WaveSimulator

Generated by Doxygen 1.8.11

Contents

1 Wavepropagation-Simulation on Android-Systems 1

1.1 Intro . . . 1

1.2 Functions . . . 1

1.2.1 Creating Waves . . . 1

1.2.2 Creating Obstacles . . . 1

1.2.3 Obstacles . . . 1

1.2.4 Boundaries . . . 1

1.2.5 Resetting . . . 1

2 Deprecated List 3

3 Namespace Index 5

3.1 Namespace List . . . 5

4 Hierarchical Index 7

4.1 Class Hierarchy . . . 7

5 Class Index 9

5.1 Class List . . . 9

6 File Index 11

6.1 File List . . . 11

7 Namespace Documentation 13

7.1 Package Solver . . . 13

7.1.1 Detailed Description . . . 13

7.2 Package Solver . . . 13

7.2.1 Detailed Description . . . 13

8 Class Documentation 15

8.1 Solver.CPPSimulator Class Reference . . . 15

8.1.1 Detailed Description . . . 16

8.1.2 Constructor & Destructor Documentation . . . 16

8.1.2.1 CPPSimulator() . . . 16

8.1.3 Member Function Documentation . . . 16

8.1.3.1 delete(int x, int y, int r) . . . 16

8.1.3.2 finalize() . . . 16

8.1.3.3 getBathymetry(int x, int y) . . . 16

8.1.3.4 getHeight(int x, int y) . . . 17

8.1.3.5 placeCircle(int x, int y, int r) . . . 17

8.1.3.6 reset() . . . 17

8.1.3.7 resetWaves() . . . 17

8.1.3.8 setBoundaryType(boolean isWall) . . . 17

8.1.3.9 setWave(int x, int y, int r, float h) . . . 17

8.1.3.10 simulatetimestep() . . . 17

8.1.4 Member Data Documentation . . . 17

8.1.4.1 cell_count . . . 17

8.1.4.2 sim . . . 18

8.1.4.3 waterlevel . . . 18

8.2 Float1D Class Reference . . . 18

8.2.1 Detailed Description . . . 18

8.2.2 Constructor & Destructor Documentation . . . 18

8.2.2.1 Float1D(float ∗_elem, int _rows, int _stride=1) . . . 18

8.2.2.2 ∼Float1D() . . . 19

8.2.3 Member Function Documentation . . . 19

8.2.3.1 elemVector() . . . 19

8.2.3.2 getSize() const . . . 19

8.2.3.3 operator[](int i) . . . 19

8.2.3.4 operator[](int i) const . . . 19

8.3 Float2D Class Reference 19
8.3.1 Detailed Description 20
8.3.2 Constructor & Destructor Documentation 20
8.3.2.1 Float2D(int _cols, int _rows, bool _allocateMemory=true) 20
8.3.2.2 Float2D(int _cols, int _rows, float ∗_elem) 20
8.3.2.3 Float2D(Float2D &_elem, bool shallowCopy) 20
8.3.2.4 ∼Float2D() 21
8.3.3 Member Function Documentation 21
8.3.3.1 elemVector() 21
8.3.3.2 getColProxy(int i) 21
8.3.3.3 getCols() const 21
8.3.3.4 getRowProxy(int j) 21
8.3.3.5 getRows() const 21
8.3.3.6 operator[](int i) 21
8.3.3.7 operator[](int i) const 21
8.4 Solver.Helper Class Reference 22
8.4.1 Detailed Description 22
8.4.2 Member Function Documentation 22
8.4.2.1 linear_map(float x1, float y1, float x2, float y2, float number) 22
8.5 Solver.SimulationRunner Class Reference 22
8.5.1 Detailed Description 22
8.5.2 Constructor & Destructor Documentation 23
8.5.2.1 SimulationRunner(MainActivity currentActivity) 23
8.5.3 Member Function Documentation 23
8.5.3.1 changeActivity(MainActivity a) 23
8.5.3.2 isStarted() 23
8.5.3.3 start() 23
8.5.3.4 stop() 23
8.6 SWE_Block Class Reference 24
8.6.1 Detailed Description 27

8.6.2 Constructor & Destructor Documentation . . . 28
8.6.2.1 SWE_Block(int l_nx, int l_ny, float l_dx, float l_dy) . . . 28
8.6.2.2 ∼SWE_Block() . . . 28
8.6.3 Member Function Documentation . . . 28
8.6.3.1 computeMaxTimestep(const float i_dryTol=0.1, const float i_cflNumber=0.4) . . . 28
8.6.3.2 computeNumericalFluxes()=0 . . . 28
8.6.3.3 getBathymetry() . . . 29
8.6.3.4 getDischarge_hu() . . . 29
8.6.3.5 getDischarge_hv() . . . 29
8.6.3.6 getMaxTimestep() . . . 29
8.6.3.7 getNx() . . . 29
8.6.3.8 getNy() . . . 29
8.6.3.9 getWaterHeight() . . . 29
8.6.3.10 grabGhostLayer(BoundaryEdge edge) . . . 30
8.6.3.11 initScenario(float _offsetX, float _offsetY, SWE_Scenario &i_scenario, const bool i_multipleBlocks=false) . . . 30
8.6.3.12 registerCopyLayer(BoundaryEdge edge) . . . 30
8.6.3.13 setBathymetry(float _b) . . . 31
8.6.3.14 setBathymetry(float(∗_b)(float, float)) . . . 31
8.6.3.15 setBathymetryXY(int x, int y, float h_set) . . . 31
8.6.3.16 setBoundaryBathymetry() . . . 31
8.6.3.17 setBoundaryConditions() . . . 31
8.6.3.18 setBoundaryType(BoundaryEdge edge, BoundaryType boundtype, const SWE←_Block1D ∗inflow=NULL) . . . 31
8.6.3.19 setDischarge(float(∗_u)(float, float), float(∗_v)(float, float)) . . . 32
8.6.3.20 setGhostLayer() . . . 32
8.6.3.21 setHuXY(int x, int y, float h_set) . . . 32
8.6.3.22 setHvXY(int x, int y, float h_set) . . . 32
8.6.3.23 setWaterHeight(float(∗_h)(float, float)) . . . 32
8.6.3.24 setWaterHeightXY(int x, int y, float h_set) . . . 32
8.6.3.25 simulate(float tStart, float tEnd) . . . 32

8.6.3.26 simulateTimestep(float dt) . . . 33
8.6.3.27 synchAfterWrite() . . . 33
8.6.3.28 synchBathymetryAfterWrite() . . . 33
8.6.3.29 synchBathymetryBeforeRead() . . . 33
8.6.3.30 synchBeforeRead() . . . 34
8.6.3.31 synchCopyLayerBeforeRead() . . . 34
8.6.3.32 synchDischargeAfterWrite() . . . 34
8.6.3.33 synchDischargeBeforeRead() . . . 34
8.6.3.34 synchGhostLayerAfterWrite() . . . 34
8.6.3.35 synchWaterHeightAfterWrite() . . . 34
8.6.3.36 synchWaterHeightBeforeRead() . . . 34
8.6.3.37 updateUnknowns(float dt)=0 . . . 34
8.6.4 Member Data Documentation . . . 35
8.6.4.1 b . . . 35
8.6.4.2 boundary . . . 35
8.6.4.3 dx . . . 35
8.6.4.4 dy . . . 35
8.6.4.5 g . . . 35
8.6.4.6 h . . . 35
8.6.4.7 hu . . . 36
8.6.4.8 hv . . . 36
8.6.4.9 maxTimestep . . . 36
8.6.4.10 neighbour . . . 36
8.6.4.11 nx . . . 36
8.6.4.12 ny . . . 36
8.6.4.13 offsetX . . . 36
8.6.4.14 offsetY . . . 37
8.7 SWE_Block1D Class Reference . . . 37
8.7.1 Detailed Description . . . 38
8.7.2 Constructor & Destructor Documentation . . . 38

8.7.2.1 SWE_Block1D(const Float1D &_h, const Float1D &_hu, const Float1D &_hv) . . 38
8.7.2.2 SWE_Block1D(float ∗_h, float ∗_hu, float ∗_hv, int _size, int _stride=1) 38
8.7.3 Member Data Documentation 38
8.7.3.1 h 38
8.7.3.2 hu 38
8.7.3.3 hv 38
8.8 SWE_FlatScenario Class Reference 39
8.8.1 Detailed Description 39
8.8.2 Member Function Documentation 40
8.8.2.1 endSimulation() 40
8.8.2.2 getBathymetry(float x, float y) 40
8.8.2.3 getBoundaryPos(BoundaryEdge i_edge) 40
8.8.2.4 getBoundaryType(BoundaryEdge edge) 40
8.8.2.5 getWaterHeight(float x, float y) 40
8.9 SWE_Scenario Class Reference 41
8.9.1 Detailed Description 41
8.9.2 Constructor & Destructor Documentation 42
8.9.2.1 ∼SWE_Scenario() 42
8.9.2.2 ∼SWE_Scenario() 42
8.9.3 Member Function Documentation 42
8.9.3.1 endSimulation() 42
8.9.3.2 endSimulation() 42
8.9.3.3 getBathymetry(float x, float y) 42
8.9.3.4 getBathymetry(float x, float y) 42
8.9.3.5 getBoundaryPos(BoundaryEdge edge) 42
8.9.3.6 getBoundaryPos(BoundaryEdge edge) 43
8.9.3.7 getBoundaryType(BoundaryEdge edge) 43
8.9.3.8 getBoundaryType(BoundaryEdge edge) 43
8.9.3.9 getVeloc_u(float x, float y) 43
8.9.3.10 getVeloc_u(float x, float y) 43

8.9.3.11 getVeloc_v(float x, float y) 43

8.9.3.12 getVeloc_v(float x, float y) 43

8.9.3.13 getWaterHeight(float x, float y) 43

8.9.3.14 getWaterHeight(float x, float y) 43

8.9.3.15 waterHeightAtRest() 44

8.9.3.16 waterHeightAtRest() 44

8.10 SWE_WavePropagationBlock Class Reference 44

8.10.1 Detailed Description 45

8.10.2 Constructor & Destructor Documentation 46

8.10.2.1 SWE_WavePropagationBlock(int l_nx, int l_ny, float l_dx, float l_dy) 46

8.10.2.2 ∼SWE_WavePropagationBlock() 46

8.10.3 Member Function Documentation 47

8.10.3.1 computeNumericalFluxes() 47

8.10.3.2 updateUnknowns(float dt) 47

8.10.3.3 updateUnknownsRow(float dt, int i) 47

8.11 solver::WavePropagation< T > Class Template Reference 47

8.11.1 Detailed Description 48

8.11.2 Member Enumeration Documentation 49

8.11.2.1 WetDryState 49

8.11.3 Constructor & Destructor Documentation 49

8.11.3.1 WavePropagation(T i_dryTolerance, T i_gravity, T i_zeroTolerance) 49

8.11.3.2 ∼WavePropagation() 49

8.11.4 Member Function Documentation 49

8.11.4.1 computeNetUpdates(const T &i_hLeft, const T &i_hRight, const T &i_huLeft, const T &i_huRight, const T &i_bLeft, const T &i_bRight, T &o_hUpdateLeft, T &o_hUpdateRight, T &o_huUpdateLeft, T &o_huUpdateRight, T &o_maxWave←Speed)=0 50

8.11.4.2 determineWetDryState()=0 50

8.11.4.3 setDryTolerance(const T i_dryTolerance) 50

8.11.4.4 storeParameters(const T &i_hLeft, const T &i_hRight, const T &i_huLeft, const T &i_huRight, const T &i_bLeft, const T &i_bRight) 50

8.11.4.5 storeParameters(const T &i_hLeft, const T &i_hRight, const T &i_huLeft, const T &i_huRight, const T &i_bLeft, const T &i_bRight, const T &i_uLeft, const T &i_←uRight) 51
8.11.5 Member Data Documentation 51
8.11.5.1 bLeft 51
8.11.5.2 bRight 51
8.11.5.3 dryTol 51
8.11.5.4 g 52
8.11.5.5 hLeft 52
8.11.5.6 hRight 52
8.11.5.7 huLeft 52
8.11.5.8 huRight 52
8.11.5.9 uLeft 52
8.11.5.10 uRight 52
8.11.5.11 wetDryState 52
8.11.5.12 zeroTol 53
8.12 WavePropagation Class Reference 53
8.12.1 Detailed Description 53

9 File Documentation 55

9.1 app/src/main/cpp/blocks/SWE_Block.cpp File Reference 55
9.1.1 Detailed Description 55
9.1.2 LICENSE 56
9.1.3 DESCRIPTION 56
9.2 app/src/main/cpp/blocks/SWE_Block.hh File Reference 56
9.2.1 Detailed Description 57
9.2.2 LICENSE 57
9.2.3 DESCRIPTION 58
9.3 app/src/main/cpp/blocks/SWE_WavePropagationBlock.cpp File Reference 58
9.3.1 Detailed Description 58
9.3.2 LICENSE 58
9.3.3 DESCRIPTION 59

9.4 app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh File Reference 59
9.4.1 Detailed Description 60
9.4.2 LICENSE 60
9.4.3 DESCRIPTION 60
9.5 app/src/main/cpp/blocks/WavePropagation.cpp File Reference 60
9.6 app/src/main/cpp/blocks/WavePropagation.hpp File Reference 61
9.7 app/src/main/cpp/scenarios/SWE_Scenario.hh File Reference 61
9.7.1 Detailed Description 62
9.7.2 LICENSE 62
9.7.3 DESCRIPTION 62
9.7.4 Typedef Documentation 62
9.7.4.1 BoundaryEdge 62
9.7.4.2 BoundaryType 63
9.7.5 Enumeration Type Documentation 63
9.7.5.1 BoundaryEdge 63
9.7.5.2 BoundaryType 63
9.8 app/src/main/cpp/tools/SWE_Scenario.hh File Reference 64
9.8.1 Typedef Documentation 65
9.8.1.1 BoundaryEdge 65
9.8.1.2 BoundaryType 65
9.8.2 Enumeration Type Documentation 65
9.8.2.1 BoundaryEdge 65
9.8.2.2 BoundaryType 65
9.9 app/src/main/cpp/scenarios/SWE_simple_scenarios.hh File Reference 66
9.9.1 Detailed Description 66
9.9.2 LICENSE 66
9.9.3 DESCRIPTION 67
9.10 app/src/main/cpp/tools/help.hh File Reference 67
9.10.1 Detailed Description 68
9.10.2 LICENSE 68

9.10.3 DESCRIPTION 68
9.10.4 Function Documentation 68
9.10.4.1 generateBaseFileName(std::string &i_baseName, int i_blockPositionX, int i_←blockPositionY) 68
9.10.4.2 generateContainerFileName(std::string baseName, int timeStep) 69
9.10.4.3 generateFileName(std::string baseName, int timeStep) 69
9.10.4.4 generateFileName(std::string i_baseName, int i_blockPositionX, int i_block←PositionY, std::string i_fileExtension="".nc"") 69
9.10.4.5 generateFileName(std::string baseName, int timeStep, int block_X, int block_Y, std::string i_fileExtension="".vts"") 69
9.11 app/src/main/DocMain.cpp File Reference 70
9.11.1 Function Documentation 70
9.11.1.1 main() 70
9.12 app/src/main/java/Solver/CPPSimulator.java File Reference 70
9.13 app/src/main/java/Solver/Helper.java File Reference 70
9.14 app/src/main/java/Solver/SimulationRunner.java File Reference 70

Index **71**

Chapter 1

Wavepropagation-Simulation on Android-Systems

Author

Gregor, Martin and Sven

1.1 Intro

This project uses the open-source SWE-implementation, to allow the user to set up any desired Tsunami scenario, placeable objects in an given domain of calmed water.

1.2 Functions

1.2.1 Creating Waves

The user can create waves of variable height by tapping the domain. The height can be adjusted via the Wave-↩ Hight-Slider below the water domain.

1.2.2 Creating Obstacles

The user can create equally high obstacles by selecting the draw mode(brush Symbol in the top right corner). While pressing once in this mode will result in a circular obstacle, moving around allows for the creation of any desired shape.

1.2.3 Obstacles

This can be done in the same fashion as the creation, as long as the Erase_Mode is chosen(minus-symbol on top of the domain). Tapping once will activate the Erase_Mode. Tapping again will change the mode back to normal.

1.2.4 Boundaries

The domain-boundaries can be chosen to be reflective or open. This is done by tapping the "Reflective Boundaries" switch located below the domain.

1.2.5 Resetting

The user can choose to reset only the waves or the whole domain. This is done by tapping the options button in the top right corner of the screen and then choosing the wanted option.

Chapter 2

Deprecated List

Member generateFileName (std::string baseName, int timeStep)

Member generateFileName (std::string baseName, int timeStep, int block_X, int block_Y, std::string i_file↩ Extension=".vts")

Member generateFileName (std::string i_baseName, int i_blockPositionX, int i_blockPositionY, std::string i_fileExtension=".nc")

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

solver
This namespace defines code that defines the SWE framework . . . . ??
Solver
The Solver entails all classes dedicated to the mathematical backend of our implementation . . 13

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Solver.CPPSimulator 15
Float1D 18
Float2D 19
Solver.Helper 22
Solver.SimulationRunner 22
SWE_Block 24
    SWE_WavePropagationBlock 44
SWE_Block1D 37
SWE_Scenario 41
    SWE_FlatScenario 39
solver::WavePropagation$<$ T $>$ 47
WavePropagation 53

Chapter 5

Class Index

5.1 Class List

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

Solver.CPPSimulator
    This class interfaces with the SWE-Code(c++); 15
Float1D
    This class is part of SWE; It gives our representation of a one dimensional datatype for update calculations 18
Float2D
    This class is part of SWE; It gives our representation of atwo dimensional datatype for update calculations 19
Solver.Helper
    This class contains a helper method that maps a number of the x domain onto the y domain via linearisation; 22
Solver.SimulationRunner
    This class handles the Threading of the Simulation 22
SWE_Block
    This file is part of SWE 24
SWE_Block1D
    This file is part SWE 37
SWE_FlatScenario
    Scenario is a test environment that simulates a flat water-surface 39
SWE_Scenario
    This class sets the standart layout for all simulateable scenarios 41
SWE_WavePropagationBlock
    File is part of SWE 44
solver::WavePropagation< T > 47
WavePropagation
    Abstract wave propagation solver for the Shallow Water Equations; T should be double or float; 53

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

app/src/main/DocMain.cpp 70
app/src/main/cpp/blocks/SWE_Block.cpp 55
app/src/main/cpp/blocks/SWE_Block.hh 56
app/src/main/cpp/blocks/SWE_WavePropagationBlock.cpp 58
app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh 59
app/src/main/cpp/blocks/WavePropagation.cpp 60
app/src/main/cpp/blocks/WavePropagation.hpp 61
app/src/main/cpp/scenarios/SWE_Scenario.hh 61
app/src/main/cpp/scenarios/SWE_simple_scenarios.hh 66
app/src/main/cpp/tools/help.hh 67
app/src/main/cpp/tools/SWE_Scenario.hh 64
app/src/main/java/Solver/CPPSimulator.java 70
app/src/main/java/Solver/Helper.java 70
app/src/main/java/Solver/SimulationRunner.java 70

Chapter 7

Namespace Documentation

7.1 Package Solver

a The Solver entails all classes dedicated to the mathematical backend of our implementation

Classes

- class CPPSimulator

  This class interfaces with the SWE-Code(c++);.
- class Helper

  This class contains a helper method that maps a number of the x domain onto the y domain via linearisation;.
- class SimulationRunner

  This class handles the Threading of the Simulation.

7.1.1 Detailed Description

a The Solver entails all classes dedicated to the mathematical backend of our implementation

7.2 Package Solver

a The Solver entails all classes dedicated to the mathematical backend of our implementation

Classes

- class CPPSimulator

  This class interfaces with the SWE-Code(c++);.
- class Helper

  This class contains a helper method that maps a number of the x domain onto the y domain via linearisation;.
- class SimulationRunner

  This class handles the Threading of the Simulation.

7.2.1 Detailed Description

a The Solver entails all classes dedicated to the mathematical backend of our implementation

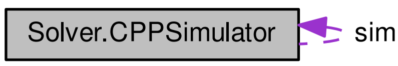
Chapter 8

Class Documentation

8.1 Solver.CPPSimulator Class Reference

This class interfaces with the SWE-Code(c++);.

Collaboration diagram for Solver.CPPSimulator:



Public Member Functions

- CPPSimulator ()
- synchronized void setWave (int x, int y, int r, float h)
- synchronized void placeCircle (int x, int y, int r)

    Positions the Bathymetry-Circle where needed.
- void setBoundaryType (boolean isWall)

    Adjust boudary-type.
- float getHeight (int x, int y)
- float getBathymetry (int x, int y)
- void delete (int x, int y, int r)
- synchronized void simulatetimestep ()

Static Public Member Functions

- static void reset ()

    resets the simulation
- static synchronized void resetWaves ()

Static Public Attributes

- static final float waterlevel = 5
- static final int cell_count = 100
- static CPPSimulator sim

    have to be static to allow screen rotation

Protected Member Functions

- void finalize ()

    free up Memory

8.1.1 Detailed Description

This class interfaces with the SWE-Code(c++);.

Definition at line 11 of file CPPSimulator.java.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 Solver.CPPSimulator.CPPSimulator ( ) [inline]

$<$ load libary

$<$ if its the first initialization reset the simulation

Definition at line 22 of file CPPSimulator.java.

8.1.3 Member Function Documentation

8.1.3.1 void Solver.CPPSimulator.delete (int *x*, int *y*, int *r*) [inline]

Definition at line 59 of file CPPSimulator.java.

8.1.3.2 void Solver.CPPSimulator.finalize ( ) [inline], [protected]

free up Memory

Definition at line 74 of file CPPSimulator.java.

8.1.3.3 float Solver.CPPSimulator.getBathymetry (int *x*, int *y*) [inline]

Definition at line 55 of file CPPSimulator.java.

8.1.3.4 float Solver.CPPSimulator.getHeight (int *x*, int *y*) [inline]

Definition at line 51 of file CPPSimulator.java.

8.1.3.5 synchronized void Solver.CPPSimulator.placeCircle (int *x*, int *y*, int *r*) [inline]

Positions the Bathymetry-Circle where needed.

Definition at line 35 of file CPPSimulator.java.

8.1.3.6 static void Solver.CPPSimulator.reset ( ) [inline], [static]

resets the simulation

< creates a new SWE_Block Object

Definition at line 45 of file CPPSimulator.java.

8.1.3.7 static synchronized void Solver.CPPSimulator.resetWaves ( ) [inline], [static]

Definition at line 69 of file CPPSimulator.java.

8.1.3.8 void Solver.CPPSimulator.setBoundaryType (boolean *isWall*) [inline]

Adjust boudary-type.

Definition at line 40 of file CPPSimulator.java.

8.1.3.9 synchronized void Solver.CPPSimulator.setWave (int *x*, int *y*, int *r*, float *h*) [inline]

< SWE_Pointer = setWave(x,y,r,h,SWE_Pointer);

Definition at line 30 of file CPPSimulator.java.

8.1.3.10 synchronized void Solver.CPPSimulator.simulatetimestep ( ) [inline]

Definition at line 63 of file CPPSimulator.java.

8.1.4 Member Data Documentation

8.1.4.1 final int Solver.CPPSimulator.cell_count = 100 [static]

Definition at line 14 of file CPPSimulator.java.

8.1.4.2 CPPSimulator Solver.CPPSimulator.sim [static]

have to be static to allow screen rotation

Definition at line 19 of file CPPSimulator.java.

8.1.4.3 final float Solver.CPPSimulator.waterlevel = 5 [static]

Definition at line 13 of file CPPSimulator.java.

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

- app/src/main/java/Solver/CPPSimulator.java

8.2 Float1D Class Reference

This class is part of SWE; It gives our representation of a one dimensional datatype for update calculations.

`#include <help.hh>`

Public Member Functions

- Float1D (float ∗_elem, int _rows, int _stride=1)
- ∼Float1D ()
- float & operator[ ] (int i)
- const float & operator[ ] (int i) const
- float ∗ elemVector ()
- int getSize () const

8.2.1 Detailed Description

This class is part of SWE; It gives our representation of a one dimensional datatype for update calculations.

class Float1D is a proxy class that can represent, for example, a column or row vector of a Float2D array, where row (sub-)arrays are stored with a respective stride. Besides constructor/deconstructor, the class provides overloading of the []-operator, such that elements can be accessed as v[i] (independent of the stride). The class will never allocate separate memory for the vectors, but point to the interior data structure of Float2D (or other "host" data structures).

Definition at line 49 of file help.hh.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Float1D::Float1D (float ∗ *_elem*, int *_rows*, int *_stride* = 1) [inline]

Definition at line 52 of file help.hh.

8.2.2.2 Float1D::∼Float1D ( ) [inline]

Definition at line 57 of file help.hh.

8.2.3 Member Function Documentation

8.2.3.1 float∗ Float1D::elemVector ( ) [inline]

Definition at line 69 of file help.hh.

8.2.3.2 int Float1D::getSize ( ) const [inline]

Definition at line 73 of file help.hh.

8.2.3.3 float& Float1D::operator[ ] (int *i*) [inline]

Definition at line 61 of file help.hh.

8.2.3.4 const float& Float1D::operator[ ] (int *i*) const [inline]

Definition at line 65 of file help.hh.

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

- app/src/main/cpp/tools/help.hh

8.3 Float2D Class Reference

This class is part of SWE; It gives our representation of atwo dimensional datatype for update calculations.

`#include <help.hh>`

Public Member Functions

- Float2D (int _cols, int _rows, bool _allocateMemory=true)
- Float2D (int _cols, int _rows, float ∗_elem)
- Float2D (Float2D &_elem, bool shallowCopy)
- ∼Float2D ()
- float ∗ operator[ ] (int i)
- float const ∗ operator[ ] (int i) const
- float ∗ elemVector ()
- int getRows () const
- int getCols () const
- Float1D getColProxy (int i)
- Float1D getRowProxy (int j)

8.3.1 Detailed Description

This class is part of SWE; It gives our representation of atwo dimensional datatype for update calculations.

class Float2D is a very basic helper class to deal with 2D float arrays: indices represent columns (1st index, "horizontal"/x-coordinate) and rows (2nd index, "vertical"/y-coordinate) of a 2D grid; values are sequentially ordered in memory using "column major" order. Besides constructor/deconstructor, the class provides overloading of the []-operator, such that elements can be accessed as a[i][j].

Definition at line 90 of file help.hh.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 Float2D::Float2D (int *_cols*, int *_rows*, bool *_allocateMemory* = `true`) `[inline]`

Constructor: takes size of the 2D array as parameters and creates a respective Float2D object; allocates memory for the array, but does not initialise value.

Parameters

_cols	number of columns (i.e., elements in horizontal direction)
_rows	rumber of rows (i.e., elements in vertical directions)

Definition at line 99 of file help.hh.

8.3.2.2 Float2D::Float2D (int *_cols*, int *_rows*, float ∗ *_elem*) `[inline]`

Constructor: takes size of the 2D array as parameters and creates a respective Float2D object; this constructor does not allocate memory for the array, but uses the allocated memory provided via the respective variable #_elem

Parameters

_cols	number of columns (i.e., elements in horizontal direction)
_rows	rumber of rows (i.e., elements in vertical directions)
_elem	pointer to a suitably allocated region of memory to be used for thew array elements

Definition at line 117 of file help.hh.

8.3.2.3 Float2D::Float2D (Float2D & *_elem*, bool *shallowCopy*) `[inline]`

Constructor: takes size of the 2D array as parameters and creates a respective Float2D object; this constructor does not allocate memory for the array, but uses the allocated memory provided via the respective variable #_elem

Parameters

_cols	number of columns (i.e., elements in horizontal direction)
_rows	rumber of rows (i.e., elements in vertical directions)
_elem	pointer to a suitably allocated region of memory to be used for thew array elements

Definition at line 134 of file help.hh.

8.3.2.4 Float2D::∼Float2D ( ) [inline]

Definition at line 151 of file help.hh.

8.3.3 Member Function Documentation

8.3.3.1 float∗ Float2D::elemVector ( ) [inline]

Definition at line 165 of file help.hh.

8.3.3.2 Float1D Float2D::getColProxy (int *i*) [inline]

Definition at line 172 of file help.hh.

8.3.3.3 int Float2D::getCols ( ) const [inline]

Definition at line 170 of file help.hh.

8.3.3.4 Float1D Float2D::getRowProxy (int *j*) [inline]

Definition at line 178 of file help.hh.

8.3.3.5 int Float2D::getRows ( ) const [inline]

Definition at line 169 of file help.hh.

8.3.3.6 float∗ Float2D::operator[ ] (int *i*) [inline]

Definition at line 157 of file help.hh.

8.3.3.7 float const∗ Float2D::operator[ ] (int *i*) const [inline]

Definition at line 161 of file help.hh.

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

- app/src/main/cpp/tools/help.hh

8.4 Solver.Helper Class Reference

This class contains a helper method that maps a number of the x domain onto the y domain via linearisation;.

Static Public Member Functions

- static float linear_map (float x1, float y1, float x2, float y2, float number)

8.4.1 Detailed Description

This class contains a helper method that maps a number of the x domain onto the y domain via linearisation;.

Definition at line 3 of file Helper.java.

8.4.2 Member Function Documentation

8.4.2.1 static float Solver.Helper.linear_map (float *x1,* float *y1,* float *x2,* float *y2,* float *number*) [inline], [static]

Definition at line 5 of file Helper.java.

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

- app/src/main/java/Solver/Helper.java

8.5 Solver.SimulationRunner Class Reference

This class handles the Threading of the Simulation.

Public Member Functions

- SimulationRunner (MainActivity currentActivity)
- void start ()

  starts the simulation
- void changeActivity (MainActivity a)
- void stop ()
- boolean isStarted ()

8.5.1 Detailed Description

This class handles the Threading of the Simulation.

Definition at line 6 of file SimulationRunner.java.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Solver.SimulationRunner.SimulationRunner (MainActivity *currentActivity*) [inline]

Definition at line 11 of file SimulationRunner.java.

8.5.3 Member Function Documentation

8.5.3.1 void Solver.SimulationRunner.changeActivity (MainActivity *a*) [inline]

Definition at line 47 of file SimulationRunner.java.

8.5.3.2 boolean Solver.SimulationRunner.isStarted ( ) [inline]

Definition at line 55 of file SimulationRunner.java.

8.5.3.3 void Solver.SimulationRunner.start ( ) [inline]

starts the simulation

< get the current time

< simulate a single timestep

< redraw view

< calculate the duration of the timestepsimulation

< sleep if took less time then sleeptime

< starts the created thread

Definition at line 17 of file SimulationRunner.java.

8.5.3.4 void Solver.SimulationRunner.stop ( ) [inline]

Definition at line 51 of file SimulationRunner.java.

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

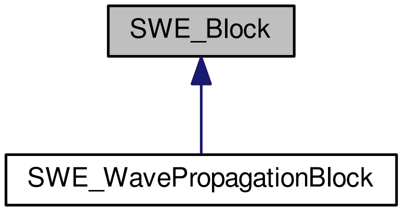
- app/src/main/java/Solver/SimulationRunner.java

8.6 SWE_Block Class Reference

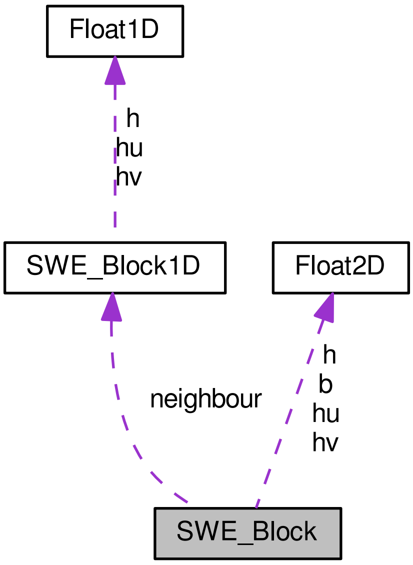
This file is part of SWE.

`#include <SWE_Block.hh>`

Inheritance diagram for SWE_Block:



Collaboration diagram for SWE_Block:



Public Member Functions

- void initScenario (float _offsetX, float _offsetY, SWE_Scenario &i_scenario, const bool i_multipleBlocks=false)

  initialise unknowns to a specific scenario:

- void setWaterHeight (float(∗_h)(float, float))

  set the water height according to a given function
- void setDischarge (float(∗_u)(float, float), float(∗_v)(float, float))

  set the momentum/discharge according to the provided functions
- void setBathymetry (float _b)

  set the bathymetry to a uniform value
- void setBathymetry (float(∗_b)(float, float))

  set the bathymetry according to a given function
- void setWaterHeightXY (int x, int y, float h_set)

  set a aditional wave with radius r and height h at x,y
- void setBathymetryXY (int x, int y, float h_set)
- void setHuXY (int x, int y, float h_set)
- void setHvXY (int x, int y, float h_set)
- const Float2D & getWaterHeight ()

  provides read access to the water height array
- const Float2D & getDischarge_hu ()

  provides read access to the momentum/discharge array (x-component)
- const Float2D & getDischarge_hv ()

  provides read access to the momentum/discharge array (y-component)
- const Float2D & getBathymetry ()

  provides read access to the bathymetry data
- void setBoundaryType (BoundaryEdge edge, BoundaryType boundtype, const SWE_Block1D ∗inflow=NULL)

  set type of boundary condition for the specified boundary
- virtual SWE_Block1D ∗ registerCopyLayer (BoundaryEdge edge)

  return a pointer to proxy class to access the copy layer
- virtual SWE_Block1D ∗ grabGhostLayer (BoundaryEdge edge)

  "grab" the ghost layer in order to set these values externally
- void setGhostLayer ()

  set values in ghost layers
- float getMaxTimestep ()

  return maximum size of the time step to ensure stability of the method
- void computeMaxTimestep (const float i_dryTol=0.1, const float i_cflNumber=0.4)
- virtual void simulateTimestep (float dt)

  execute a single time step (with fixed time step size) of the simulation
- virtual float simulate (float tStart, float tEnd)
- virtual void computeNumericalFluxes ()=0

  compute the numerical fluxes for each edge of the Cartesian grid
- virtual void updateUnknowns (float dt)=0

  compute the new values of the unknowns h, hu, and hv in all grid cells
- int getNx ()

  returns nx, i.e. the grid size in x-direction
- int getNy ()

  returns ny, i.e. the grid size in y-direction

Static Public Attributes

- static const float g = 9.81f

  static variable that holds the gravity constant (g = 9.81 m/s^2):

Protected Member Functions

- SWE_Block (int l_nx, int l_ny, float l_dx, float l_dy)
- virtual ∼SWE_Block ()
- void setBoundaryBathymetry ()
- virtual void synchAfterWrite ()
- virtual void synchWaterHeightAfterWrite ()
- virtual void synchDischargeAfterWrite ()
- virtual void synchBathymetryAfterWrite ()
- virtual void synchGhostLayerAfterWrite ()
- virtual void synchBeforeRead ()
- virtual void synchWaterHeightBeforeRead ()
- virtual void synchDischargeBeforeRead ()
- virtual void synchBathymetryBeforeRead ()
- virtual void synchCopyLayerBeforeRead ()
- virtual void setBoundaryConditions ()

    set boundary conditions in ghost layers (set boundary conditions)

Protected Attributes

- int nx

    size of Cartesian arrays in x-direction
- int ny

    size of Cartesian arrays in y-direction
- float dx

    mesh size of the Cartesian grid in x-direction
- float dy

    mesh size of the Cartesian grid in y-direction
- Float2D h

    array that holds the water height for each element
- Float2D hu

    array that holds the x-component of the momentum for each element (water height h multiplied by velocity in x-direction)
- Float2D hv

    array that holds the y-component of the momentum for each element (water height h multiplied by velocity in y-direction)
- Float2D b

    array that holds the bathymetry data (sea floor elevation) for each element
- BoundaryType boundary [4]

    type of boundary conditions at LEFT, RIGHT, TOP, and BOTTOM boundary
- const SWE_Block1D ∗ neighbour [4]

    for CONNECT boundaries: pointer to connected neighbour block
- float maxTimestep

    maximum time step allowed to ensure stability of the method
- float offsetX

    x-coordinate of the origin (left-bottom corner) of the Cartesian grid
- float offsetY

    y-coordinate of the origin (left-bottom corner) of the Cartesian grid

8.6.1 Detailed Description

This file is part of SWE.

SWE_Block is the main data structure to compute our shallow water model on a single Cartesian grid block: SWE_Block is an abstract class (and interface) that should be extended by respective implementation classes.

Cartesian Grid for Discretization:

SWE_Blocks uses a regular Cartesian grid of size nx by ny, where each grid cell carries three unknowns:

- the water level h
- the momentum components hu and hv (in x- and y- direction, resp.)
- the bathymetry b

Each of the components is stored as a 2D array, implemented as a Float2D object, and are defined on grid indices [0,..,nx+1]∗[0,..,ny+1]. The computational domain is indexed with [1,..,nx]∗[1,..,ny].

The mesh sizes of the grid in x- and y-direction are stored in static variables dx and dy. The position of the Cartesian grid in space is stored via the coordinates of the left-bottom corner of the grid, in the variables offsetX and offsetY.

Ghost layers:

To implement the behaviour of the fluid at boundaries and for using multiple block in serial and parallel settings, SWE_Block adds an additional layer of so-called ghost cells to the Cartesian grid, as illustrated in the following figure. Cells in the ghost layer have indices 0 or nx+1 / ny+1.

Memory Model:

The variables h, hu, hv for water height and momentum will typically be updated by classes derived from SWE_Block. However, it is not assumed that such and updated will be performed in every time step. Instead, subclasses are welcome to update h, hu, and hv in a lazy fashion, and keep data in faster memory (incl. local memory of acceleration hardware, such as GPGPUs), instead.

It is assumed that the bathymetry data b is not changed during the algorithm (up to the exceptions mentioned in the following).

To force a synchronization of the respective data structures, the following methods are provided as part of SWE_Block:

- synchAfterWrite() to synchronize h, hu, hv, and b after an external update (reading a file, e.g.);
- synchWaterHeightAfterWrite(), synchDischargeAfterWrite(), synchBathymetryAfterWrite(): to synchronize only h or momentum (hu and hv) or bathymetry b;
- synchGhostLayerAfterWrite() to synchronize only the ghost layers
- synchBeforeRead() to synchronize h, hu, hv, and b before an output of the variables (writing a visualization file, e.g.)
- synchWaterHeightBeforeRead(), synchDischargeBeforeRead(), synchBathymetryBeforeRead(): as synchBeforeRead(), but only for the specified variables
- synchCopyLayerBeforeRead(): synchronizes the copy layer only (i.e., a layer that is to be replicated in a neighbouring SWE_Block.

Derived Classes

As SWE_Block just provides an abstract base class together with the most important data structures, the implementation of concrete models is the job of respective derived classes (see the class diagram at the top of this page). Similar, parallel implementations that are based on a specific parallel programming model (such as OpenMP) or parallel architecture (such as GPU/CUDA) should form subclasses of their own. Please refer to the documentation of these classes for more details on the model and on the parallelisation approach.

Definition at line 115 of file SWE_Block.hh.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 SWE_Block::SWE_Block (int *l_nx,* int *l_ny,* float *l_dx,* float *l_dy*) `[protected]`

Constructor: allocate variables for simulation

unknowns h (water height), hu,hv (discharge in x- and y-direction), and b (bathymetry) are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

The constructor is protected: no instances of SWE_Block can be generated.

Definition at line 52 of file SWE_Block.cpp.

8.6.2.2 SWE_Block::∼SWE_Block ( ) `[protected]`, `[virtual]`

Destructor: de-allocate all variables

Definition at line 70 of file SWE_Block.cpp.

8.6.3 Member Function Documentation

8.6.3.1 void SWE_Block::computeMaxTimestep (const float *i_dryTol* = `0.1`, const float *i_cflNumber* = `0.4`)

Compute the largest allowed time step for the current grid block (reference implementation) depending on the current values of variables h, hu, and hv, and store this time step size in member variable maxTimestep.

Parameters

i_dryTol	dry tolerance (dry cells do not affect the time step).
i_cflNumber	CFL number of the used method.

Definition at line 490 of file SWE_Block.cpp.

8.6.3.2 virtual void SWE_Block::computeNumericalFluxes ( ) `[pure virtual]`

compute the numerical fluxes for each edge of the Cartesian grid

The computation of fluxes strongly depends on the chosen numerical method. Hence, this purely virtual function has to be implemented in the respective derived classes.

Implemented in SWE_WavePropagationBlock.

8.6.3.3 const Float2D & SWE_Block::getBathymetry ( )

provides read access to the bathymetry data

return reference to bathymetry unknown b

Definition at line 259 of file SWE_Block.cpp.

8.6.3.4 const Float2D & SWE_Block::getDischarge_hu ( )

provides read access to the momentum/discharge array (x-component)

return reference to discharge unknown hu

Definition at line 243 of file SWE_Block.cpp.

8.6.3.5 const Float2D & SWE_Block::getDischarge_hv ( )

provides read access to the momentum/discharge array (y-component)

return reference to discharge unknown hv

Definition at line 251 of file SWE_Block.cpp.

8.6.3.6 float SWE_Block::getMaxTimestep ( ) [inline]

return maximum size of the time step to ensure stability of the method

Returns

current value of the member variable maxTimestep

Definition at line 166 of file SWE_Block.hh.

8.6.3.7 int SWE_Block::getNx ( ) [inline]

returns nx, i.e. the grid size in x-direction

Definition at line 199 of file SWE_Block.hh.

8.6.3.8 int SWE_Block::getNy ( ) [inline]

returns ny, i.e. the grid size in y-direction

Definition at line 201 of file SWE_Block.hh.

8.6.3.9 const Float2D & SWE_Block::getWaterHeight ( )

provides read access to the water height array

Restores values for h, v, and u from file data

Parameters

↩ _↩ *b*	array holding b-values in sequence return reference to water height unknown h

Definition at line 235 of file SWE_Block.cpp.

8.6.3.10 SWE_Block1D ∗ SWE_Block::grabGhostLayer (BoundaryEdge *edge*) [virtual]

"grab" the ghost layer in order to set these values externally

"grab" the ghost layer at the specific boundary in order to set boundary values in this ghost layer externally. The boundary conditions at the respective ghost layer is set to PASSIVE, such that the grabbing program component is responsible to provide correct values in the ghost layer, for example by receiving data from a remote copy layer via MPI communication.

Parameters

specified	edge

Returns

a SWE_Block1D object that contains row variables h, hu, and hv

Definition at line 398 of file SWE_Block.cpp.

8.6.3.11 void SWE_Block::initScenario (float *_offsetX,* float *_offsetY,* SWE_Scenario & *i_scenario,* const bool *i_multipleBlocks* = false)

initialise unknowns to a specific scenario:

Initializes the unknowns and bathymetry in all grid cells according to the given SWE_Scenario.

In the case of multiple SWE_Blocks at this point, it is not clear how the boundary conditions should be set. This is because an isolated SWE_Block doesn't have any in information about the grid. Therefore the calling routine, which has the information about multiple blocks, has to take care about setting the right boundary conditions.

Parameters

i_scenario	scenario, which is used during the setup.
i_multipleBlocks	are the multiple SWE_blocks?

Definition at line 90 of file SWE_Block.cpp.

8.6.3.12 SWE_Block1D ∗ SWE_Block::registerCopyLayer (BoundaryEdge *edge*) [virtual]

return a pointer to proxy class to access the copy layer register the row or column layer next to a boundary as a "copy layer", from which values will be copied into the ghost layer or a neighbour;

Returns

a SWE_Block1D object that contains row variables h, hu, and hv

Definition at line 373 of file SWE_Block.cpp.

8.6.3.13 void SWE_Block::setBathymetry (float *_b*)

set the bathymetry to a uniform value

set Bathymetry b in all grid cells (incl. ghost/boundary layers) to a uniform value bathymetry source terms are re-computed

Definition at line 179 of file SWE_Block.cpp.

8.6.3.14 void SWE_Block::setBathymetry (float(∗)(float, float) *_b*)

set the bathymetry according to a given function

set Bathymetry b in all grid cells (incl. ghost/boundary layers) using the specified bathymetry function; bathymetry source terms are re-computed

Definition at line 193 of file SWE_Block.cpp.

8.6.3.15 void SWE_Block::setBathymetryXY (int *x*, int *y*, float *h_set*)

Definition at line 131 of file SWE_Block.cpp.

8.6.3.16 void SWE_Block::setBoundaryBathymetry ( ) [protected]

Sets the bathymetry on OUTFLOW or WALL boundaries. Should be called very time a boundary is changed to a OUTFLOW or WALL boundary **or** the bathymetry changes.

Definition at line 337 of file SWE_Block.cpp.

8.6.3.17 void SWE_Block::setBoundaryConditions ( ) [protected], [virtual]

set boundary conditions in ghost layers (set boundary conditions)

set the values of all ghost cells depending on the specifed boundary conditions

- set boundary conditions for typs WALL and OUTFLOW
- derived classes need to transfer ghost layers

Definition at line 534 of file SWE_Block.cpp.

8.6.3.18 void SWE_Block::setBoundaryType (BoundaryEdge *edge*, BoundaryType *boundtype*, const SWE_Block1D ∗ *i_inflow* = NULL)

set type of boundary condition for the specified boundary

Set the boundary type for specific block boundary.

Parameters

i_edge	location of the edge relative to the SWE_block.
i_boundaryType	type of the boundary condition.
i_inflow	pointer to an SWE_Block1D, which specifies the inflow (should be NULL for WALL or OUTFLOW boundary)

Definition at line 320 of file SWE_Block.cpp.

8.6.3.19 void SWE_Block::setDischarge (float(∗)(float, float) *_u*, float(∗)(float, float) *_v*)

set the momentum/discharge according to the provided functions

set discharge in all interior grid cells (i.e. except ghost layer) to values specified by parameter functions Note: unknowns hu and hv represent momentum, while parameters u and v are velocities!

Definition at line 161 of file SWE_Block.cpp.

8.6.3.20 void SWE_Block::setGhostLayer ( )

set values in ghost layers

set the values of all ghost cells depending on the specifed boundary conditions; if the ghost layer replicates the variables of a remote SWE_Block, the values are copied

Definition at line 421 of file SWE_Block.cpp.

8.6.3.21 void SWE_Block::setHuXY (int *x*, int *y*, float *h_set*)

Definition at line 135 of file SWE_Block.cpp.

8.6.3.22 void SWE_Block::setHvXY (int *x*, int *y*, float *h_set*)

Definition at line 139 of file SWE_Block.cpp.

8.6.3.23 void SWE_Block::setWaterHeight (float(∗)(float, float) *_h*)

set the water height according to a given function

set water height h in all interior grid cells (i.e. except ghost layer) to values specified by parameter function _h

Definition at line 147 of file SWE_Block.cpp.

8.6.3.24 void SWE_Block::setWaterHeightXY (int *x*, int *y*, float *h_set*)

set a aditional wave with radius r and height h at x,y

Definition at line 127 of file SWE_Block.cpp.

8.6.3.25 float SWE_Block::simulate (float *i_tStart*, float *i_tEnd*) [virtual]

perform the simulation starting with simulation time tStart, until simulation time tEnd is reached

simulate implements the main simulation loop between two checkpoints; Note: this implementation can only be used, if you only use a single SWE_Block and only apply simple boundary conditions! In particular, SWE_Block↩ ::simulate can not trigger calls to exchange values of copy and ghost layers between blocks!

Parameters

tStart	time where the simulation is started
tEnd	time of the next checkpoint

Returns

actual end time reached

Definition at line 293 of file SWE_Block.cpp.

8.6.3.26 void SWE_Block::simulateTimestep (float *dt*) [virtual]

execute a single time step (with fixed time step size) of the simulation

Executes a single timestep with fixed time step size

- compute net updates for every edge
- update cell values with the net updates

Parameters

dt	time step width of the update

Definition at line 276 of file SWE_Block.cpp.

8.6.3.27 void SWE_Block::synchAfterWrite ( ) [protected], [virtual]

Update all temporary and non-local (for heterogeneous computing) variables after an external update of the main variables h, hu, hv, and b.

Definition at line 708 of file SWE_Block.cpp.

8.6.3.28 void SWE_Block::synchBathymetryAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the bathymetry b

Definition at line 730 of file SWE_Block.cpp.

8.6.3.29 void SWE_Block::synchBathymetryBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the bathymetry b

Definition at line 765 of file SWE_Block.cpp.

8.6.3.30 void SWE_Block::synchBeforeRead ( ) [protected], [virtual]

Update all temporary and non-local (for heterogeneous computing) variables before an external access to the main variables h, hu, hv, and b.

Definition at line 743 of file SWE_Block.cpp.

8.6.3.31 void SWE_Block::synchCopyLayerBeforeRead ( ) [protected], [virtual]

Update (for heterogeneous computing) variables in copy layers before an external access to the unknowns

Definition at line 771 of file SWE_Block.cpp.

8.6.3.32 void SWE_Block::synchDischargeAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the discharge variables hu and hv

Definition at line 724 of file SWE_Block.cpp.

8.6.3.33 void SWE_Block::synchDischargeBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the discharge variables hu and hv

Definition at line 759 of file SWE_Block.cpp.

8.6.3.34 void SWE_Block::synchGhostLayerAfterWrite ( ) [protected], [virtual]

Update the ghost layers (only for CONNECT and PASSIVE boundary conditions) after an external update of the main variables h, hu, hv, and b in the ghost layer.

Definition at line 737 of file SWE_Block.cpp.

8.6.3.35 void SWE_Block::synchWaterHeightAfterWrite ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables after an external update of the water height h

Definition at line 718 of file SWE_Block.cpp.

8.6.3.36 void SWE_Block::synchWaterHeightBeforeRead ( ) [protected], [virtual]

Update temporary and non-local (for heterogeneous computing) variables before an external access to the water height h

Definition at line 753 of file SWE_Block.cpp.

8.6.3.37 virtual void SWE_Block::updateUnknowns (float *dt*) [pure virtual]

compute the new values of the unknowns h, hu, and hv in all grid cells

based on the numerical fluxes (computed by computeNumericalFluxes) and the specified time step size dt, an Euler time step is executed. As the computational fluxes will depend on the numerical method, this purely virtual function has to be implemented separately for each specific numerical model (and parallelisation approach).

Parameters

dt	size of the time step

Implemented in SWE_WavePropagationBlock.

8.6.4 Member Data Documentation

8.6.4.1 Float2D SWE_Block::b [protected]

array that holds the bathymetry data (sea floor elevation) for each element

Definition at line 245 of file SWE_Block.hh.

8.6.4.2 BoundaryType SWE_Block::boundary[4] [protected]

type of boundary conditions at LEFT, RIGHT, TOP, and BOTTOM boundary

Definition at line 248 of file SWE_Block.hh.

8.6.4.3 float SWE_Block::dx [protected]

mesh size of the Cartesian grid in x-direction

Definition at line 236 of file SWE_Block.hh.

8.6.4.4 float SWE_Block::dy [protected]

mesh size of the Cartesian grid in y-direction

Definition at line 237 of file SWE_Block.hh.

8.6.4.5 const float SWE_Block::g = 9.81f [static]

static variable that holds the gravity constant (g = 9.81 m/s^2):

Definition at line 205 of file SWE_Block.hh.

8.6.4.6 Float2D SWE_Block::h [protected]

array that holds the water height for each element

Definition at line 242 of file SWE_Block.hh.

8.6.4.7 Float2D SWE_Block::hu [protected]

array that holds the x-component of the momentum for each element (water height h multiplied by velocity in x-direction)

Definition at line 243 of file SWE_Block.hh.

8.6.4.8 Float2D SWE_Block::hv [protected]

array that holds the y-component of the momentum for each element (water height h multiplied by velocity in y-direction)

Definition at line 244 of file SWE_Block.hh.

8.6.4.9 float SWE_Block::maxTimestep [protected]

maximum time step allowed to ensure stability of the method

maxTimestep can be updated as part of the methods computeNumericalFluxes and updateUnknowns (depending on the numerical method)

Definition at line 257 of file SWE_Block.hh.

8.6.4.10 const SWE_Block1D* SWE_Block::neighbour[4] [protected]

for CONNECT boundaries: pointer to connected neighbour block

Definition at line 250 of file SWE_Block.hh.

8.6.4.11 int SWE_Block::nx [protected]

size of Cartesian arrays in x-direction

Definition at line 233 of file SWE_Block.hh.

8.6.4.12 int SWE_Block::ny [protected]

size of Cartesian arrays in y-direction

Definition at line 234 of file SWE_Block.hh.

8.6.4.13 float SWE_Block::offsetX [protected]

x-coordinate of the origin (left-bottom corner) of the Cartesian grid

Definition at line 260 of file SWE_Block.hh.

8.6.4.14 float SWE_Block::offsetY `[protected]`

y-coordinate of the origin (left-bottom corner) of the Cartesian grid

Definition at line 261 of file SWE_Block.hh.

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

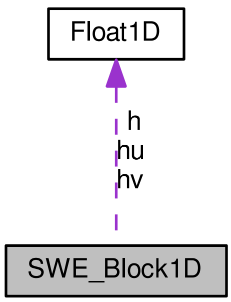
- app/src/main/cpp/blocks/SWE_Block.hh
- app/src/main/cpp/blocks/SWE_Block.cpp

8.7 SWE_Block1D Class Reference

This file is part SWE.

`#include <SWE_Block.hh>`

Collaboration diagram for SWE_Block1D:



Public Member Functions

- SWE_Block1D (const Float1D &_h, const Float1D &_hu, const Float1D &_hv)
- SWE_Block1D (float ∗_h, float ∗_hu, float ∗_hv, int _size, int _stride=1)

Public Attributes

- Float1D h
- Float1D hu
- Float1D hv

8.7.1 Detailed Description

This file is part SWE.

SWE_Block1D is a simple struct that can represent a single line or row of SWE_Block unknowns (using the Float1D proxy class). It is intended to unify the implementation of inflow and periodic boundary conditions, as well as the ghost/copy-layer connection between several SWE_Block grids.

Definition at line 271 of file SWE_Block.hh.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 SWE_Block1D::SWE_Block1D (const Float1D & *_h*, const Float1D & *_hu*, const Float1D & *_hv*) [inline]

Definition at line 272 of file SWE_Block.hh.

8.7.2.2 SWE_Block1D::SWE_Block1D (float ∗ *_h*, float ∗ *_hu*, float ∗ *_hv*, int *_size*, int *_stride* = 1) [inline]

Definition at line 274 of file SWE_Block.hh.

8.7.3 Member Data Documentation

8.7.3.1 Float1D SWE_Block1D::h

Definition at line 275 of file SWE_Block.hh.

8.7.3.2 Float1D SWE_Block1D::hu

Definition at line 278 of file SWE_Block.hh.

8.7.3.3 Float1D SWE_Block1D::hv

Definition at line 279 of file SWE_Block.hh.

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

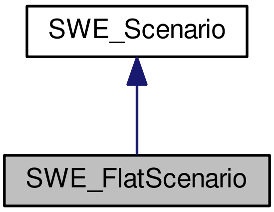
- app/src/main/cpp/blocks/SWE_Block.hh

8.8 SWE_FlatScenario Class Reference

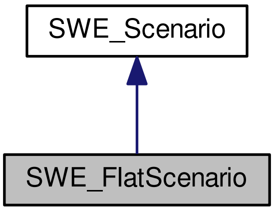
scenario is a test environment that simulates a flat water-surface

`#include <SWE_simple_scenarios.hh>`

Inheritance diagram for SWE_FlatScenario:



Collaboration diagram for SWE_FlatScenario:



Public Member Functions

- float getBathymetry (float x, float y)
- float getWaterHeight (float x, float y)
- virtual float endSimulation ()
- virtual BoundaryType getBoundaryType (BoundaryEdge edge)
- float getBoundaryPos (BoundaryEdge i_edge)

8.8.1 Detailed Description

scenario is a test environment that simulates a flat water-surface

Scenario "Radial Dam Break": elevated water in the center of the domain

Definition at line 41 of file SWE_simple_scenarios.hh.

8.8.2 Member Function Documentation

8.8.2.1 virtual float SWE_FlatScenario::endSimulation ( ) [inline], [virtual]

Reimplemented from SWE_Scenario.

Definition at line 54 of file SWE_simple_scenarios.hh.

8.8.2.2 float SWE_FlatScenario::getBathymetry (float *x*, float *y*) [inline], [virtual]

Reimplemented from SWE_Scenario.

Definition at line 45 of file SWE_simple_scenarios.hh.

8.8.2.3 float SWE_FlatScenario::getBoundaryPos (BoundaryEdge *i_edge*) [inline], [virtual]

Get the boundary positions

Parameters

i_edge	which edge

Returns

value in the corresponding dimension

Reimplemented from SWE_Scenario.

Definition at line 63 of file SWE_simple_scenarios.hh.

8.8.2.4 virtual BoundaryType SWE_FlatScenario::getBoundaryType (BoundaryEdge *edge*) [inline], [virtual]

Reimplemented from SWE_Scenario.

Definition at line 56 of file SWE_simple_scenarios.hh.

8.8.2.5 float SWE_FlatScenario::getWaterHeight (float *x*, float *y*) [inline], [virtual]

Reimplemented from SWE_Scenario.

Definition at line 49 of file SWE_simple_scenarios.hh.

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

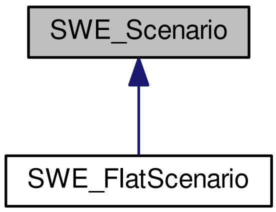
- app/src/main/cpp/scenarios/SWE_simple_scenarios.hh

8.9 SWE_Scenario Class Reference

This class sets the standart layout for all simulateable scenarios.

`#include <SWE_Scenario.hh>`

Inheritance diagram for SWE_Scenario:



Public Member Functions

- virtual float getWaterHeight (float x, float y)
- virtual float getVeloc_u (float x, float y)
- virtual float getVeloc_v (float x, float y)
- virtual float getBathymetry (float x, float y)
- virtual float waterHeightAtRest ()
- virtual float endSimulation ()
- virtual BoundaryType getBoundaryType (BoundaryEdge edge)
- virtual float getBoundaryPos (BoundaryEdge edge)
- virtual ∼SWE_Scenario ()
- virtual float getWaterHeight (float x, float y)
- virtual float getVeloc_u (float x, float y)
- virtual float getVeloc_v (float x, float y)
- virtual float getBathymetry (float x, float y)
- virtual float waterHeightAtRest ()
- virtual float endSimulation ()
- virtual BoundaryType getBoundaryType (BoundaryEdge edge)
- virtual float getBoundaryPos (BoundaryEdge edge)
- virtual ∼SWE_Scenario ()

8.9.1 Detailed Description

This class sets the standart layout for all simulateable scenarios.

SWE_Scenario defines an interface to initialise the unknowns of a shallow water simulation - i.e. to initialise water height, velocities, and bathymatry according to certain scenarios. SWE_Scenario can act as stand-alone scenario class, providing a very basic scenario (all functions are constant); however, the idea is to provide derived classes that implement the SWE_Scenario interface for more interesting scenarios.

Definition at line 55 of file SWE_Scenario.hh.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 virtual SWE_Scenario::∼SWE_Scenario ( ) [inline], [virtual]

Definition at line 76 of file SWE_Scenario.hh.

8.9.2.2 virtual SWE_Scenario::∼SWE_Scenario ( ) [inline], [virtual]

Definition at line 75 of file SWE_Scenario.hh.

8.9.3 Member Function Documentation

8.9.3.1 virtual float SWE_Scenario::endSimulation ( ) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 65 of file SWE_Scenario.hh.

8.9.3.2 virtual float SWE_Scenario::endSimulation ( ) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 66 of file SWE_Scenario.hh.

8.9.3.3 virtual float SWE_Scenario::getBathymetry (float *x*, float *y*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 61 of file SWE_Scenario.hh.

8.9.3.4 virtual float SWE_Scenario::getBathymetry (float *x*, float *y*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 62 of file SWE_Scenario.hh.

8.9.3.5 virtual float SWE_Scenario::getBoundaryPos (BoundaryEdge *edge*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 68 of file SWE_Scenario.hh.

8.9.3.6 virtual float SWE_Scenario::getBoundaryPos (**BoundaryEdge** *edge*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 69 of file SWE_Scenario.hh.

8.9.3.7 virtual **BoundaryType** SWE_Scenario::getBoundaryType (**BoundaryEdge** *edge*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 67 of file SWE_Scenario.hh.

8.9.3.8 virtual **BoundaryType** SWE_Scenario::getBoundaryType (**BoundaryEdge** *edge*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 68 of file SWE_Scenario.hh.

8.9.3.9 virtual float SWE_Scenario::getVeloc_u (float *x*, float *y*) [inline], [virtual]

Definition at line 59 of file SWE_Scenario.hh.

8.9.3.10 virtual float SWE_Scenario::getVeloc_u (float *x*, float *y*) [inline], [virtual]

Definition at line 60 of file SWE_Scenario.hh.

8.9.3.11 virtual float SWE_Scenario::getVeloc_v (float *x*, float *y*) [inline], [virtual]

Definition at line 60 of file SWE_Scenario.hh.

8.9.3.12 virtual float SWE_Scenario::getVeloc_v (float *x*, float *y*) [inline], [virtual]

Definition at line 61 of file SWE_Scenario.hh.

8.9.3.13 virtual float SWE_Scenario::getWaterHeight (float *x*, float *y*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 58 of file SWE_Scenario.hh.

8.9.3.14 virtual float SWE_Scenario::getWaterHeight (float *x*, float *y*) [inline], [virtual]

Reimplemented in SWE_FlatScenario.

Definition at line 59 of file SWE_Scenario.hh.

8.9.3.15 virtual float SWE_Scenario::waterHeightAtRest ( ) [inline], [virtual]

Definition at line 63 of file SWE_Scenario.hh.

8.9.3.16 virtual float SWE_Scenario::waterHeightAtRest ( ) [inline], [virtual]

Definition at line 64 of file SWE_Scenario.hh.

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

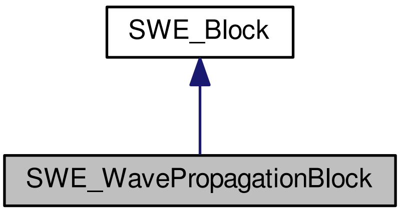
- app/src/main/cpp/scenarios/SWE_Scenario.hh

8.10 SWE_WavePropagationBlock Class Reference

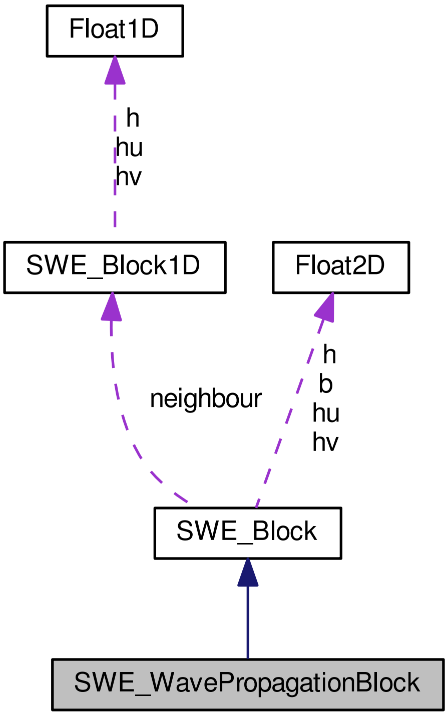
file is part of SWE

`#include <SWE_WavePropagationBlock.hh>`

Inheritance diagram for SWE_WavePropagationBlock:



Collaboration diagram for SWE_WavePropagationBlock:



Public Member Functions

- SWE_WavePropagationBlock (int l_nx, int l_ny, float l_dx, float l_dy)
- void computeNumericalFluxes ()
- void updateUnknowns (float dt)
- void updateUnknownsRow (float dt, int i)
- virtual ∼SWE_WavePropagationBlock ()

Additional Inherited Members

8.10.1 Detailed Description

file is part of SWE

SWE_WavePropagationBlock is an implementation of the SWE_Block abstract class. It uses a wave propagation solver which is defined with the pre-compiler flag WAVE_PROPAGATION_SOLVER (see above).

Possible wave propagation solvers are: F-Wave, Apprximate Augmented Riemann, Hybrid (f-wave + augmented). (details can be found in the corresponding source files)

Definition at line 53 of file SWE_WavePropagationBlock.hh.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 SWE_WavePropagationBlock::SWE_WavePropagationBlock (int *l_nx*, int *l_ny*, float *l_dx*, float *l_dy*)

Constructor of a SWE_WavePropagationBlock.

Allocates the variables for the simulation: unknowns h,hu,hv,b are defined on grid indices [0,..,nx+1]∗[0,..,ny+1] (-> Abstract class SWE_Block) -> computational domain is [1,..,nx]∗[1,..,ny] -> plus ghost cell layer

net-updates are defined for edges with indices [0,..,nx]∗[0,..,ny-1] or [0,..,nx-1]∗`0,..,ny`

A left/right net update with index (i-1,j-1) is located on the edge between cells with index (i-1,j) and (i,j):

```
*********************
*          *          *
* (i-1,j)  *   (i,j)   *
*          *          *
*********************


          *
         ***
        *****
          *
          *
NetUpdatesLeft(i-1,j-1)
          or
NetUpdatesRight(i-1,j-1)
```

A below/above net update with index (i-1, j-1) is located on the edge between cells with index (i, j-1) and (i,j):

```
*          *
* (i, j)   *   *
*          *  **  NetUpdatesBelow(i-1,j-1)
*********** *****         or
*          *  **  NetUpdatesAbove(i-1,j-1)
* (i,j-1)  *   *
*          *
```

Definition at line 80 of file SWE_WavePropagationBlock.cpp.

8.10.2.2 virtual SWE_WavePropagationBlock::∼SWE_WavePropagationBlock ( ) [inline], [virtual]

Destructor of a SWE_WavePropagationBlock.

In the case of a hybrid solver (NDEBUG not defined) information about the used solvers will be printed.

Definition at line 97 of file SWE_WavePropagationBlock.hh.

8.10.3 Member Function Documentation

8.10.3.1 void SWE_WavePropagationBlock::computeNumericalFluxes ( ) [virtual]

Compute net updates for the block. The member variable maxTimestep will be updated with the maximum allowed time step size

Implements SWE_Block.

Definition at line 99 of file SWE_WavePropagationBlock.cpp.

8.10.3.2 void SWE_WavePropagationBlock::updateUnknowns (float *dt*) [virtual]

Updates the unknowns with the already computed net-updates.

Parameters

dt	time step width used in the update.

Implements SWE_Block.

Definition at line 171 of file SWE_WavePropagationBlock.cpp.

8.10.3.3 void SWE_WavePropagationBlock::updateUnknownsRow (float *dt*, int *i*)

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

- app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh
- app/src/main/cpp/blocks/SWE_WavePropagationBlock.cpp

8.11 solver::WavePropagation< T > Class Template Reference

```
#include <WavePropagation.hpp>
```

Public Member Functions

- virtual void computeNetUpdates (const T &i_hLeft, const T &i_hRight, const T &i_huLeft, const T &i_hu↩Right, const T &i_bLeft, const T &i_bRight, T &o_hUpdateLeft, T &o_hUpdateRight, T &o_huUpdateLeft, T &o_huUpdateRight, T &o_maxWaveSpeed)=0
- void setDryTolerance (const T i_dryTolerance)
- virtual ~WavePropagation ()

Protected Types

- enum WetDryState {
  DryDry, WetWet, WetDryInundation, WetDryWall,
  WetDryWallInundation, DryWetInundation, DryWetWall, DryWetWallInundation }

Protected Member Functions

- virtual void determineWetDryState ()=0

  Determine the wet/dry-state and set local values if we have to.
- WavePropagation (T i_dryTolerance, T i_gravity, T i_zeroTolerance)
- void storeParameters (const T &i_hLeft, const T &i_hRight, const T &i_huLeft, const T &i_huRight, const T &i_bLeft, const T &i_bRight)
- void storeParameters (const T &i_hLeft, const T &i_hRight, const T &i_huLeft, const T &i_huRight, const T &i_bLeft, const T &i_bRight, const T &i_uLeft, const T &i_uRight)

Protected Attributes

- T dryTol

  numerical definition of "dry".
- const T g

  gravity constant
- const T zeroTol

  numerical definition of zero.
- T hLeft

  height on the left side of the edge (could change during execution).
- T hRight

  height on the right side of the edge (could change during execution).
- T huLeft

  momentum on the left side of the edge (could change during execution).
- T huRight

  momentum on the right side of the edge (could change during execution).
- T bLeft

  bathymetry on the left side of the edge (could change during execution).
- T bRight

  bathymetry on the right side of the edge (could change during execution).
- T uLeft

  velocity on the left side of the edge (computed by determineWetDryState).
- T uRight

  velocity on the right side of the edge (computed by determineWetDryState).
- WetDryState wetDryState

  wet/dry state of our Riemann-problem (determined by determineWetDryState)

8.11.1 Detailed Description

template<typename T>
class solver::WavePropagation< T >

Definition at line 23 of file WavePropagation.hpp.

8.11.2 Member Enumeration Documentation

8.11.2.1 template<typename T > enum solver::WavePropagation::WetDryState [protected]

The wet/dry state of the Riemann-problem.

Enumerator

- ***DryDry*** Both cells are dry.
- ***WetWet*** Both cells are wet.
- ***WetDryInundation*** 1st cell: wet, 2nd cell: dry. 1st cell lies higher than the 2nd one.
- ***WetDryWall*** 1st cell: wet, 2nd cell: dry. 1st cell lies lower than the 2nd one. Momentum is not large enough to overcome the difference.
- ***WetDryWallInundation*** 1st cell: wet, 2nd cell: dry. 1st cell lies lower than the 2nd one. Momentum is large enough to overcome the difference.
- ***DryWetInundation*** 1st cell: dry, 2nd cell: wet. 1st cell lies lower than the 2nd one.
- ***DryWetWall*** 1st cell: dry, 2nd cell: wet. 1st cell lies higher than the 2nd one. Momentum is not large enough to overcome the difference.
- ***DryWetWallInundation*** 1st cell: dry, 2nd cell: wet. 1st cell lies higher than the 2nd one. Momentum is large enough to overcome the difference.

Definition at line 80 of file WavePropagation.hpp.

8.11.3 Constructor & Destructor Documentation

8.11.3.1 template<typename T > solver::WavePropagation< T >::WavePropagation (T *i_dryTolerance*, T *i_gravity*, T *i_zeroTolerance*) [inline], [protected]

Constructor of a wave propagation solver.

Parameters

gravity	gravity constant.
dryTolerance	numerical definition of "dry".
zeroTolerance	numerical definition of zero.

Definition at line 109 of file WavePropagation.hpp.

8.11.3.2 template<typename T > virtual solver::WavePropagation< T >::~WavePropagation ( ) [inline], [virtual]

Definition at line 206 of file WavePropagation.hpp.

8.11.4 Member Function Documentation

8.11.4.1 template<typename T > virtual void solver::WavePropagation< T >::computeNetUpdates (const T & *i_hLeft*, const T & *i_hRight*, const T & *i_huLeft*, const T & *i_huRight*, const T & *i_bLeft*, const T & *i_bRight*, T & *o_hUpdateLeft*, T & *o_hUpdateRight*, T & *o_huUpdateLeft*, T & *o_huUpdateRight*, T & *o_maxWaveSpeed*) [pure virtual]

Compute net updates for the cell on the left/right side of the edge. This is the default method every standalone wave propagation solver should provide.

Parameters

i_hLeft	height on the left side of the edge.
i_hRight	height on the right side of the edge.
i_huLeft	momentum on the left side of the edge.
i_huRight	momentum on the right side of the edge.
i_bLeft	bathymetry on the left side of the edge.
i_bRight	bathymetry on the right side of the edge.
o_hUpdateLeft	will be set to: Net-update for the height of the cell on the left side of the edge.
o_hUpdateRight	will be set to: Net-update for the height of the cell on the right side of the edge.
o_huUpdateLeft	will be set to: Net-update for the momentum of the cell on the left side of the edge.
o_huUpdateRight	will be set to: Net-update for the momentum of the cell on the right side of the edge.
o_maxWaveSpeed	will be set to: Maximum (linearized) wave speed -> Should be used in the CFL-condition.

8.11.4.2 template<typename T > virtual void solver::WavePropagation< T >::determineWetDryState ( ) [protected], [pure virtual]

Determine the wet/dry-state and set local values if we have to.

8.11.4.3 template<typename T > void solver::WavePropagation< T >::setDryTolerance (const T *i_dryTolerance*) [inline]

Sets the dry tolerance of the solver.

Parameters

i_dryTolerance	dry tolerance.

Definition at line 202 of file WavePropagation.hpp.

8.11.4.4 template<typename T > void solver::WavePropagation< T >::storeParameters (const T & *i_hLeft*, const T & *i_hRight*, const T & *i_huLeft*, const T & *i_huRight*, const T & *i_bLeft*, const T & *i_bRight*) [inline], [protected]

Store parameters to member variables.

Parameters

i_hLeft	height on the left side of the edge.
i_hRight	height on the right side of the edge.

Parameters

i_huLeft	momentum on the left side of the edge.
i_huRight	momentum on the right side of the edge.
i_bLeft	bathymetry on the left side of the edge.
i_bRight	bathymetry on the right side of the edge.

Definition at line 126 of file WavePropagation.hpp.

8.11.4.5 template<typename T > void solver::WavePropagation< T >::storeParameters (const T & *i_hLeft*, const T & *i_hRight*, const T & *i_huLeft*, const T & *i_huRight*, const T & *i_bLeft*, const T & *i_bRight*, const T & *i_uLeft*, const T & *i_uRight*) [inline], [protected]

Store parameters to member variables.

Parameters

i_hLeft	height on the left side of the edge.
i_hRight	height on the right side of the edge.
i_huLeft	momentum on the left side of the edge.
i_huRight	momentum on the right side of the edge.
i_bLeft	bathymetry on the left side of the edge.
i_bRight	bathymetry on the right side of the edge.
i_uLeft	velocity on the left side of the edge.
i_uRight	velocity on the right side of the edge.

Definition at line 152 of file WavePropagation.hpp.

8.11.5 Member Data Documentation

8.11.5.1 template<typename T > T solver::WavePropagation< T >::bLeft [protected]

bathymetry on the left side of the edge (could change during execution).

Definition at line 67 of file WavePropagation.hpp.

8.11.5.2 template<typename T > T solver::WavePropagation< T >::bRight [protected]

bathymetry on the right side of the edge (could change during execution).

Definition at line 69 of file WavePropagation.hpp.

8.11.5.3 template<typename T > T solver::WavePropagation< T >::dryTol [protected]

numerical definition of "dry".

Definition at line 31 of file WavePropagation.hpp.

8.11.5.4 template< typename T > const T solver::WavePropagation< T >::g [protected]

gravity constant

Definition at line 33 of file WavePropagation.hpp.

8.11.5.5 template< typename T > T solver::WavePropagation< T >::hLeft [protected]

height on the left side of the edge (could change during execution).

Definition at line 59 of file WavePropagation.hpp.

8.11.5.6 template< typename T > T solver::WavePropagation< T >::hRight [protected]

height on the right side of the edge (could change during execution).

Definition at line 61 of file WavePropagation.hpp.

8.11.5.7 template< typename T > T solver::WavePropagation< T >::huLeft [protected]

momentum on the left side of the edge (could change during execution).

Definition at line 63 of file WavePropagation.hpp.

8.11.5.8 template< typename T > T solver::WavePropagation< T >::huRight [protected]

momentum on the right side of the edge (could change during execution).

Definition at line 65 of file WavePropagation.hpp.

8.11.5.9 template< typename T > T solver::WavePropagation< T >::uLeft [protected]

velocity on the left side of the edge (computed by determineWetDryState).

Definition at line 72 of file WavePropagation.hpp.

8.11.5.10 template< typename T > T solver::WavePropagation< T >::uRight [protected]

velocity on the right side of the edge (computed by determineWetDryState).

Definition at line 74 of file WavePropagation.hpp.

8.11.5.11 template< typename T > WetDryState solver::WavePropagation< T >::wetDryState [protected]

wet/dry state of our Riemann-problem (determined by determineWetDryState)

Definition at line 96 of file WavePropagation.hpp.

8.11.5.12 template< typename T > const T solver::WavePropagation< T >::zeroTol [protected]

numerical definition of zero.

Definition at line 35 of file WavePropagation.hpp.

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

- app/src/main/cpp/blocks/WavePropagation.hpp

8.12 WavePropagation Class Reference

Abstract wave propagation solver for the Shallow Water Equations; T should be double or float;.

8.12.1 Detailed Description

Abstract wave propagation solver for the Shallow Water Equations; T should be double or float;.

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

- app/src/main/cpp/blocks/WavePropagation.hpp

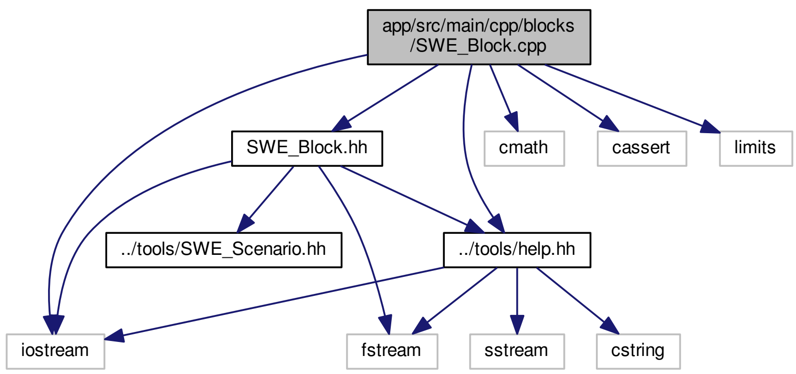
Chapter 9

File Documentation

9.1 app/src/main/cpp/blocks/SWE_Block.cpp File Reference

```
#include "SWE_Block.hh"
#include "../tools/help.hh"
#include <cmath>
#include <iostream>
#include <cassert>
#include <limits>
```

Include dependency graph for SWE_Block.cpp:



9.1.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/↩
Sebastian_Rettenberger,_M.Sc.)

9.1.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.gnu.org/licenses/.

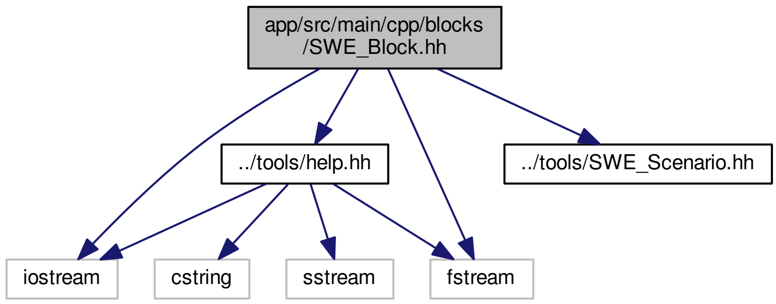
9.1.3 DESCRIPTION

TODO

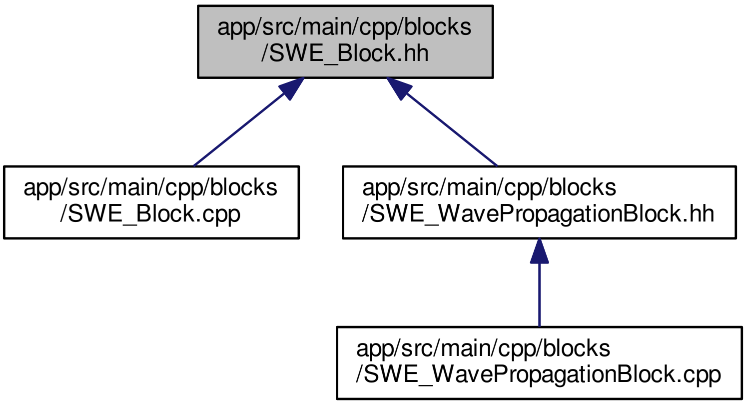
9.2 app/src/main/cpp/blocks/SWE_Block.hh File Reference

```
#include "../tools/help.hh"
#include "../tools/SWE_Scenario.hh"
#include <iostream>
#include <fstream>
```

Include dependency graph for SWE_Block.hh:



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



Classes

- class SWE_Block

  This file is part of SWE.
- class SWE_Block1D

  This file is part SWE.

9.2.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/↩ Sebastian_Rettenberger,_M.Sc.)

9.2.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.↩ gnu.org/licenses/.

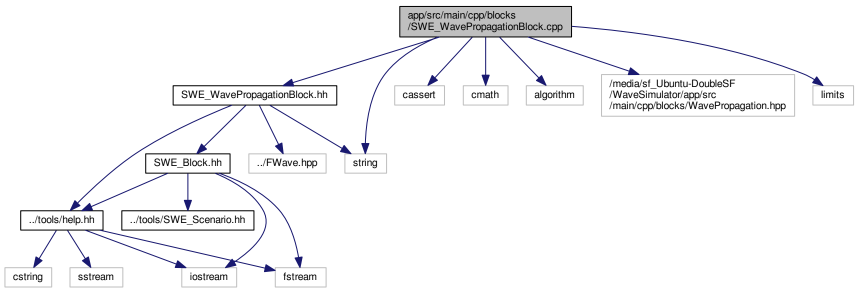
9.2.3 DESCRIPTION

TODO

9.3 app/src/main/cpp/blocks/SWE_WavePropagationBlock.cpp File Reference

```
#include "SWE_WavePropagationBlock.hh"
#include <string>
#include <limits>
```

Include dependency graph for SWE_WavePropagationBlock.cpp:



9.3.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.-Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.)
Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Michael_Bader)

9.3.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.gnu.org/licenses/.

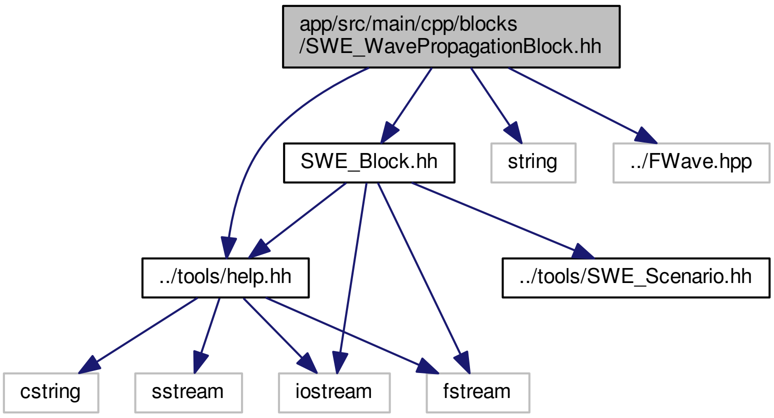
9.3.3 DESCRIPTION

Implementation of SWE_Block that uses solvers in the wave propagation formulation.

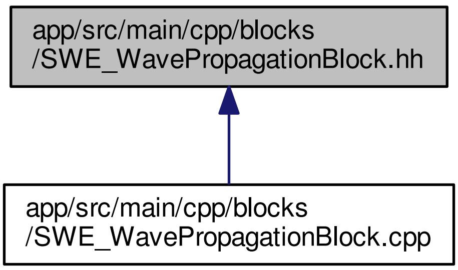
9.4 app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh File Reference

```
#include "SWE_Block.hh"
#include "../tools/help.hh"
#include <string>
#include "../FWave.hpp"
```

Include dependency graph for SWE_WavePropagationBlock.hh:



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



Classes

- class SWE_WavePropagationBlock
  file is part of SWE

9.4.1 Detailed Description

This file is part of SWE.

Author

Alexander Breuer (breuera AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Dipl.-Math._Alexander_Breuer)
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Sebastian_Rettenberger,_M.Sc.)
Michael Bader (bader AT in.tum.de, http://www5.in.tum.de/wiki/index.php/Michael_Bader)

9.4.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.gnu.org/licenses/.

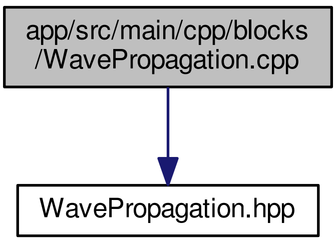
9.4.3 DESCRIPTION

Implementation of SWE_Block that uses solvers in the wave propagation formulation.

9.5 app/src/main/cpp/blocks/WavePropagation.cpp File Reference

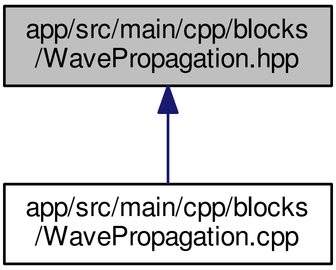
```
#include "WavePropagation.hpp"
```

Include dependency graph for WavePropagation.cpp:



9.6 app/src/main/cpp/blocks/WavePropagation.hpp File Reference

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



Classes

- class solver::WavePropagation< T >
- class solver::WavePropagation< T >

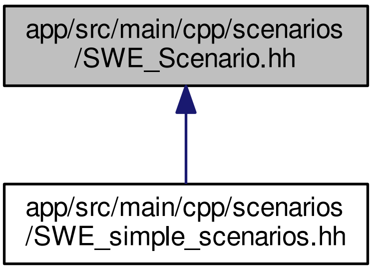
Namespaces

- solver

  This namespace defines code that defines the SWE framework.

9.7 app/src/main/cpp/scenarios/SWE_Scenario.hh File Reference

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



Classes

- class SWE_Scenario
  This class sets the standart layout for all simulateable scenarios.

Typedefs

- typedef enum BoundaryType BoundaryType
- typedef enum BoundaryEdge BoundaryEdge

Enumerations

- enum BoundaryType {
  OUTFLOW, WALL, INFLOW, CONNECT,
  PASSIVE, OUTFLOW, WALL, INFLOW,
  CONNECT, PASSIVE }
- enum BoundaryEdge {
  BND_LEFT, BND_RIGHT, BND_BOTTOM, BND_TOP,
  BND_LEFT, BND_RIGHT, BND_BOTTOM, BND_TOP }

9.7.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel

9.7.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.gnu.org/licenses/`.

9.7.3 DESCRIPTION

TODO

9.7.4 Typedef Documentation

9.7.4.1 typedef enum BoundaryEdge BoundaryEdge

enum type: numbering of the boundary edges

9.7.4.2 typedef enum BoundaryType BoundaryType

enum type: available types of boundary conditions

9.7.5 Enumeration Type Documentation

9.7.5.1 enum BoundaryEdge

enum type: numbering of the boundary edges

Enumerator

BND_LEFT

BND_RIGHT

BND_BOTTOM

BND_TOP

BND_LEFT

BND_RIGHT

BND_BOTTOM

BND_TOP

Definition at line 41 of file SWE_Scenario.hh.

9.7.5.2 enum BoundaryType

enum type: available types of boundary conditions

Enumerator

OUTFLOW

WALL

INFLOW

CONNECT

PASSIVE

OUTFLOW

WALL

INFLOW

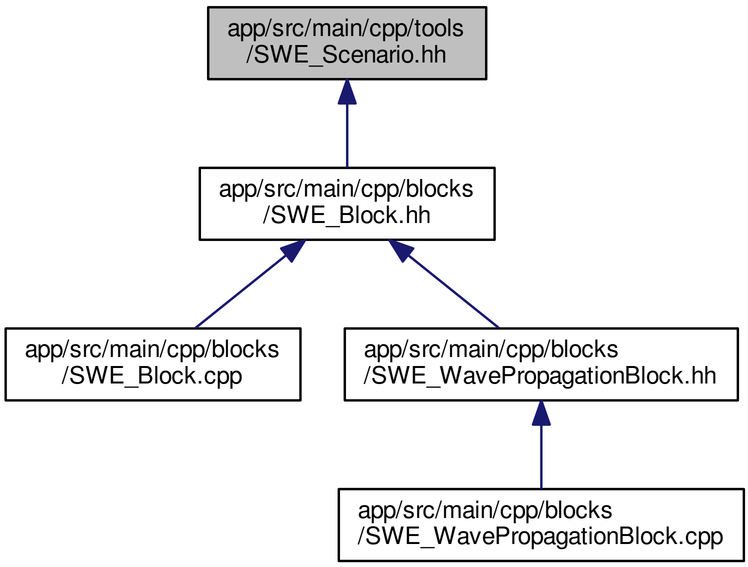
CONNECT

PASSIVE

Definition at line 34 of file SWE_Scenario.hh.

9.8 app/src/main/cpp/tools/SWE_Scenario.hh File Reference

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



Classes

- class SWE_Scenario

    This class sets the standart layout for all simulateable scenarios.

Typedefs

- typedef enum BoundaryType BoundaryType
- typedef enum BoundaryEdge BoundaryEdge

Enumerations

- enum BoundaryType {
  OUTFLOW, WALL, INFLOW, CONNECT,
  PASSIVE, OUTFLOW, WALL, INFLOW,
  CONNECT, PASSIVE }
- enum BoundaryEdge {
  BND_LEFT, BND_RIGHT, BND_BOTTOM, BND_TOP,
  BND_LEFT, BND_RIGHT, BND_BOTTOM, BND_TOP }

9.8.1 Typedef Documentation

9.8.1.1 typedef enum BoundaryEdge BoundaryEdge

enum type: numbering of the boundary edges

9.8.1.2 typedef enum BoundaryType BoundaryType

enum type: available types of boundary conditions

9.8.2 Enumeration Type Documentation

9.8.2.1 enum BoundaryEdge

enum type: numbering of the boundary edges

Enumerator

- ***BND_LEFT***
- ***BND_RIGHT***
- ***BND_BOTTOM***
- ***BND_TOP***
- ***BND_LEFT***
- ***BND_RIGHT***
- ***BND_BOTTOM***
- ***BND_TOP***

Definition at line 41 of file SWE_Scenario.hh.

9.8.2.2 enum BoundaryType

enum type: available types of boundary conditions

Enumerator

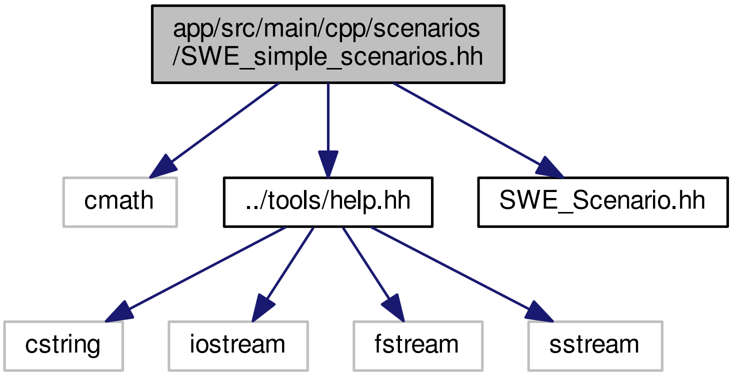
- ***OUTFLOW***
- ***WALL***
- ***INFLOW***
- ***CONNECT***
- ***PASSIVE***
- ***OUTFLOW***
- ***WALL***
- ***INFLOW***
- ***CONNECT***
- ***PASSIVE***

Definition at line 34 of file SWE_Scenario.hh.

9.9 app/src/main/cpp/scenarios/SWE_simple_scenarios.hh File Reference

```
#include <cmath>
#include "../tools/help.hh"
#include "SWE_Scenario.hh"
```

Include dependency graph for SWE_simple_scenarios.hh:



Classes

- class SWE_FlatScenario

  scenario is a test environment that simulates a flat water-surface

9.9.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema, Tobias Schnabel
Sebastian Rettenberger (rettenbs AT in.tum.de, http://www5.in.tum.de/wiki/index.php/↩Sebastian_Rettenberger,_M.Sc.)

9.9.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see http://www.↩gnu.org/licenses/.

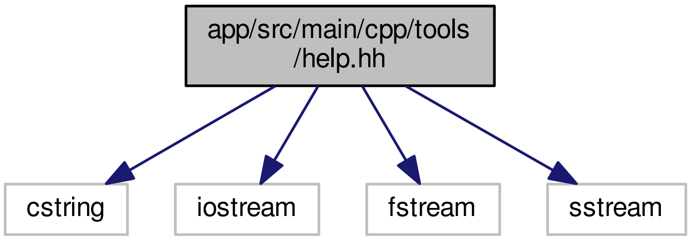
9.9.3 DESCRIPTION

TODO

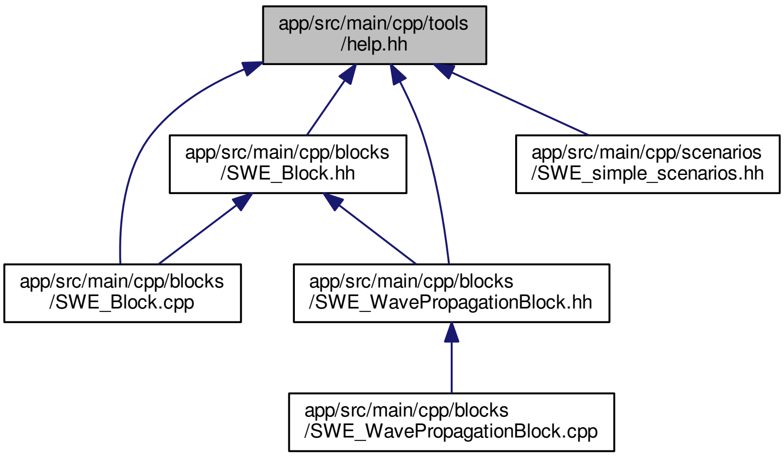
9.10 app/src/main/cpp/tools/help.hh File Reference

```
#include <cstring>
#include <iostream>
#include <fstream>
#include <sstream>
```

Include dependency graph for help.hh:



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



Classes

- class Float1D

  This class is part of SWE; It gives our representation of a one dimensional datatype for update calculations.
- class Float2D

  This class is part of SWE; It gives our representation of atwo dimensional datatype for update calculations.

Functions

- std::string generateFileName (std::string baseName, int timeStep)
- std::string generateFileName (std::string i_baseName, int i_blockPositionX, int i_blockPositionY, std::string i_fileExtension=".nc")
- std::string generateFileName (std::string baseName, int timeStep, int block_X, int block_Y, std::string i_file↵ Extension=".vts")
- std::string generateBaseFileName (std::string &i_baseName, int i_blockPositionX, int i_blockPositionY)
- std::string generateContainerFileName (std::string baseName, int timeStep)

9.10.1 Detailed Description

This file is part of SWE.

Author

Michael Bader, Kaveh Rahnema
Sebastian Rettenberger

9.10.2 LICENSE

SWE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SWE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SWE. If not, see `http://www.↵ gnu.org/licenses/`.

9.10.3 DESCRIPTION

TODO

9.10.4 Function Documentation

9.10.4.1 std::string generateBaseFileName (std::string & *i_baseName*, int *i_blockPositionX*, int *i_blockPositionY*) [inline]

Generates an output file name for a multiple SWE_Block version based on the ordering of the blocks.

Parameters

i_baseName	base name of the output.
i_blockPositionX	position of the SWE_Block in x-direction.
i_blockPositionY	position of the SWE_Block in y-direction.

Returns

the output filename **without** timestep information and file extension

Definition at line 253 of file help.hh.

9.10.4.2 std::string generateContainerFileName (std::string *baseName*, int *timeStep*) [inline]

generate output filename for the ParaView-Container-File (to visualize multiple SWE_Blocks per checkpoint)

Definition at line 265 of file help.hh.

9.10.4.3 std::string generateFileName (std::string *baseName*, int *timeStep*) [inline]

generate output filenames for the single-SWE_Block version (for serial and OpenMP-parallelised versions that use only a single SWE_Block - one output file is generated per checkpoint)

Deprecated

Definition at line 200 of file help.hh.

9.10.4.4 std::string generateFileName (std::string *i_baseName*, int *i_blockPositionX*, int *i_blockPositionY*, std::string *i_fileExtension* = ".nc") [inline]

Generates an output file name for a multiple SWE_Block version based on the ordering of the blocks.

Parameters

i_baseName	base name of the output.
i_blockPositionX	position of the SWE_Block in x-direction.
i_blockPositionY	position of the SWE_Block in y-direction.
i_fileExtension	file extension of the output file.

Returns

Deprecated

Definition at line 218 of file help.hh.

9.10.4.5 std::string generateFileName (std::string *baseName*, int *timeStep*, int *block_X*, int *block_Y*, std::string *i_fileExtension* = ".vts") [inline]

generate output filename for the multiple-SWE_Block version (for serial and parallel (OpenMP and MPI) versions that use multiple SWE_Blocks - for each block, one output file is generated per checkpoint)

Deprecated

Definition at line 236 of file help.hh.

9.11 app/src/main/DocMain.cpp File Reference

Functions

- int main ()

9.11.1 Function Documentation

9.11.1.1 int main ( )

Definition at line 39 of file DocMain.cpp.

9.12 app/src/main/java/Solver/CPPSimulator.java File Reference

Classes

- class Solver.CPPSimulator
  This class interfaces with the SWE-Code(c++);.

Packages

- package Solver
  a The Solver entails all classes dedicated to the mathematical backend of our implementation

9.13 app/src/main/java/Solver/Helper.java File Reference

Classes

- class Solver.Helper
  This class contains a helper method that maps a number of the x domain onto the y domain via linearisation;.

Packages

- package Solver
  a The Solver entails all classes dedicated to the mathematical backend of our implementation

9.14 app/src/main/java/Solver/SimulationRunner.java File Reference

Classes

- class Solver.SimulationRunner
  This class handles the Threading of the Simulation.

Packages

- package Solver
  a The Solver entails all classes dedicated to the mathematical backend of our implementation

Index

~Float1D
    Float1D, 18
~Float2D
    Float2D, 21
~SWE_Block
    SWE_Block, 28
~SWE_Scenario
    SWE_Scenario, 42
~SWE_WavePropagationBlock
    SWE_WavePropagationBlock, 46
~WavePropagation
    solver::WavePropagation, 49

app/src/main/DocMain.cpp, 70
app/src/main/cpp/blocks/SWE_Block.cpp, 55
app/src/main/cpp/blocks/SWE_Block.hh, 56
app/src/main/cpp/blocks/SWE_WavePropagation↩
        Block.cpp, 58
app/src/main/cpp/blocks/SWE_WavePropagation↩
        Block.hh, 59
app/src/main/cpp/blocks/WavePropagation.cpp, 60
app/src/main/cpp/blocks/WavePropagation.hpp, 61
app/src/main/cpp/scenarios/SWE_Scenario.hh, 61
app/src/main/cpp/scenarios/SWE_simple_scenarios.hh,
        66
app/src/main/cpp/tools/SWE_Scenario.hh, 64
app/src/main/cpp/tools/help.hh, 67
app/src/main/java/Solver/CPPSimulator.java, 70
app/src/main/java/Solver/Helper.java, 70
app/src/main/java/Solver/SimulationRunner.java, 70

b
    SWE_Block, 35
bLeft
    solver::WavePropagation, 51
BND_BOTTOM
    scenarios/SWE_Scenario.hh, 63
    tools/SWE_Scenario.hh, 65
BND_LEFT
    scenarios/SWE_Scenario.hh, 63
    tools/SWE_Scenario.hh, 65
BND_RIGHT
    scenarios/SWE_Scenario.hh, 63
    tools/SWE_Scenario.hh, 65
BND_TOP
    scenarios/SWE_Scenario.hh, 63
    tools/SWE_Scenario.hh, 65
bRight
    solver::WavePropagation, 51
boundary
    SWE_Block, 35
BoundaryEdge
    scenarios/SWE_Scenario.hh, 62, 63
    tools/SWE_Scenario.hh, 65
BoundaryType
    scenarios/SWE_Scenario.hh, 62, 63
    tools/SWE_Scenario.hh, 65

CONNECT
    scenarios/SWE_Scenario.hh, 63
    tools/SWE_Scenario.hh, 65
CPPSimulator
    Solver::CPPSimulator, 16
cell_count
    Solver::CPPSimulator, 17
changeActivity
    Solver::SimulationRunner, 23
computeMaxTimestep
    SWE_Block, 28
computeNetUpdates
    solver::WavePropagation, 49
computeNumericalFluxes
    SWE_Block, 28
    SWE_WavePropagationBlock, 47

delete
    Solver::CPPSimulator, 16
determineWetDryState
    solver::WavePropagation, 50
DocMain.cpp
    main, 70
DryDry
    solver::WavePropagation, 49
dryTol
    solver::WavePropagation, 51
DryWetInundation
    solver::WavePropagation, 49
DryWetWall
    solver::WavePropagation, 49
DryWetWallInundation
    solver::WavePropagation, 49
dx
    SWE_Block, 35
dy
    SWE_Block, 35

elemVector
    Float1D, 19
    Float2D, 21
endSimulation

SWE_FlatScenario, 40
SWE_Scenario, 42

finalize
Solver::CPPSimulator, 16
Float1D, 18
~Float1D, 18
elemVector, 19
Float1D, 18
getSize, 19
operator[], 19
Float2D, 19
~Float2D, 21
elemVector, 21
Float2D, 20
getColProxy, 21
getCols, 21
getRowProxy, 21
getRows, 21
operator[], 21

g
SWE_Block, 35
solver::WavePropagation, 51
generateBaseFileName
help.hh, 68
generateContainerFileName
help.hh, 69
generateFileName
help.hh, 69
getBathymetry
SWE_Block, 29
SWE_FlatScenario, 40
SWE_Scenario, 42
Solver::CPPSimulator, 16
getBoundaryPos
SWE_FlatScenario, 40
SWE_Scenario, 42
getBoundaryType
SWE_FlatScenario, 40
SWE_Scenario, 43
getColProxy
Float2D, 21
getCols
Float2D, 21
getDischarge_hu
SWE_Block, 29
getDischarge_hv
SWE_Block, 29
getHeight
Solver::CPPSimulator, 16
getMaxTimestep
SWE_Block, 29
getNx
SWE_Block, 29
getNy
SWE_Block, 29
getRowProxy
Float2D, 21
getRows
Float2D, 21
getSize
Float1D, 19
getVeloc_u
SWE_Scenario, 43
getVeloc_v
SWE_Scenario, 43
getWaterHeight
SWE_Block, 29
SWE_FlatScenario, 40
SWE_Scenario, 43
grabGhostLayer
SWE_Block, 30

h
SWE_Block, 35
SWE_Block1D, 38
hLeft
solver::WavePropagation, 52
hRight
solver::WavePropagation, 52
help.hh
generateBaseFileName, 68
generateContainerFileName, 69
generateFileName, 69
hu
SWE_Block, 35
SWE_Block1D, 38
huLeft
solver::WavePropagation, 52
huRight
solver::WavePropagation, 52
hv
SWE_Block, 36
SWE_Block1D, 38

INFLOW
scenarios/SWE_Scenario.hh, 63
tools/SWE_Scenario.hh, 65
initScenario
SWE_Block, 30
isStarted
Solver::SimulationRunner, 23

linear_map
Solver::Helper, 22

main
DocMain.cpp, 70
maxTimestep
SWE_Block, 36

neighbour
SWE_Block, 36
nx
SWE_Block, 36
ny
SWE_Block, 36

OUTFLOW
scenarios/SWE_Scenario.hh, 63
tools/SWE_Scenario.hh, 65
offsetX
SWE_Block, 36
offsetY
SWE_Block, 36
operator[]
Float1D, 19
Float2D, 21

PASSIVE
scenarios/SWE_Scenario.hh, 63
tools/SWE_Scenario.hh, 65
placeCircle
Solver::CPPSimulator, 17

registerCopyLayer
SWE_Block, 30
reset
Solver::CPPSimulator, 17
resetWaves
Solver::CPPSimulator, 17

SWE_Block, 24
∼SWE_Block, 28
b, 35
boundary, 35
computeMaxTimestep, 28
computeNumericalFluxes, 28
dx, 35
dy, 35
g, 35
getBathymetry, 29
getDischarge_hu, 29
getDischarge_hv, 29
getMaxTimestep, 29
getNx, 29
getNy, 29
getWaterHeight, 29
grabGhostLayer, 30
h, 35
hu, 35
hv, 36
initScenario, 30
maxTimestep, 36
neighbour, 36
nx, 36
ny, 36
offsetX, 36
offsetY, 36
registerCopyLayer, 30
SWE_Block, 28
setBathymetry, 31
setBathymetryXY, 31
setBoundaryBathymetry, 31
setBoundaryConditions, 31
setBoundaryType, 31
setDischarge, 32
setGhostLayer, 32
setHuXY, 32
setHvXY, 32
setWaterHeight, 32
setWaterHeightXY, 32
simulate, 32
simulateTimestep, 33
synchAfterWrite, 33
synchBathymetryAfterWrite, 33
synchBathymetryBeforeRead, 33
synchBeforeRead, 33
synchCopyLayerBeforeRead, 34
synchDischargeAfterWrite, 34
synchDischargeBeforeRead, 34
synchGhostLayerAfterWrite, 34
synchWaterHeightAfterWrite, 34
synchWaterHeightBeforeRead, 34
updateUnknowns, 34
SWE_Block1D, 37
h, 38
hu, 38
hv, 38
SWE_Block1D, 38
SWE_FlatScenario, 39
endSimulation, 40
getBathymetry, 40
getBoundaryPos, 40
getBoundaryType, 40
getWaterHeight, 40
SWE_Scenario, 41
∼SWE_Scenario, 42
endSimulation, 42
getBathymetry, 42
getBoundaryPos, 42
getBoundaryType, 43
getVeloc_u, 43
getVeloc_v, 43
getWaterHeight, 43
waterHeightAtRest, 43, 44
SWE_WavePropagationBlock, 44
∼SWE_WavePropagationBlock, 46
computeNumericalFluxes, 47
SWE_WavePropagationBlock, 46
updateUnknowns, 47
updateUnknownsRow, 47
scenarios/SWE_Scenario.hh
BND_BOTTOM, 63
BND_LEFT, 63
BND_RIGHT, 63
BND_TOP, 63
BoundaryEdge, 62, 63
BoundaryType, 62, 63
CONNECT, 63
INFLOW, 63
OUTFLOW, 63
PASSIVE, 63
WALL, 63
setBathymetry

SWE_Block, 31
setBathymetryXY
SWE_Block, 31
setBoundaryBathymetry
SWE_Block, 31
setBoundaryConditions
SWE_Block, 31
setBoundaryType
SWE_Block, 31
Solver::CPPSimulator, 17
setDischarge
SWE_Block, 32
setDryTolerance
solver::WavePropagation, 50
setGhostLayer
SWE_Block, 32
setHuXY
SWE_Block, 32
setHvXY
SWE_Block, 32
setWaterHeight
SWE_Block, 32
setWaterHeightXY
SWE_Block, 32
setWave
Solver::CPPSimulator, 17
sim
Solver::CPPSimulator, 17
simulate
SWE_Block, 32
simulateTimestep
SWE_Block, 33
simulatetimestep
Solver::CPPSimulator, 17
SimulationRunner
Solver::SimulationRunner, 23
Solver, 13
Solver.CPPSimulator, 15
Solver.Helper, 22
Solver.SimulationRunner, 22
Solver::CPPSimulator
CPPSimulator, 16
cell_count, 17
delete, 16
finalize, 16
getBathymetry, 16
getHeight, 16
placeCircle, 17
reset, 17
resetWaves, 17
setBoundaryType, 17
setWave, 17
sim, 17
simulatetimestep, 17
waterlevel, 18
Solver::Helper
linear_map, 22
Solver::SimulationRunner
changeActivity, 23
isStarted, 23
SimulationRunner, 23
start, 23
stop, 23
solver::WavePropagation
∼WavePropagation, 49
bLeft, 51
bRight, 51
computeNetUpdates, 49
determineWetDryState, 50
DryDry, 49
dryTol, 51
DryWetInundation, 49
DryWetWall, 49
DryWetWallInundation, 49
g, 51
hLeft, 52
hRight, 52
huLeft, 52
huRight, 52
setDryTolerance, 50
storeParameters, 50, 51
uLeft, 52
uRight, 52
WavePropagation, 49
WetDryInundation, 49
WetDryState, 49
wetDryState, 52
WetDryWall, 49
WetDryWallInundation, 49
WetWet, 49
zeroTol, 52
solver::WavePropagation< T >, 47
start
Solver::SimulationRunner, 23
stop
Solver::SimulationRunner, 23
storeParameters
solver::WavePropagation, 50, 51
synchAfterWrite
SWE_Block, 33
synchBathymetryAfterWrite
SWE_Block, 33
synchBathymetryBeforeRead
SWE_Block, 33
synchBeforeRead
SWE_Block, 33
synchCopyLayerBeforeRead
SWE_Block, 34
synchDischargeAfterWrite
SWE_Block, 34
synchDischargeBeforeRead
SWE_Block, 34
synchGhostLayerAfterWrite
SWE_Block, 34
synchWaterHeightAfterWrite
SWE_Block, 34

synchWaterHeightBeforeRead
SWE_Block, 34

tools/SWE_Scenario.hh
BND_BOTTOM, 65
BND_LEFT, 65
BND_RIGHT, 65
BND_TOP, 65
BoundaryEdge, 65
BoundaryType, 65
CONNECT, 65
INFLOW, 65
OUTFLOW, 65
PASSIVE, 65
WALL, 65

uLeft
solver::WavePropagation, 52
uRight
solver::WavePropagation, 52
updateUnknowns
SWE_Block, 34
SWE_WavePropagationBlock, 47
updateUnknownsRow
SWE_WavePropagationBlock, 47

WALL
scenarios/SWE_Scenario.hh, 63
tools/SWE_Scenario.hh, 65
waterHeightAtRest
SWE_Scenario, 43, 44
waterlevel
Solver::CPPSimulator, 18
WavePropagation, 53
solver::WavePropagation, 49
WetDryInundation
solver::WavePropagation, 49
WetDryState
solver::WavePropagation, 49
wetDryState
solver::WavePropagation, 52
WetDryWall
solver::WavePropagation, 49
WetDryWallInundation
solver::WavePropagation, 49
WetWet
solver::WavePropagation, 49

zeroTol
solver::WavePropagation, 52