OMicroN

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 Application Class Reference

Manages the overall application flow.

`#include <application.h>`

Public Member Functions

- Application (const std::string &configFile)
  Constructs an Application object with a configuration file.
- void run ()
  Runs the application.

3.1.1 Detailed Description

Manages the overall application flow.

This class handles the initialization of settings, running the simulation based on these settings, initializing the log file, managing the microstructure class and performing all the simulation steps.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Application()

```
Application::Application (
            const std::string & inputFile )
```

Constructs an Application object with a configuration file.

Constructs an Application object with a user input (parameter) file.

Parameters

configFile	Path to the configuration file.
inputFile	Path to the input (parameter) file.

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

- src/application.h
- src/application.cpp

3.2 Microstructure Class Reference

Manages the microstructure simulation for solid state transformations (phase transformations, recrystallization, grain growth) and solute redistribution (partitioning, diffusion, trapping to defects).

`#include <microstructure.h>`

Public Member Functions

- Microstructure (const UserSettings &userSettings)

  Constructs a Microstructure object with user settings.
- bool IsStillSimulating () const

  Checks if the simulation is still running.
- void printMicrostructureParameters () const

  Prints the microstructure parameters.
- void SimulationStep ()

  Executes a simulation step.
- void AddInterfaceIndex (int index)

  Adds an index to the set of interface indices.
- void RemoveInterfaceIndex (int index)

  Removes an index from the set of interface indices.
- void AddInterphaseIndex (int index)

  Adds an index to the set of interphase indices.
- void RemoveInterphaseIndex (int index)

  Removes an index from the set of interphase indices.
- const std::unordered_set< int > & GetInterfaceIndices () const

  Gets the set of interface indices.
- const std::unordered_set< int > & GetInterphaseIndices () const

  Gets the set of interphase indices.

3.2.1 Detailed Description

Manages the microstructure simulation for solid state transformations (phase transformations, recrystallization, grain growth) and solute redistribution (partitioning, diffusion, trapping to defects).

This class handles the initialization, simulation steps, and state management for the microstructure in the OMicroN simulation program. It interacts with all other classes, setting state variables and parameters, and retrieving information from instances.

Additionally, it is the main class responsible for writing and exporting simulation outputs.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Microstructure()

```
Microstructure::Microstructure (
            const UserSettings & userSettings )
```

Constructs a Microstructure object with user settings.

Constructor for the Microstructure class.

Parameters

userSettings	Reference to user settings.

Initializes the microstructure simulation with default or user-specified parameters. This function sets up necessary data structures and prepares the class for simulation steps.

Parameters

inputFile	The path to the input file containing user-defined settings.

3.2.3 Member Function Documentation

3.2.3.1 AddInterfaceIndex()

```
void Microstructure::AddInterfaceIndex (
            int index )
```

Adds an index to the set of interface indices.

Updates set of (sub)grain boundary cells by adding the cell of index, and updates cell's (index) identity regarding whether it is interface.

Parameters

index	Index to add.
index	the cell's index that will now be recognized as (sub)grain boundary cell

3.2.3.2 AddInterphaseIndex()

```
void Microstructure::AddInterphaseIndex (
```

```
            int index )
```

Adds an index to the set of interphase indices.

Updates set of phase boundary cells by adding the cell of index, and updates cell's (index) identity regarding whether it is interface.

Parameters

index	Index to add.
index	the cell's index that will now be recognized as phase cell

3.2.3.3 GetInterfaceIndices()

```
const std::unordered_set<int>& Microstructure::GetInterfaceIndices ( ) const  [inline]
```

Gets the set of interface indices.

Returns

Const reference to the set of interface indices.

3.2.3.4 GetInterphaseIndices()

```
const std::unordered_set<int>& Microstructure::GetInterphaseIndices ( ) const  [inline]
```

Gets the set of interphase indices.

Returns

Const reference to the set of interphase indices.

3.2.3.5 IsStillSimulating()

```
bool Microstructure::IsStillSimulating ( ) const  [inline]
```

Checks if the simulation is still running.

Returns

True if the simulation is still running, otherwise false.

3.2.3.6 RemoveInterfaceIndex()

```
void Microstructure::RemoveInterfaceIndex (
            int index )
```

Removes an index from the set of interface indices.

Updates set of (sub)grain boundary cells by removing the cell of index, and updates cell's (index) identity regarding whether it is interface.

Parameters

index	Index to remove.
index	the cell's index that will no longer be recognized as (sub)grain boundary cell

3.2.3.7 RemoveInterphaseIndex()

```
void Microstructure::RemoveInterphaseIndex (
            int index )
```

Removes an index from the set of interphase indices.

Updates set of phase boundary cells by removing the cell of index, and updates cell's (index) identity regarding whether it is interface.

Parameters

index	Index to remove.
index	the cell's index that will now no longer recognized as phase cell

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

- src/microstructure.h
- src/microstructure.cpp

3.3 Orientations Class Reference

Class to handle orientations and symmetry operations.

```
#include <orientation.h>
```

Public Member Functions

- Orientations (void)

  Default constructor for the Orientations class.
- void SetOrientationParameters (const UserSettings &us)

  Destructor (no need).
- Eigen::Vector3f GetEulerAngles (const int id) const

  Retrieves the Euler angles for a given orientation ID.
- void DebugPrintOrientations () const

  Prints all orientations for debugging purposes.

Euler Grid and Problem Size

- float Lim (const int i) const

Returns the limit of Euler space for a given component.

- int addOriAndReturnId (const Eigen::Vector3f &ea)
  Adds an orientation and returns its ID.
- float misorientationQ (const Eigen::Quaternionf q0, const Eigen::Quaternionf q1) const
  Calculates the misorientation between two quaternions.
- float DistanceOfBoundaryFromCSL19a (const Eigen::Vector3f &ea1, const Eigen::Vector3f &ea2) const
  Calculates the distance of the boundary from CSL19a.
- float CalculateMisorientationBetweenTwoOriIds (const int Id1, const int Id2) const
  Calculates the misorientation between two orientations given their IDs.
- float CalcMisorientationFromEulerAngles (const Eigen::Vector3f &ea1, const Eigen::Vector3f &ea2) const
  Calculates the misorientation between two orientations given their Euler angles.

3.3.1 Detailed Description

Class to handle orientations and symmetry operations.

This class manages the orientations and crystal symmetries of materials in the OMicroN simulation. It provides functionalities to add and retrieve orientations, convert between Euler angles and quaternions, and calculate misorientations and other orientation-related metrics.

Orientations are stored (provided/exported) as Euler angles, but all operations are performed using quaternions.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Orientations()

```
Orientations::Orientations (
            void  )
```

Default constructor for the Orientations class.

Initializes an instance of the Orientations class. This constructor does not perform any specific initialization tasks beyond setting up the class instance.

This constructor initializes an instance of the Orientations class. As it is a default constructor, it does not perform any specific actions or initialize any member variables.

3.3.3 Member Function Documentation

3.3.3.1 addOriAndReturnId()

```
int Orientations::addOriAndReturnId (
            const Eigen::Vector3f & ea )
```

Adds an orientation and returns its ID.

Returns

ID of the added orientation.

Parameters

ea	Euler angles.

Returns

ID of the added orientation.

3.3.3.2 CalcMisorientationFromEulerAngles()

```
float Orientations::CalcMisorientationFromEulerAngles (
            const Eigen::Vector3f & ea1,
            const Eigen::Vector3f & ea2 ) const
```

Calculates the misorientation between two orientations given their Euler angles.

Calculates the misorientation angle between two orientations given their Euler angles.

Parameters

ea1	Euler angles of the first orientation.
ea2	Euler angles of the second orientation.

Returns

Misorientation angle.

Parameters

ea1	Euler angles of the first orientation.
ea2	Euler angles of the second orientation.

Returns

Misorientation angle in degrees.

3.3.3.3 CalculateMisorientationBetweenTwoOriIds()

```
float Orientations::CalculateMisorientationBetweenTwoOriIds (
            const int Id1,
            const int Id2 ) const
```

Calculates the misorientation between two orientations given their IDs.

Calculates the misorientation angle between two orientations given their IDs.

Parameters

Id1	ID of the first orientation.
Id2	ID of the second orientation.

Returns

Misorientation angle.

Parameters

Id1	ID of the first orientation.
Id2	ID of the second orientation.

Returns

Misorientation angle in degrees.

3.3.3.4 DistanceOfBoundaryFromCSL19a()

```
float Orientations::DistanceOfBoundaryFromCSL19a (
            const Eigen::Vector3f & ea1,
            const Eigen::Vector3f & ea2 ) const
```

Calculates the distance of the boundary from CSL19a.

Calculates the distance of the boundary from the CSL19a for given Euler angles.

Parameters

ea1	Euler angles of the first phase.
ea2	Euler angles of the second phase.

Returns

Misorientation angle.

Parameters

ea1	Euler angles of the first orientation.
ea2	Euler angles of the second orientation.

Returns

Misorientation angle in degrees.

3.3.3.5 GetEulerAngles()

```
Eigen::Vector3f Orientations::GetEulerAngles (
            const int id ) const  [inline]
```

Retrieves the Euler angles for a given orientation ID.

Parameters

id	Orientation ID.

Returns

Euler angles as an Eigen::Vector3f.

3.3.3.6 Lim()

```
float Orientations::Lim (
            const int i ) const  [inline]
```

Returns the limit of Euler space for a given component.

Parameters

i	Component index (0:phi1, 1:Phi, 2:phi2).

Returns

Limit of Euler space for the component.

3.3.3.7 misorientationQ()

```
float Orientations::misorientationQ (
            const Eigen::Quaternionf q0,
            const Eigen::Quaternionf q1 ) const
```

Calculates the misorientation between two quaternions.

Calculates the misorientation angle between two quaternions.

Parameters

q0	First quaternion.
q1	Second quaternion.

Returns

Misorientation angle.

Parameters

q0	First quaternion.
q1	Second quaternion.

Returns

Misorientation angle in degrees.

3.3.3.8 SetOrientationParameters()

```
void Orientations::SetOrientationParameters (
            const UserSettings & us )  [inline]
```

Destructor (no need).

Sets the orientation parameters from user settings.

Parameters

us	UserSettings object containing the parameters.

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

- src/orientation.h
- src/orientation.cpp

3.4 StateVars Class Reference

Public Member Functions

- void SetEBSDRelatedStuffForRexAndGG (int index, float CI, int RX, float rho)

  Sets id of grain to which the cell belongs.
- void **SetStateVariablesRelatedToRexAndGG** (int index, float ReRate, int ReFraction, float MaxAngle↩ Passed)
- void **SetCellAsInterface** (int index)
- void **SetCellAsInterphase** (int index)
- void **SetCellAsNonInterface** (int index)
- void **SetCellAsNonInterphase** (int index)
- int **IsCellInterface** (int index)
- int **IsCellInterphase** (int index)
- void **SetOrIdOfCell** (int index, int OrId)

- void **SetIfCellHasNonIndNeighbours** (int index, int HasNeighNonInd)
- void **SetLatticeIdOfCell** (int index, int latticeValue)

Constructors and destructor

- StateVars (void)
  Default constructor.
- StateVars (int nx, int ny, int nz, float dx, int grid, bool IncludeSoluteDiffusion, bool IncludeRexAndGG, bool AllowIntMigration, bool initXCEqInt, bool initXCEqDef)
  Constructs a new instance of StateVars, initialing the grid and the basic state variables, as well as the state variables related to the simulation (i.e. froom user defined settings)
- ~StateVars (void)
  Destructor.

Function for deleting KAM - it is called after initialing all pointers with simulation-necessary state variables (KAM is not useful after setting dislocation density (if not read)).

- void **eraseKAM** ()

Functions for retrieving state variables of cells as they are stored in CA (orientation id, lattice id)

- int **GetLatticeIdOfCell** (int index)
- int **GetOrIdOfCell** (int index)

Functions for retrieving lattice name based on lattice Id stored in CA

- bool **IsCellFCC** (int index)
- bool **IsCellBCC** (int index)
- bool **IsCellHCP** (int index)
- bool **IsLatticeIdFCC** (int latticeId)
- bool **IsLatticeIdBCC** (int latticeId)
- bool **IsLatticeIdHCP** (int latticeId)

Functions for retrieving geometry data

- std::vector< int > GetAllNeighbourCells (int index) const
  Returns a vector with indices of all neighbour cells, depending on the grid and settings used in the simulation.
- void GetNeighbourCellOffsets (int index, int *all_p) const
  Calculates in place the pointer to the array with indices of all neighbour cells, depending on the grid and settings used in the simulation.
- int **GetGridType** (void) const
- int IJKToIndex (int i, int j, int k) const
  Converts X, Y, Z grid positions (i, j, k) to cell index.
- int GetIndexFromIJK (int i, int j, int k) const
  Alias of IJK2Index.
- void IndexToIJK (int index, int *i, int *j, int *k) const
  Converts cell index to X, Y, Z grid positions (i, j, k)
- void **IndexToIJK** (int index, float *i, float *j, float *k) const
- std::array< int, 3 > GetIJKFromIndex (int index) const
  Similar to Index2IJK, but returns i, j, k as std:array.
- float GetXFromIndex (int index) const
  Gets the X coordinate (in m) from the cell index.
- float GetYFromIndex (int index) const
  Gets the Y coordinate (in m) from the cell index.
- float GetZFromIndex (int index) const
  Gets the Z coordinate (in m) from the cell index.
- std::array< float, 3 > GetXYZFromIndex (float index) const

Gets the XYZ coordinates (in m) from the cell index.

- float GetDistanceBetweenCells (int indexA, int indexB) const

  Returns the distance between two cells (in m)
- double GetXCOfCell (int index) const

  Returns (optionally used) local carbon concentration.
- double **GetKappaFactorForCTrappedInCell** (int index) const
- double **GetXCTrappedInCell** (int index) const
- void **ConvertTotCellConcentrationsInCarbonPerIron** () const
- void **ConvertTotCellConcentrationsInAtFraction** () const
- unsigned char GetBoundaryCell (int index) const

  Returns the value of mvpBoundaryCell. This value is associated to the position of the cell on the grid. If it belongs to the boundary, it assumes a value > 0.
- bool IsCellOnBoundary (int index) const

  Returns flag that determines if a cell is on a system (periodic) boundary.

Diffusion calculation

- void SoluteDiffusionStep (ThermChemKin *th_p, double dt, bool AllowSoluteSegregation, double max↩ DiffusivityInTimeStep, bool IsPartitioningHappeningHere, bool IsSoluteSegregationHappeningHere)

  Solves one diffusion step.
- void **PutBackCarbonTrappedAndCalculateNewCarbonFreeCarbonTrapped** ()
- void **SetCarbonTrappedAndCarbonFreeForGivenXcTot** ()

Friends

- class **Microstructure**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 StateVars()

```
StateVars::StateVars (
            int nx,
            int ny,
            int nz,
            float dx,
            int grid,
            bool IncludeSoluteDiffusion,
            bool IncludeRexAndGG,
            bool AllowIntMigration,
            bool initXCEqInt,
            bool initXCEqDef )
```

Constructs a new instance of StateVars, initialing the grid and the basic state variables, as well as the state variables related to the simulation (i.e. froom user defined settings)

Parameters

nx	Number of cells along X direction
ny	Number of cells along Y direction
nz	Number of cells along Z direction

Parameters

dx	Grid spacing in m
grid	Type of grid (e.g. 0 for square/cubic and 1 for hexagonal)
IncludeSoluteDiffusion	Flag signalling if solute diffusion will be simulated
IncludeRexAndGG	Flag signalling if recrystallization and/or grain growth will be simulated
AllowIntMigration	Flag signalling if phase transformations (interface migration between dissimilar phases) simulated
initXCEqInt	Flag signaling whether the carbon equilibrium at local interphase partitioning takes place before or during the diffusion step. If before (ALTHOUGH THIS IS NOT RECOMMENDED) then initXCEqInt = true and should be initialized here
initXCEqDef	Flag signaling whether the carbon equilibrium between free lattice and local defects takes place before or during the diffusion step. If before (ALTHOUGH THIS IS NOT RECOMMENDED) then initXCEqInt = true and should be initialized here

3.4.1.2 ~StateVars()

```
StateVars::~StateVars (
            void  )
```

Destructor.

Takes care of freeing memory assigned to pointers

3.4.2 Member Function Documentation

3.4.2.1 GetAllNeighbourCells()

```
std::vector< int > StateVars::GetAllNeighbourCells (
            int index ) const
```

Returns a vector with indices of all neighbour cells, depending on the grid and settings used in the simulation.

Parameters

index	Index of cell for which to return the neighbour cells

Returns

A vector with all neighbours

3.4.2.2 GetBoundaryCell()

```
unsigned char StateVars::GetBoundaryCell (
            int index ) const  [inline]
```

Returns the value of mvpBoundaryCell. This value is associated to the position of the cell on the grid. If it belongs to the boundary, it assumes a value > 0.

Parameters

index	Cell index

Returns

The value of mvpBoundaryCell, which is > 0 if the cell belongs to the periodic boundary. 0 otherwise.

3.4.2.3 GetDistanceBetweenCells()

```
float StateVars::GetDistanceBetweenCells (
            int indexA,
            int indexB ) const
```

Returns the distance between two cells (in m)

Gets the distance between two cells in m (taking care of periodic boundaries)

Parameters

indexA	index of first cell
indexB	index of second cell

Returns

distance between the two cells in m

3.4.2.4 GetIJKFromIndex()

```
std::array< int, 3 > StateVars::GetIJKFromIndex (
            int index ) const
```

Similar to Index2IJK, but returns i, j, k as std:array.

Parameters

index	Cell index

Returns

X, Y, Z coordinates as grid positions as std::array

3.4.2.5 GetNeighbourCellOffsets()

```
void StateVars::GetNeighbourCellOffsets (
            int index,
            int * all_p ) const
```

Calculates in place the pointer to the array with indices of all neighbour cells, depending on the grid and settings used in the simulation.

Parameters

index	Index of cell for which to return the neighbour cells
∗all_p	Pointer to array of offsets for all neighbours

3.4.2.6 GetXCOfCell()

```
double StateVars::GetXCOfCell (
            int index ) const  [inline]
```

Returns (optionally used) local carbon concentration.

Parameters

index	Cell index

Returns

Local carbon concentration of cell in at. fraction is available, otherwise, returns -1

3.4.2.7 GetXFromIndex()

```
float StateVars::GetXFromIndex (
            int index ) const
```

Gets the X coordinate (in m) from the cell index.

Parameters

index	Cell index

Returns

The X coordinate in m

3.4.2.8 GetXYZFromIndex()

```
std::array< float, 3 > StateVars::GetXYZFromIndex (
            float index ) const
```

Gets the XYZ coordinates (in m) from the cell index.

Parameters

index	Cell index

Returns

The XYZ coordinates in m as an std::array

3.4.2.9 GetYFromIndex()

```
float StateVars::GetYFromIndex (
            int index ) const
```

Gets the Y coordinate (in m) from the cell index.

Parameters

index	Cell index

Returns

The Y coordinate in m

3.4.2.10 GetZFromIndex()

```
float StateVars::GetZFromIndex (
            int index ) const
```

Gets the Z coordinate (in m) from the cell index.

Parameters

index	Cell index

Returns

The Z coordinate in m

3.4.2.11 IJKToIndex()

```
int StateVars::IJKToIndex (
            int i,
            int j,
            int k ) const
```

Converts X, Y, Z grid positions (i, j, k) to cell index.

Parameters

i	X coordinate as grid position (dimensionless)
j	Y coordinate as grid position (dimensionless)
k	Z coordinate as grid position (dimensionless)

Returns

Cell index

3.4.2.12 IndexToIJK()

```
void StateVars::IndexToIJK (
            int index,
            int * i,
            int * j,
            int * k ) const
```

Converts cell index to X, Y, Z grid positions (i, j, k)

Parameters

index	Cell index
$*i$	X coordinate as grid position returned by pointer value
$*j$	Y coordinate as grid position returned by pointer value
$*k$	Z coordinate as grid position returned by pointer value

3.4.2.13 IsCellOnBoundary()

```
bool StateVars::IsCellOnBoundary (
            int index ) const  [inline]
```

Returns flag that determines if a cell is on a system (periodic) boundary.

Parameters

index	Cell index

Returns

A boolean: true if cell belongs to periodic boundary, false otherwise.

3.4.2.14 SetEBSDRelatedStuffForRexAndGG()

```
void StateVars::SetEBSDRelatedStuffForRexAndGG (
            int index,
            float CI,
            int RX,
            float rho )
```

Sets id of grain to which the cell belongs.

Initialises measurement-related quantities when microstructure comes from EBSD ∗.

Parameters

index	Cell index
grain↩ Id	Grain id
index	the index of cell for which the state variables will be set
CI	the confidence index of the measured pixel
RX	state variable regarding the state of the pixel (normally non-recrystallized for deformed inputs) - unless we read partially recrystallized state or state after previous simulation
rho	the dislocation density measured / calculated / simulated from other software

3.4.2.15 SoluteDiffusionStep()

```
void StateVars::SoluteDiffusionStep (
            ThermChemKin * TCK_p,
```

```
double dt,
bool AllowSoluteSegregation,
double maxDiffusivityInTimeStep,
bool IsPartitioningHappeningHere,
bool IsSoluteSegregationHappeningHere )
```

Solves one diffusion step.

Parameters

TCK_p	pointer to ThermChemKin instance
dt	predetermined time step
AllowSoluteSegregation	bool on whether we have solute trapping in general during simulation
maxDiffusivityInTimeStep	the maximum value of diffusivity between neighbour cells calculated in microstructure class
IsPartitioningHappeningHere	bool on whether the diffusion equation includes the solute trapping equilibrium (i.e. if trapping is part of the numerical system Ax=b)
IsSoluteSegregationHappeningHere	bool on whether the diffusion equation includes the solute trapping equilibrium (i.e. if trapping is part of the numerical system Ax=b)

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

- src/stateVars.h
- src/stateVars.cpp

3.5 ThermChemKin Class Reference

Manages all the physical quantities that affect thermodynamics and kinetics on the microstructure simulation. Stores all thermodynamic, chemical, kinetic properties in relation to lattice, temperature, solute concentration, defect density, etc.. These affect all simulated processes recrystallization, grain growth), solute redistribution (partitioning, diffusion, trapping to defects), phase transformations.

```
#include <ThermoChemKinetics.h>
```

Public Member Functions

- ThermChemKin (double userXCo, double userStartTemperature, double userGB_Mo, double userGB_Qg, double userGB_E, double userPB_Mo, double userPB_Qg, double userPB_E, double user_DiffPreFactor, double user_DiffQg)

  Constructor for ThermChemKin class.
- ~ThermChemKin ()

  Destructor.
- void SetTemperatureForThisStep (double temp)

  Sets the temperature for the current simulation step.
- double GetTemperatureForThisStep () const

  Gets the temperature for the current simulation step.
- double GetGrainBoundaryMobility () const

  Calculates the grain boundary mobility.
- double GetPhaseBoundaryMobility () const

  Calculates the phase boundary mobility.

- double GetGrainBoundaryEnergy () const

  Gets the grain boundary energy density.
- double GetPhaseBoundaryEnergy () const

  Gets the phase boundary energy density.

Helper Functions

- double CarbonWtPercentToAtFraction (const double XcInWt)

  Converts carbon weight percent to atomic fraction.
- double CarbonAtFractionToWtPercent (const double XcInAtFraction)

  Converts carbon atomic fraction to weight percent.
- void CalculateXcEqNextToInterphase (const double T, double InitialXcAlphaInAt, double InitialXcGamma↩ InAt, double ∗XcAlphaEq, double ∗XcGammaEq)

  Calculates the equilibrium carbon concentration next to the interphase.
- double GetCDiffusivityAgren (const double xC) const

  Gets the carbon diffusivity based on temperature using the Agren model.
- double GetCDiffusivityMartensite () const

  Gets the carbon diffusivity for martensite.
- double GetCDiffusivityAgrenWithRestrictionForXC (const double xC, const double MaxAllowedXcTo↩ AffectDiffusivity) const

  Gets the carbon diffusivity with restriction for maximum allowed carbon concentration.
- double GetSoluteDiffusivityFromUser () const

  Gets the solute diffusivity from user-defined parameters.

Chemical Potentials and Thermodynamic Equilibrium

- void LoadChemicalPotentialsAndLocalEquilibriumXC (const char ∗Thermofilename, bool KeepConstant↩ IronAtomsPerCell)

  Loads chemical potentials and local equilibrium data from a file.
- void MakeTablesYForXCEqVSCarbonInterface (const double ∗XCLocalEqFCC_AsRead, const double ∗XCLocalEqBCC_AsRead)

  Creates tables for Y vs. equilibrium carbon concentration at the interface.
- void MakeTablesYForGibbsVSCarbon (const double ∗GibbsFCC_AsRead, const double ∗GibbsBCC_As↩ Read, const double ∗MuSubstitutionalFCC_AsRead, const double ∗MuSubstitutionalBCC_AsRead)

  Creates tables for Gibbs energy vs. carbon concentration.
- void GetEqMuCarbonRelatedParametersABCD (double ∗A, double ∗B, double ∗C, double ∗D)

  Gets the parameters A, B, C, D for equilibrium carbon concentration.
- double GetMuSubstitutionalInBCC (const double xC) const

  Gets the substitutional chemical potential in BCC.
- double GetMuSubstitutionalInFCC (const double xC) const

  Gets the substitutional chemical potential in FCC.
- double GetEqXcFCCAtThisInterface (const double xCBCCNow, const double xCFCCNow) const

  Gets the equilibrium carbon concentration in FCC at the interface.
- double GetValueLocalEqXC_Clamped (const double X, const double ∗Y) const

  Gets the local equilibrium carbon concentration, clamped to table limits.
- double GetValueGibbs_Clamped (const double X, const double ∗Y) const

  Gets the Gibbs energy, clamped to table limits.
- double FerriteMolarVolume () const

  Returns the molar volume for iron BCC.
- double AusteniteMolarVolume () const

  Returns the molar volume for iron FCC.

3.5.1 Detailed Description

Manages all the physical quantities that affect thermodynamics and kinetics on the microstructure simulation. Stores all thermodynamic, chemical, kinetic properties in relation to lattice, temperature, solute concentration, defect density, etc.. These affect all simulated processes recrystallization, grain growth), solute redistribution (partitioning, diffusion, trapping to defects), phase transformations.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 ThermChemKin()

```
ThermChemKin::ThermChemKin (
            double userXCo,
            double userStartTemperature,
            double userGB_Mo,
            double userGB_Qg,
            double userGB_E,
            double userPB_Mo,
            double userPB_Qg,
            double userPB_E,
            double user_DiffPreFactor,
            double user_DiffQg )
```

Constructor for ThermChemKin class.

Initializes the ThermChemKin object with user-defined parameters.

Parameters

userXCo	Initial carbon concentration (atomic fraction).
userStartTemperature	Initial temperature (K).
userGB_Mo	Pre-exponential factor for grain boundary migration.
userGB_Qg	Activation energy for grain boundary migration (J/mol).
userGB_E	Grain boundary energy density (J/m^2).
userPB_Mo	Pre-exponential factor for phase boundary migration.
userPB_Qg	Activation energy for phase boundary migration (J/mol).
userPB_E	Phase boundary energy density (J/m^2).
user_DiffPreFactor	Pre-exponential factor for solute diffusion.
user_DiffQg	Activation energy for solute diffusion (J/mol).

3.5.3 Member Function Documentation

3.5.3.1 AusteniteMolarVolume()

```
double ThermChemKin::AusteniteMolarVolume ( ) const  [inline]
```

Returns the molar volume for iron FCC.

Returns

Molar volume for iron FCC (m^3/mol).

3.5.3.2 CalculateXcEqNextToInterphase()

```
void ThermChemKin::CalculateXcEqNextToInterphase (
            const double T,
            double InitialXcAlphaInAt,
            double InitialXcGammaInAt,
            double * XcAlphaEq,
            double * XcGammaEq )
```

Calculates the equilibrium carbon concentration next to the interphase.

The purpose of this function is to partition the solute ONLY IN CASE the chemical potentials are known /∗∗ Calculates local equilibrium (partitioning) carbon concentrations based on analytical function for iron-carbon that is solved numerically.

Parameters

T	Temperature (K).
InitialXcAlphaInAt	Initial carbon concentration in alpha phase (atomic fraction).
InitialXcGamma↩ InAt	Initial carbon concentration in gamma phase (atomic fraction).
XcAlphaEq	Pointer to store equilibrium carbon concentration in alpha phase.
XcGammaEq	Pointer to store equilibrium carbon concentration in gamma phase.

Parameters

T	the current temperature
InitialXcAlphaInAt	the current carbon concentration in BCC next to interphase
InitialXcGamma↩ InAt	the current carbon concentration in FCC next to interphase

Returns

- XcAlphaEq the partitioned (and local equilibrium) carbon concentration in BCC next to interphase

Returns

- XcGammaEq the partitioned (and local equilibrium) carbon concentration in FCC next to interphase
- TODO: read the lattice data

3.5.3.3 CarbonAtFractionToWtPercent()

```
double ThermChemKin::CarbonAtFractionToWtPercent (
            const double XcInAtFraction )
```

Converts carbon atomic fraction to weight percent.

Parameters

XcInAtFraction	Carbon atomic fraction.

Returns

Carbon weight percent.

Returns the converted at frac to wt pct for carbon - iron

Parameters

XcInAtFraction	concentration of carbon in at fraction • ∗ TODO: read the lattice data

3.5.3.4 CarbonWtPercentToAtFraction()

```
double ThermChemKin::CarbonWtPercentToAtFraction (
            const double XcInWt )
```

Converts carbon weight percent to atomic fraction.

Parameters

XcInWt	Carbon weight percent.

Returns

Carbon atomic fraction.

Returns the converted wt pct to at frac for carbon - iron

Parameters

XcInWt	concentration of carbon in wt pct

-
- TODO: read the lattice data

3.5.3.5 FerriteMolarVolume()

```
double ThermChemKin::FerriteMolarVolume ( ) const  [inline]
```

Returns the molar volume for iron BCC.

Returns

Molar volume for iron BCC (m^3/mol).

3.5.3.6 GetCDiffusivityAgren()

```
double ThermChemKin::GetCDiffusivityAgren (
            const double xC ) const
```

Gets the carbon diffusivity based on temperature using the Agren model.

Gets Ågren's composition dependent austenite carbon diffusivity at temperature #mVT (J. Ågren, Scr. Metall. 20 (1986) 1507–1510)

Parameters

xC	Carbon atomic fraction.

Returns

Carbon diffusivity.

Parameters

Carbon	concentration in at. fraction

Returns

Diffusivity in $m^2 s^{-1}$

3.5.3.7 GetCDiffusivityAgrenWithRestrictionForXC()

```
double ThermChemKin::GetCDiffusivityAgrenWithRestrictionForXC (
            const double xC,
            const double MaxAllowedXcToAffectDiffusivity ) const
```

Gets the carbon diffusivity with restriction for maximum allowed carbon concentration.

This function is useful if there are tiny FCC grains that will enrich so much that diffusivity becomes very high and time steps reqired for diffusion are then very small. Gets carbon diffusivity using Agren's expression (for temperature mvT) but with a max allowed value.

Parameters

xC	Carbon atomic fraction.
MaxAllowedXcToAffectDiffusivity	Maximum allowed carbon atomic fraction to affect diffusivity.

Returns

Carbon diffusivity.

Parameters

xC	the local carbon concentration
MaxAllowedXcToAffectDiffusivity	the max allowed value

Returns

Diffusivity in $m^2 s^{-1}$

3.5.3.8 GetCDiffusivityMartensite()

```
double ThermChemKin::GetCDiffusivityMartensite ( ) const
```

Gets the carbon diffusivity for martensite.

Gets Ågren's martensite austenite carbon diffusivity at temperature #mVT (J. Ågren, Journal of Physics and Chemistry of Solids (1982)) $*$.

Returns

Carbon diffusivity.

Diffusivity in $m^2 s^{-1}$

3.5.3.9 GetEqMuCarbonRelatedParametersABCD()

```
void ThermChemKin::GetEqMuCarbonRelatedParametersABCD (
            double * A,
            double * B,
            double * C,
            double * D )
```

Gets the parameters A, B, C, D for equilibrium carbon concentration.

Parameters

A	Pointer to store parameter A.
B	Pointer to store parameter B.
C	Pointer to store parameter C.
D	Pointer to store parameter D.

3.5.3.10 GetEqXcFCCAtThisInterface()

```
double ThermChemKin::GetEqXcFCCAtThisInterface (
            const double xCBCCNow,
            const double xCFCCNow ) const
```

Gets the equilibrium carbon concentration in FCC at the interface.

Returns the linearly interpolated value equilibrium (local partitioning) carbon concentration in FCC given the total interphase carbon concentration.

Parameters

xCBCCNow	Carbon concentration in BCC (atomic fraction).
xCFCCNow	Carbon concentration in FCC (atomic fraction).

Returns

Equilibrium carbon concentration in FCC (atomic fraction).

Parameters

xCBCCNow	the current (before further partitioning) carbon concentration in adjacent BCC
xCFCCNow	the current (before further partitioning) carbon concentration in adjacent FCC

Returns

xCFCC linearly interpolated value equilibrium (local partitioning) carbon concentration in FCC

3.5.3.11 GetGrainBoundaryEnergy()

```
double ThermChemKin::GetGrainBoundaryEnergy ( ) const  [inline]
```

Gets the grain boundary energy density.

Returns

Grain boundary energy density (J/m^2).

3.5.3.12 GetGrainBoundaryMobility()

```
double ThermChemKin::GetGrainBoundaryMobility ( ) const  [inline]
```

Calculates the grain boundary mobility.

Returns

Grain boundary mobility.

3.5.3.13 GetMuSubstitutionalInBCC()

```
double ThermChemKin::GetMuSubstitutionalInBCC (
            const double xC ) const
```

Gets the substitutional chemical potential in BCC.

Returns the linearly interpolated value of chemical potential of substitutional atoms in BCC given the carbon concentration.

Parameters

xC	Carbon atomic fraction.

Returns

Substitutional chemical potential (J/mol).

Parameters

xC	the carbon concentration

3.5.3.14 GetMuSubstitutionalInFCC()

```
double ThermChemKin::GetMuSubstitutionalInFCC (
            const double xC ) const
```

Gets the substitutional chemical potential in FCC.

Returns the linearly interpolated value of chemical potential of substitutional atoms in FCC given the carbon concentration.

Parameters

xC	Carbon atomic fraction.

Returns

Substitutional chemical potential (J/mol).

Parameters

xC	the carbon concentration

3.5.3.15 GetPhaseBoundaryEnergy()

```
double ThermChemKin::GetPhaseBoundaryEnergy ( ) const  [inline]
```

Gets the phase boundary energy density.

Returns

Phase boundary energy density (J/m^2).

3.5.3.16 GetPhaseBoundaryMobility()

```
double ThermChemKin::GetPhaseBoundaryMobility ( ) const  [inline]
```

Calculates the phase boundary mobility.

Returns

Phase boundary mobility.

3.5.3.17 GetSoluteDiffusivityFromUser()

```
double ThermChemKin::GetSoluteDiffusivityFromUser ( ) const
```

Gets the solute diffusivity from user-defined parameters.

Gets solute diffusivity using Arrhenius type equation at temperature #mT given user defined input values.

Returns

Solute diffusivity.
Diffusivity in $m^2 s^{-1}$

3.5.3.18 GetTemperatureForThisStep()

```
double ThermChemKin::GetTemperatureForThisStep ( ) const  [inline]
```

Gets the temperature for the current simulation step.

Returns

Temperature (K).

3.5.3.19 GetValueGibbs_Clamped()

```
double ThermChemKin::GetValueGibbs_Clamped (
            const double X,
            const double * Y ) const
```

Gets the Gibbs energy, clamped to table limits.

Parameters

X	Carbon concentration.
Y	Pointer to the data table.

Returns

Gibbs energy.

3.5.3.20 GetValueLocalEqXC_Clamped()

```
double ThermChemKin::GetValueLocalEqXC_Clamped (
            const double X,
            const double * Y ) const
```

Gets the local equilibrium carbon concentration, clamped to table limits.

Parameters

X	Carbon concentration.
Y	Pointer to the data table.

Returns

Local equilibrium carbon concentration.

3.5.3.21 LoadChemicalPotentialsAndLocalEquilibriumXC()

```
void ThermChemKin::LoadChemicalPotentialsAndLocalEquilibriumXC (
            const char * filename,
            bool KeepConstantIronAtomsPerCell )
```

Loads chemical potentials and local equilibrium data from a file.

Loads the hdf5 file that contains chemical potentials of substitutional lattice, the equilibrium (paritioning) solute concentration for various interphase compositions.

Parameters

Thermofilename	Filename containing the thermodynamic data.
KeepConstantIronAtomsPerCell	Boolean flag to keep constant iron atoms per cell.
filename	the name of the file with the data
KeepConstantIronAtomsPerCell	user defined flag signaling whether equilibrium requires constant at. fractions or substitional-to-total fraction

3.5.3.22 MakeTablesYForGibbsVSCarbon()

```
void ThermChemKin::MakeTablesYForGibbsVSCarbon (
            const double * GibbsFCC_AsRead,
            const double * GibbsBCC_AsRead,
            const double * MuSubstitutionalFCC_AsRead,
            const double * MuSubstitutionalBCC_AsRead )
```

Creates tables for Gibbs energy vs. carbon concentration.

Makes tables for carbon local equilibrium due to partitioning.

Parameters

GibbsFCC_AsRead	Pointer to Gibbs energy data for FCC.
GibbsBCC_AsRead	Pointer to Gibbs energy data for BCC.
MuSubstitutionalFCC_AsRead	Pointer to chemical potential data for FCC.
MuSubstitutionalBCC_AsRead	Pointer to chemical potential data for BCC.
GibbsFCC_AsRead	the Gibbs energy in FCC (J per mole) given (AND EQUIDISTANT) total carbon at interphase
GibbsBCC_AsRead	the Gibbs energy in BCC (J per mole) given (AND EQUIDISTANT) total carbon at interphase
MuSubstitutionalFCC_AsRead	the chemical potential of substitutional atoms in FCC (J per mole) given the (EQUIDISTANT) carbon concentration
MuSubstitutionalBCC_AsRead	the chemical potential of substitutional atoms in BCC (J per mole) given (EQUIDISTANT) carbon concentration

3.5.3.23 MakeTablesYForXCEqVSCarbonInterface()

```
void ThermChemKin::MakeTablesYForXCEqVSCarbonInterface (
            const double * XCLocalEqFCC_AsRead,
            const double * XCLocalEqBCC_AsRead )
```

Creates tables for Y vs. equilibrium carbon concentration at the interface.

Makes tables for carbon local equilibrium due to partitioning.

Parameters

XCLocalEqFCC_AsRead	Pointer to data for FCC.
XCLocalEqBCC_AsRead	Pointer to data for BCC.
XCLocalEqFCC_AsRead	the FCC eq. concentration for given (AND EQUIDISTANT) total carbon at interphase
XCLocalEqBCC_AsRead	the BCC eq. concentration for given (AND EQUIDISTANT) total carbon at interphase

3.5.3.24 SetTemperatureForThisStep()

```
void ThermChemKin::SetTemperatureForThisStep (
            double temp )  [inline]
```

Sets the temperature for the current simulation step.

Parameters

temp	Temperature (K).

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

- src/ThermoChemKinetics.h
- src/ThermoChemKinetics.cpp

3.6 UserSettings Class Reference

Handles all the simulation parameters and sets them according to the input file and/or default values.

`#include <settings.h>`

Public Member Functions

- UserSettings (const std::string &inputFile)

  Constructor that initializes settings from the input file.
- void setDefaultValues ()

  Sets default values for all parameters. If a parameter is found in the input file, its value will be used instead.

- void writeToFile (const std::string &outputFile)

  Writes the settings to the output file, including default values.
- void printParameters ()

  Prints the parameters as read from the input file.
- std::string getStartFileName () const

  Returns the path to the input (starting) microstructure file.
- std::string getOutputFolderPath () const

  Returns the path to the folder where simulation outputs are stored.
- std::string getThermodynamicDataFile () const

  Returns the path to the file containing thermodynamic data.

Public Attributes

- double mvTimeStep

  Desired time step for solute redistribution. Note: When recrystallization and grain growth are simulated, the time step will adapt to the maximum reorientation rate. Note: When simulating solute redistribution the user must set it according to the highest expected diffusion rate, which in the numerical system is dt ∗ maxDiffusivityInTimeStep / (mvDx ∗ mvDx). So dt must be chosen such as dt ∗ maxDiffusivityInTimeStep / (mvDx ∗ mvDx) is lower than 1.
- double mvStartTemperature

  Temperature at the beginning of the applied treatment (in K).
- double mvEndTemperature

  Temperature at the end of the applied treatment (in K).
- double mvTimeTotal

  Total simulation time (s).
- int mvNx

  Number of elements in the x direction.
- int mvNy

  Number of elements in the y direction.
- int mvNz

  Number of elements in the z direction.
- double mvDx

  Grid spacing (m).
- int mvGridType

  Integer determining the type of grid used. 0 is regular (square/cubic) and 1 is hexagonal.
- int mvHasSoluteDiffusion

  Flag indicating whether solute redistribution (e.g., diffusion, partitioning, trapping) should be simulated.
- int mvIsRexAndGG

  Flag indicating whether interface migration in the same phase (e.g., recrystallization and grain growth) should be simulated.
- float mvLowerMisorientationCutOff

  Misorientation below which any misorientation between pixels is considered noise and no orientation gradient is considered (e.g., 0.4 degrees).
- float mvHAGB

  Misorientation above which a high-angle grain boundary is considered. Energy density and mobility are constant above this value (e.g., 15 degrees).
- std::string mvStartFileName

  Path to the input (starting) microstructure file.
- std::string mvOutputFolderPath

  Path to the folder where simulation output files are written.
- std::string mvThermodynamicsDataFilename

  Path to the HDF5 file containing thermodynamic inputs for solute partitioning and/or phase transformations.

- int mvIsTestRVE

  Flag indicating whether the simulation involves microstructure evolution (considering imported state variables) or code testing.
- int mvTestVonNeumannInSquare

  Flag indicating whether the code is tested for grain growth. Turn on means modeling grain growth in an unacceptable grid, useful for testing.
- int mvAllowPeriodicBoundConditions

  Flag indicating whether boundary conditions are enabled. Turning off is useful for test simulations of specific cases or recrystallization in very small RVEs.
- float mvMinimumCI

  Minimum confidence index for valid crystal orientations imported (useful when importing uncleaned EBSD microstructures).
- float mvRexGGParameterC

  Parameter C that calibrates the driving force for grain growth and recrystallization. Accepted values are 0.5, 0.7, 1.0.
- int mvReadDislocationDensity

  Flag indicating whether the dislocation density provided by the input file should be used in the simulation. Turn off means calculating local dislocation density based on orientation gradients.
- int mvIncludeCSL19FastGrowth

  Flag indicating whether CSL19 boundaries (Ibe and Lucke) grow faster.
- int mvIsOnlyAnOrientationSubsetConsidered

  Flag indicating whether only a subset of orientations is considered in the microstructure evolution.
- int mvConsiderConstantIronAtomsPerCell

  Flag indicating whether mass conservation for solutes refers to C/Fe or C/(Fe+C). C/Fe is more accurate, while C/(Fe+C) is commonly used.
- int mvSoluteSegregationDislocations

  Flag indicating whether solute trapping to defects should be considered.
- double mvKappaFactorForCsegInClusterArea

  Mesoscale enrichment ratio at defects (e.g., k = 7∗10^-15).
- int mvAllowDislocationsSegregationInAustenite

  Flag indicating whether solute trapping to defects in the soft phase (e.g., austenite) should be considered.
- int mvIncludeCarbonInterphasePartitioning

  Flag indicating whether solute phase partitioning is to be simulated.
- int mvIsSoluteSegregationHappeningInDiffusionStep

  Flag indicating whether solute trapping to defects is part of the numerical solution of diffusion. Recommended for mesh- and time step-independent simulation.
- int mvIsPartitioningHappeningInDiffusionStep

  Flag indicating whether solute phase partitioning is part of the numerical solution of diffusion. Recommended for mesh- and time step-independent simulation.
- int mvCarbonPartitioningFromInterpolatedSolutions

  Flag indicating whether solute phase partitioning follows thermodynamic inputs for local concentrations. Turn off means using pre-defined analytical expressions.
- int mvAllowInterfaceMovementDuringPartitioning

  Flag indicating whether phase transformation simulation is enabled.
- int mvConcentrationDependentDiffusivityInAustenite

  Flag indicating whether diffusivity follows Agren's concentration-dependent expression (useful when simulating carbon).
- double mvXcToUseConstantDiffusivityAgrenInAustenite

  Carbon concentration that is used as "effective" value at which Agren's concentration-dependent expression gives diffusivity (useful when the simulation leads to high localized carbon contents which would then require very high time steps).
- double mvAverageCarbon

  Concentration of solute (carbon or other) that redistributes through the simulation (atomic fraction).
- double mvBurgersVectorBCC

BCC lattice Burgers vector for the material investigated (m).

- double mvBurgersVectorFCC

  FCC lattice Burgers vector for the material investigated (m).
- double mvBurgersVectorHCP

  HCP lattice Burgers vector for the material investigated (m).
- double mvTimeRangeToWriteOutput

  Time range to write output (s).
- double mvSoluteDiffusivityActivationEnergy

  Activation energy for solute diffusion (J/mol).
- double mvSoluteDiffusivityPreFactor

  Pre-exponential factor for solute diffusion (J/mol).
- double mvGrainBoundaryMobilityProExponential

  Pre-exponential factor for grain boundary migration (J/mol).
- double mvGrainBoundaryMobilityActivationEnergy

  Activation energy for grain boundary migration (J/mol).
- double mvGrainBoundaryEnergy

  Grain boundary energy density (J/m^2).
- double mvPhaseBoundaryMobilityProExponential

  Pre-exponential factor for phase boundary migration.
- double mvPhaseBoundaryMobilityActivationEnergy

  Activation energy for phase boundary migration (J/mol).
- double mvPhaseBoundaryEnergy

  Phase boundary energy density (J/m^2).

3.6.1 Detailed Description

Handles all the simulation parameters and sets them according to the input file and/or default values.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 UserSettings()

```
UserSettings::UserSettings (
            const std::string & inputFile )
```

Constructor that initializes settings from the input file.

Parameters

inputFile	Path to the input file containing simulation parameters.

3.6.3 Member Function Documentation

3.6.3.1 getOutputFolderPath()

```
std::string UserSettings::getOutputFolderPath ( ) const  [inline]
```

Returns the path to the folder where simulation outputs are stored.

Returns

Path to the output folder.

3.6.3.2 getStartFileName()

```
std::string UserSettings::getStartFileName ( ) const  [inline]
```

Returns the path to the input (starting) microstructure file.

Returns

Path to the input microstructure file.

3.6.3.3 getThermodynamicDataFile()

```
std::string UserSettings::getThermodynamicDataFile ( ) const  [inline]
```

Returns the path to the file containing thermodynamic data.

Returns

Path to the thermodynamic data file.

3.6.3.4 writeToFile()

```
void UserSettings::writeToFile (
            const std::string & outputFile )
```

Writes the settings to the output file, including default values.

Parameters

outputFile	Path to the output file where settings will be saved.

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

- src/settings.h
- src/settings.cpp

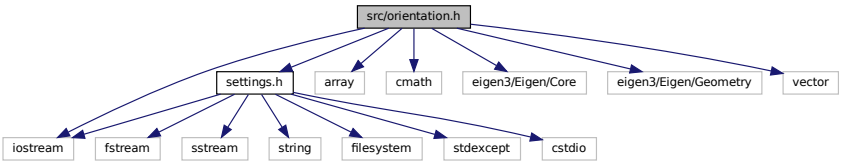
Chapter 4

File Documentation

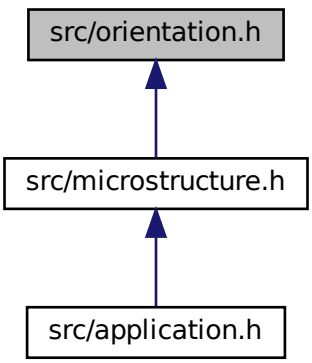
4.1 src/orientation.h File Reference

```
#include "settings.h"
#include <array>
#include <cmath>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/Geometry>
#include <iostream>
#include <vector>
```

Include dependency graph for orientation.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Orientations

  Class to handle orientations and symmetry operations.

Macros

- #define AA2Q(A, X, Y, Z) Eigen::Quaternionf(Eigen::AngleAxisf(A, Eigen::Vector3f(X, Y, Z) / sqrt(X ∗ X + Y ∗ Y + Z ∗ Z))).normalized()

  Convert rotation in the angle-axis representation to quaternion.

Variables

Constants

- const float MISORIENTATION_MAX = 64.0f

  Maximum misorientation angle. 63.8 degrees for cubic symmetry.

Symmetry and Orientation Relationship Operators

- const int N_SYM = 24

  Number of symmetry operators for the cubic system.
- const Eigen::Quaternionf SYM [N_SYM]

  Cubic symmetry operators (0-23)
- const int N_CSL19a = N_SYM

  Number of CSL19a equivalent representations (i.e., 27 degrees around <110>) orientation relationships.
- const Eigen::Quaternionf CSL19a [N_CSL19a]

4.1.1 Detailed Description

OMicroN (Optimizing Microstructures Numerically) simulation program. Header file containing the Orientations class definitions and implementation.

4.1.2 Variable Documentation

4.1.2.1 CSL19a

```
const Eigen::Quaternionf CSL19a[N_CSL19a]
```

Initial value:

```
= {
    AA2Q(0.463f, -0.7071f, 0.0f, 0.7071f),
    CSL19a[0] * SYM[1],
    CSL19a[0] * SYM[2],
    CSL19a[0] * SYM[3],
    CSL19a[0] * SYM[4],
    CSL19a[0] * SYM[5],
    CSL19a[0] * SYM[6],
    CSL19a[0] * SYM[7],
    CSL19a[0] * SYM[8],
    CSL19a[0] * SYM[9],
    CSL19a[0] * SYM[10],
    CSL19a[0] * SYM[11],
    CSL19a[0] * SYM[12],
    CSL19a[0] * SYM[13],
    CSL19a[0] * SYM[14],
    CSL19a[0] * SYM[15],
    CSL19a[0] * SYM[16],
    CSL19a[0] * SYM[17],
    CSL19a[0] * SYM[18],
    CSL19a[0] * SYM[19],
    CSL19a[0] * SYM[20],
    CSL19a[0] * SYM[21],
    CSL19a[0] * SYM[22],
    CSL19a[0] * SYM[23]}
```

Symmetrically equivalent operators to apply the CSL19a (Ibe and Lucke)

4.1.2.2 SYM

```
const Eigen::Quaternionf SYM[N_SYM]
```

Initial value:

```
= {
    AA2Q(0.0f, 1.0f, 0.0f, 0.0f),
    AA2Q(M_PI, 1.0f, 0.0f, 0.0f),
    AA2Q(M_PI, 0.0f, 1.0f, 0.0f),
    AA2Q(M_PI, 0.0f, 0.0f, 1.0f),
    AA2Q(M_PI / 2.0f, 1.0f, 0.0f, 0.0f),
    AA2Q(M_PI / 2.0f, 0.0f, 1.0f, 0.0f),
    AA2Q(M_PI / 2.0f, 0.0f, 0.0f, 1.0f),
    AA2Q(M_PI / 2.0f, -1.0f, 0.0f, 0.0f),
    AA2Q(M_PI / 2.0f, 0.0f, -1.0f, 0.0f),
    AA2Q(M_PI / 2.0f, 0.0f, 0.0f, -1.0f),
    AA2Q(M_PI, 1.0f, 1.0f, 0.0f),
    AA2Q(M_PI, 1.0f, 0.0f, 1.0f),
    AA2Q(M_PI, 0.0f, 1.0f, 1.0f),
    AA2Q(M_PI, 1.0f, -1.0f, 0.0f),
    AA2Q(M_PI, -1.0f, 0.0f, 1.0f),
    AA2Q(M_PI, 0.0f, 1.0f, -1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, 1.0f, 1.0f, 1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, 1.0f, -1.0f, 1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, -1.0f, 1.0f, 1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, -1.0f, -1.0f, 1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, 1.0f, 1.0f, -1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, 1.0f, -1.0f, -1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, -1.0f, 1.0f, -1.0f),
    AA2Q(M_PI * 2.0f / 3.0f, -1.0f, -1.0f, -1.0f)}
```

Cubic symmetry operators (0-23)

Index