main and runs both user interfaces for the program.

This function is called in the main.cpp file and passes the console arguments given at run time.

Parameters

argc	number of arguments provided by the user at the time of execution
argv	list of C-strings that was provided by the user at the time of execution

5.40.2.16 int run_test (UI_DATA * *ui_dat*)

Function will call the user requested test function.

This function checks the option variable of the ui_dat structure and runs the corresponding test function.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.17 bool test (const std::string & *input*)

Function returns true if user requests to run a test.

This function will check the input string for "-t" or "&ndash;test" and determine whether or not the user requests to run an ecosystem test function.

Parameters

input	input string user gives to the console

5.40.2.18 int test_loop (UI_DATA ∗ *ui_dat*)

Function that loops the Basic UI until a valid test option was selected.

This function loops the Basic UI menu for running a test until a valid test is selected by the user. If a valid test is not selected, and the maximum number of loops has been reached, then this function will cause the program to force quit.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.19 bool valid_addon_options (UI_DATA ∗ *ui_dat*)

Function returns true if the user has choosen a valid additional runtime option.

This function will check all additional input options in the user_input variable of ui_dat to determine if the user requests any additional options during runtime. Valid additional options are -p or –path and -i or –input.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.20 bool valid_exec_string (const std::string & *input,* UI_DATA ∗ *ui_dat*)

Function returns true if the user gave a valid execution option.

This function will check the input string given by the user and determine whether that string denotes a valid execution option. Then, it will mark the option variable in ui_dat with the appropriate option from the valid_options enum.

Parameters

input	input string the user gives to the console
ui_dat	pointer to the data structure for the ui object

5.40.2.21 bool valid_input_execute (UI_DATA ∗ *ui_dat*)

Function returns true if user gave a valid executable function to run.

This function checks the user_input argument of ui_dat for a valid executable option. If no valid executable was given, then this function returns false.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.22 bool valid_input_main (UI_DATA ∗ *ui_dat*)

Function returns true if user gave valid input in Basic UI.

This function is only called if the user is running the Basic UI. It checks the given console argument stored in user_input of ui_dat for a valid option. If no valid option is given, then this function returns false.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.23 bool valid_input_tests (UI_DATA ∗ *ui_dat*)

Function returns true if user gave a valid test function to run.

This function checks the user_input argument of ui_dat for a valid test option. If no valid test was given, then this function returns false.

Parameters

ui_dat	pointer to the data structure for the ui object

5.40.2.24 bool valid_test_string (const std::string & *input*, UI_DATA ∗ *ui_dat*)

Function returns true if the user gave a valid test option.

This function will check the input string given by the user and determine whether that string denotes a valid test. Then, it will mark the option variable in ui_dat with the appropriate option from the valid_options enum.

Parameters

input	input string the user gives to the console
ui_dat	pointer to the data structure for the ui object

5.40.2.25 bool version (const std::string & *input*)

Function returns true if user requests to know the executable version.

This function will check the input string for "version", "-v", or "&ndash;version" and will tell the executable to display version information about the executable.

Parameters

input	input string user gives to the console

5.41 ui.h File Reference

User Interface for Ecosystem.

```
#include <fstream>
#include <string>
#include <iostream>
#include "error.h"
#include "yaml_wrapper.h"
#include "flock.h"
#include "school.h"
#include "sandbox.h"
#include "Trajectory.h"
```

Classes

- struct UI_DATA

  Data structure holding the UI arguments.

Macros

- #define UI_HPP_
- #define ECO_VERSION "0.0 alpha"

  Macro expansion for executable current version number.
- #define ECO_EXECUTABLE "eco0"

  Macro expansion for executable current name.

Enumerations

- enum valid_options {
  TEST, EXECUTE, EXIT, CONTINUE,
  HELP, dogfish, eel, egret,
  finch, lark, macaw, mola,
  monkfish, sandbox, scopsowl, shark,
  skua, gsta_opt, magpie, scops_opt,
  skua_opt, trajectory }

  Valid options available upon execution of the code.

Functions

- void aui_help ()

  Function to display help for Advanced User Interface.
- void bui_help ()

  Function to display help for Basic User Interface.
- std::string allLower (const std::string &input)

  Function to return an all lower case string based on the passed argument.
- bool exit (const std::string &input)

  Function returns true if user requests exit.
- bool help (const std::string &input)

  Function returns trun if the user requests help.
- bool version (const std::string &input)

  Function returns true if user requests to know the executable version.
- bool test (const std::string &input)

  Function returns true if user requests to run a test.
- bool exec (const std::string &input)

Function returns true if the user requests to run a simulation/executable.
- bool path (const std::string &input)

  Function returns true if the user indicates that input files share a common path.
- bool input (const std::string &input)

  Function returns true if the user indicates that the next arguments are input files.
- bool valid_test_string (const std::string &input, UI_DATA ∗ui_dat)

  Function returns true if the user gave a valid test option.
- bool valid_exec_string (const std::string &input, UI_DATA ∗ui_dat)

  Function returns true if the user gave a valid execution option.
- int number_files (UI_DATA ∗ui_dat)

  Function returns the number of expected input files for the user's run option.
- bool valid_addon_options (UI_DATA ∗ui_dat)

  Function returns true if the user has choosen a valid additional runtime option.
- void display_help (UI_DATA ∗ui_dat)

  Function to call the appropriate help menu based on type of interface.
- void display_version (UI_DATA ∗ui_dat)

  Function to display ecosystem version information to the console.
- int invalid_input (int count, int max)

  Function returns a CONTINUE or EXIT when invalid input is given.
- bool valid_input_main (UI_DATA ∗ui_dat)

  Function returns true if user gave valid input in Basic UI.
- bool valid_input_tests (UI_DATA ∗ui_dat)

  Function returns true if user gave a valid test function to run.
- bool valid_input_execute (UI_DATA ∗ui_dat)

  Function returns true if user gave a valid executable function to run.
- int test_loop (UI_DATA ∗ui_dat)

  Function that loops the Basic UI until a valid test option was selected.
- int exec_loop (UI_DATA ∗ui_dat)

  Function that loops the Basic UI until a valid executable option was selected.
- int run_test (UI_DATA ∗ui_dat)

  Function will call the user requested test function.
- int run_exec (UI_DATA ∗ui_dat)

  Function will call the user requested executable function.
- int run_executable (int argc, const char ∗argv[ ])

  Function called by the main and runs both user interfaces for the program.

5.41.1 Detailed Description

User Interface for Ecosystem. ui.cpp

These routines define how the user will interface with the software

Author

Austin Ladshaw

Version

0.0 beta

Date

08/25/2015

Copyright

5.41.2 Macro Definition Documentation

5.41.2.1 #define ECO_EXECUTABLE "eco0"

Macro expansion for executable current name.

5.41.2.2 #define ECO_VERSION "0.0 alpha"

Macro expansion for executable current version number.

5.41.2.3 #define UI_HPP_

5.41.3 Enumeration Type Documentation

5.41.3.1 enum valid_options

Valid options available upon execution of the code.

Enumeration of valid options for executing the ecosystem code. More options become available as the code updates. Some options that appear here may not be viewable in the "help" screen of the executable. Those options are hidden, but are still valid entries.

Enumerator

TEST
EXECUTE
EXIT
CONTINUE
HELP
dogfish
eel
egret
finch
lark
macaw
mola
monkfish
sandbox
scopsowl
shark
skua
gsta_opt
magpie
scops_opt
skua_opt
trajectory

5.41.4 Function Documentation

5.41.4.1 std::string allLower (const std::string & *input*)

Function to return an all lower case string based on the passed argument.

This function will copy the input paramter and convert that copy to all lower case. The copy is then returned and can be checked against valid or allowed strings.

Parameters

input	string to copy and convert to lower case

5.41.4.2 void aui_help ( )

Function to display help for Advanced User Interface.

The Advanved User Interface help screen is accessed by including run option -h or –help when executing the program from command line.

5.41.4.3 void bui_help ( )

Function to display help for Basic User Interface.

The Basic User Interface help screen is accessed by running the executable, then typing "help" at any point during the console prompts. Exception to this occurs when the console prompts you to provide input files for your choosen routine. In this circumstance, the executable always assumes that what the user types in will be an input file.

5.41.4.4 void display_help (UI_DATA ∗ *ui_dat*)

Function to call the appropriate help menu based on type of interface.

This function looks at the ui_dat structure and the user's OS files to determine what help menu to display and how to display it. There are two different types of help menus that can be displayed: (i) Advanced Help and (ii) Basic Help. Additionally, this function checks the OS file system for the existence of installed help files. If it finds those files, then it instructs the command terminal to read the contents of those files with the "less" command. Otherwise, it will just print the appropriate help menu to the console window.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.5 void display_version (UI_DATA ∗ *ui_dat*)

Function to display ecosystem version information to the console.

This function will check the ui_dat structure to see which type of interface the user is using, then print out the version information for the executable being run.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.6 bool exec (const std::string & *input*)

Function returns true if the user requests to run a simulation/executable.

This function will check the input string for "-e" or "&ndash;execute" and determine whether or not the user requests to run an ecosystem executable function.

Parameters

input	input string the user gives to the console

5.41.4.7 int exec_loop (UI_DATA ∗ *ui_dat*)

Function that loops the Basic UI until a valid executable option was selected.

This function loops the Basic UI menu for running an executable until a valid executable is selected by the user. If a valid executable is not selected, and the maximum number of loops has been reached, then this function will cause the program to force quit.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.8 bool exit (const std::string & *input*)

Function returns true if user requests exit.

This function will check the input string for "exit" or "quit" and terminate the executable. Only checked if using the Basic User Interface.

Parameters

input	input string user gives to the console

5.41.4.9 bool help (const std::string & *input*)

Function returns trun if the user requests help.

This function will check the input string for "help", "-h", or "&ndash;help" and will tell the executable to display the help menu. The help menu that gets displayed depends on how the executable was run to begin with.

Parameters

input	input string user gives to the console

5.41.4.10 bool input (const std::string & *input*)

Function returns true if the user indicates that the next arguments are input files.

This function will check the input string for "-i" or "&ndash;input" and determine whether or not the user's next arguments are input files for a specific simulation. Only used in Advanced User Interface.

Parameters

input	input string the user gives to the console

5.41.4.11 int invalid_input (int *count,* int *max*)

Function returns a CONTINUE or EXIT when invalid input is given.

This function looks at the current count and the max iterations and determines whether or not to force the executable to terminate. If the user provides too many incorrect options during the Basic User Interface, then the executable will force quit.

Parameters

count	number of times the user has provided a bad option
max	maximum allowable bad options before force quit

5.41.4.12 int number_files (UI_DATA ∗ *ui_dat*)

Function returns the number of expected input files for the user's run option.

This function will check the option variable in the ui_dat structure to determine the number of input files that is expected to be given. Running different executable functions in ecosystem may require various number of input files.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.13 bool path (const std::string & *input*)

Function returns true if the user indicates that input files share a common path.

This function will check the input string for "-p" or "&ndash;path" and determine whether or not the user will give a common path to all input files needed for the specified simulation. Only used in Advanced User Interface.

Parameters

input	input string the user gives to the console

5.41.4.14 int run_exec (UI_DATA ∗ *ui_dat*)

Function will call the user requested executable function.

This function checks the option variable of the ui_dat structure and runs the corresponding executable function.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.15 int run_executable (int *argc,* const char ∗ *argv[ ]*)

Function called by the main and runs both user interfaces for the program.

This function is called in the main.cpp file and passes the console arguments given at run time.

Parameters

argc	number of arguments provided by the user at the time of execution
argv	list of C-strings that was provided by the user at the time of execution

5.41.4.16 int run_test (UI_DATA ∗ *ui_dat*)

Function will call the user requested test function.

This function checks the option variable of the ui_dat structure and runs the corresponding test function.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.17 bool test (const std::string & *input*)

Function returns true if user requests to run a test.

This function will check the input string for "-t" or "&ndash;test" and determine whether or not the user requests to run an ecosystem test function.

Parameters

input	input string user gives to the console

5.41.4.18 int test_loop (UI_DATA ∗ *ui_dat*)

Function that loops the Basic UI until a valid test option was selected.

This function loops the Basic UI menu for running a test until a valid test is selected by the user. If a valid test is not selected, and the maximum number of loops has been reached, then this function will cause the program to force quit.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.19 bool valid_addon_options (UI_DATA ∗ *ui_dat*)

Function returns true if the user has choosen a valid additional runtime option.

This function will check all additional input options in the user_input variable of ui_dat to determine if the user requests any additional options during runtime. Valid additional options are -p or –path and -i or –input.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.20 bool valid_exec_string (const std::string & *input,* UI_DATA ∗ *ui_dat*)

Function returns true if the user gave a valid execution option.

This function will check the input string given by the user and determine whether that string denotes a valid execution option. Then, it will mark the option variable in ui_dat with the appropriate option from the valid_options enum.

Parameters

input	input string the user gives to the console
ui_dat	pointer to the data structure for the ui object

5.41.4.21 bool valid_input_execute (UI_DATA ∗ *ui_dat*)

Function returns true if user gave a valid executable function to run.

This function checks the user_input argument of ui_dat for a valid executable option. If no valid executable was given, then this function returns false.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.22 bool valid_input_main (UI_DATA ∗ *ui_dat*)

Function returns true if user gave valid input in Basic UI.

This function is only called if the user is running the Basic UI. It checks the given console argument stored in user_input of ui_dat for a valid option. If no valid option is given, then this function returns false.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.23 bool valid_input_tests (UI_DATA ∗ *ui_dat*)

Function returns true if user gave a valid test function to run.

This function checks the user_input argument of ui_dat for a valid test option. If no valid test was given, then this function returns false.

Parameters

ui_dat	pointer to the data structure for the ui object

5.41.4.24 bool valid_test_string (const std::string & *input,* UI_DATA ∗ *ui_dat*)

Function returns true if the user gave a valid test option.

This function will check the input string given by the user and determine whether that string denotes a valid test. Then, it will mark the option variable in ui_dat with the appropriate option from the valid_options enum.

Parameters

input	input string the user gives to the console
ui_dat	pointer to the data structure for the ui object

5.41.4.25 bool version (const std::string & *input*)

Function returns true if user requests to know the executable version.

This function will check the input string for "version", "-v", or "&ndash;version" and will tell the executable to display version information about the executable.

Parameters

input	input string user gives to the console

5.42 yaml_wrapper.cpp File Reference

```
#include "yaml_wrapper.h"
```

Functions

- int YAML_WRAPPER_TESTS ()
- int YAML_CPP_TEST (const char ∗file)

5.42.1 Function Documentation

5.42.1.1 int YAML_CPP_TEST (const char ∗ *file*)

5.42.1.2 int YAML_WRAPPER_TESTS ( )

5.43 yaml_wrapper.h File Reference

```
#include "yaml.h"
#include "error.h"
#include <map>
#include <string>
#include <iostream>
#include <utility>
#include <stdexcept>
```

Classes

- class ValueTypePair
- class KeyValueMap
- class SubHeader
- class Header
- class Document
- class YamlWrapper
- class yaml_cpp_class

Typedefs

- typedef enum data_type data_type
- typedef enum header_state header_state

Enumerations

- enum data_type {
  STRING, BOOLEAN, DOUBLE, INT,
  UNKNOWN }
- enum header_state { ANCHOR, ALIAS, NONE }

Functions

- int YAML_WRAPPER_TESTS ()
- int YAML_CPP_TEST (const char ∗file)

5.43.1 Typedef Documentation

5.43.1.1 typedef enum **data_type data_type**

5.43.1.2 typedef enum **header_state header_state**

5.43.2 Enumeration Type Documentation

5.43.2.1 enum **data_type**

Enumerator

STRING

BOOLEAN

DOUBLE

INT

UNKNOWN

5.43.2.2 enum **header_state**

Enumerator

ANCHOR

ALIAS

NONE

5.43.3 Function Documentation

5.43.3.1 int YAML_CPP_TEST (const char ∗ *file*)

5.43.3.2 int YAML_WRAPPER_TESTS ( )

Index