

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 Interface	13
7.1.1	Detailed Description	13
7.2	solver Namespace Reference	13
7.2.1	Detailed Description	14
7.3	Package Solver	14
7.3.1	Detailed Description	14
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	Interface.MainActivity Class Reference	22
8.5.1	Detailed Description	23
8.5.2	Member Function Documentation	23
8.5.2.1	getSimulationRunner()	23

8.5.2.2	getWaveHeightValue()	24
8.5.2.3	getWaveView()	24
8.5.2.4	onCheckedSwitch(CompoundButton buttonView, boolean isChecked)	24
8.5.2.5	onCreate(Bundle savedInstanceState)	24
8.5.2.6	onCreateOptionsMenu(Menu menu)	24
8.5.2.7	onOptionsItemSelected(MenuItem item)	24
8.5.2.8	WaveHeightSeekBarChanged()	25
8.6	Solver.SimulationRunner Class Reference	25
8.6.1	Detailed Description	25
8.6.2	Constructor & Destructor Documentation	25
8.6.2.1	SimulationRunner(MainActivity currentActivity)	25
8.6.3	Member Function Documentation	25
8.6.3.1	changeActivity(MainActivity a)	25
8.6.3.2	isStarted()	26
8.6.3.3	start()	26
8.6.3.4	stop()	26
8.7	SWE_Block Class Reference	26
8.7.1	Detailed Description	29
8.7.2	Constructor & Destructor Documentation	30
8.7.2.1	SWE_Block(int l_nx, int l_ny, float l_dx, float l_dy)	30
8.7.2.2	~SWE_Block()	31
8.7.3	Member Function Documentation	31
8.7.3.1	computeMaxTimestep(const float i_dryTol=0.1, const float i_cflNumber=0.4) . . .	31
8.7.3.2	computeNumericalFluxes()=0	31
8.7.3.3	getBathymetry()	31
8.7.3.4	getDischarge_hu()	31
8.7.3.5	getDischarge_hv()	32
8.7.3.6	getMaxTimestep()	32
8.7.3.7	getNx()	32
8.7.3.8	getNy()	32

8.7.3.9	<code>getWaterHeight()</code>	32
8.7.3.10	<code>grabGhostLayer(BoundaryEdge edge)</code>	32
8.7.3.11	<code>initScenario(float _offsetX, float _offsetY, SWE_Scenario &i_scenario, const bool i_multipleBlocks=false)</code>	33
8.7.3.12	<code>registerCopyLayer(BoundaryEdge edge)</code>	33
8.7.3.13	<code>setBathymetry(float _b)</code>	33
8.7.3.14	<code>setBathymetry(float(*_b)(float, float))</code>	34
8.7.3.15	<code>setBathymetryXY(int x, int y, float h_set)</code>	34
8.7.3.16	<code>setBoundaryBathymetry()</code>	34
8.7.3.17	<code>setBoundaryConditions()</code>	34
8.7.3.18	<code>setBoundaryType(BoundaryEdge edge, BoundaryType boundtype, const SWE↔_Block1D *inflow=NULL)</code>	34
8.7.3.19	<code>setDischarge(float(*_u)(float, float), float(*_v)(float, float))</code>	35
8.7.3.20	<code>setGhostLayer()</code>	35
8.7.3.21	<code>setHuXY(int x, int y, float h_set)</code>	35
8.7.3.22	<code>setHvXY(int x, int y, float h_set)</code>	35
8.7.3.23	<code>setWaterHeight(float(*_h)(float, float))</code>	35
8.7.3.24	<code>setWaterHeightXY(int x, int y, float h_set)</code>	35
8.7.3.25	<code>simulate(float tStart, float tEnd)</code>	35
8.7.3.26	<code>simulateTimestep(float dt)</code>	36
8.7.3.27	<code>synchAfterWrite()</code>	36
8.7.3.28	<code>synchBathymetryAfterWrite()</code>	36
8.7.3.29	<code>synchBathymetryBeforeRead()</code>	36
8.7.3.30	<code>synchBeforeRead()</code>	37
8.7.3.31	<code>synchCopyLayerBeforeRead()</code>	37
8.7.3.32	<code>synchDischargeAfterWrite()</code>	37
8.7.3.33	<code>synchDischargeBeforeRead()</code>	37
8.7.3.34	<code>synchGhostLayerAfterWrite()</code>	37
8.7.3.35	<code>synchWaterHeightAfterWrite()</code>	37
8.7.3.36	<code>synchWaterHeightBeforeRead()</code>	37
8.7.3.37	<code>updateUnknowns(float dt)=0</code>	37

8.7.4	Member Data Documentation	38
8.7.4.1	b	38
8.7.4.2	boundary	38
8.7.4.3	dx	38
8.7.4.4	dy	38
8.7.4.5	g	38
8.7.4.6	h	38
8.7.4.7	hu	39
8.7.4.8	hv	39
8.7.4.9	maxTimestep	39
8.7.4.10	neighbour	39
8.7.4.11	nx	39
8.7.4.12	ny	39
8.7.4.13	offsetX	39
8.7.4.14	offsetY	40
8.8	SWE_Block1D Class Reference	40
8.8.1	Detailed Description	41
8.8.2	Constructor & Destructor Documentation	41
8.8.2.1	SWE_Block1D(const Float1D &_h, const Float1D &_hu, const Float1D &_hv)	41
8.8.2.2	SWE_Block1D(float *_h, float *_hu, float *_hv, int _size, int _stride=1)	41
8.8.3	Member Data Documentation	41
8.8.3.1	h	41
8.8.3.2	hu	41
8.8.3.3	hv	41
8.9	SWE_FlatScenario Class Reference	42
8.9.1	Detailed Description	42
8.9.2	Member Function Documentation	43
8.9.2.1	endSimulation()	43
8.9.2.2	getBathymetry(float x, float y)	43
8.9.2.3	getBoundaryPos(BoundaryEdge i_edge)	43

8.9.2.4	getBoundaryType(BoundaryEdge edge)	43
8.9.2.5	getWaterHeight(float x, float y)	43
8.10	SWE_Scenario Class Reference	44
8.10.1	Detailed Description	44
8.10.2	Constructor & Destructor Documentation	45
8.10.2.1	~SWE_Scenario()	45
8.10.2.2	~SWE_Scenario()	45
8.10.3	Member Function Documentation	45
8.10.3.1	endSimulation()	45
8.10.3.2	endSimulation()	45
8.10.3.3	getBathymetry(float x, float y)	45
8.10.3.4	getBathymetry(float x, float y)	45
8.10.3.5	getBoundaryPos(BoundaryEdge edge)	45
8.10.3.6	getBoundaryPos(BoundaryEdge edge)	46
8.10.3.7	getBoundaryType(BoundaryEdge edge)	46
8.10.3.8	getBoundaryType(BoundaryEdge edge)	46
8.10.3.9	getVeloc_u(float x, float y)	46
8.10.3.10	getVeloc_u(float x, float y)	46
8.10.3.11	getVeloc_v(float x, float y)	46
8.10.3.12	getVeloc_v(float x, float y)	46
8.10.3.13	getWaterHeight(float x, float y)	46
8.10.3.14	getWaterHeight(float x, float y)	47
8.10.3.15	waterHeightAtRest()	47
8.10.3.16	waterHeightAtRest()	47
8.11	SWE_WavePropagationBlock Class Reference	47
8.11.1	Detailed Description	48
8.11.2	Constructor & Destructor Documentation	49
8.11.2.1	SWE_WavePropagationBlock(int l_nx, int l_ny, float l_dx, float l_dy)	49
8.11.2.2	~SWE_WavePropagationBlock()	49
8.11.3	Member Function Documentation	50

8.11.3.1	computeNumericalFluxes()	50
8.11.3.2	updateUnknowns(float dt)	50
8.11.3.3	updateUnknownsRow(float dt, int i)	50
8.12	WavePropagation Class Reference	50
8.12.1	Detailed Description	50
8.13	solver::WavePropagation< T > Class Template Reference	50
8.13.1	Detailed Description	52
8.13.2	Member Enumeration Documentation	52
8.13.2.1	WetDryState	52
8.13.3	Constructor & Destructor Documentation	52
8.13.3.1	WavePropagation(T i_dryTolerance, T i_gravity, T i_zeroTolerance)	52
8.13.3.2	~WavePropagation()	53
8.13.4	Member Function Documentation	53
8.13.4.1	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_maxWaveSpeed)=0	53
8.13.4.2	determineWetDryState()=0	53
8.13.4.3	setDryTolerance(const T i_dryTolerance)	53
8.13.4.4	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)	54
8.13.4.5	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)	54
8.13.5	Member Data Documentation	54
8.13.5.1	bLeft	54
8.13.5.2	bRight	54
8.13.5.3	dryTol	55
8.13.5.4	g	55
8.13.5.5	hLeft	55
8.13.5.6	hRight	55
8.13.5.7	huLeft	55
8.13.5.8	huRight	55

8.13.5.9	<code>uLeft</code>	55
8.13.5.10	<code>uRight</code>	55
8.13.5.11	<code>wetDryState</code>	56
8.13.5.12	<code>zeroTol</code>	56
8.14	Interface.WaveView Class Reference	56
8.14.1	Detailed Description	57
8.14.2	Constructor & Destructor Documentation	57
8.14.2.1	<code>WaveView(Context context, AttributeSet attrs)</code>	57
8.14.3	Member Function Documentation	57
8.14.3.1	<code>onDraw(Canvas canvas)</code>	57
8.14.3.2	<code>onSizeChanged(int w, int h, int old_w, int old_h)</code>	58
8.15	Interface.WaveViewTouchListener Class Reference	58
8.15.1	Detailed Description	59
8.15.2	Member Function Documentation	59
8.15.2.1	<code>onDoubleTap(MotionEvent motionEvent)</code>	59
8.15.2.2	<code>onDoubleTapEvent(MotionEvent motionEvent)</code>	59
8.15.2.3	<code>onDown(MotionEvent motionEvent)</code>	60
8.15.2.4	<code>onFling(MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1)</code>	60
8.15.2.5	<code>onLongPress(MotionEvent motionEvent)</code>	60
8.15.2.6	<code>onScroll(MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1)</code>	60
8.15.2.7	<code>onShowPress(MotionEvent motionEvent)</code>	60
8.15.2.8	<code>onSingleTapConfirmed(MotionEvent motionEvent)</code>	60
8.15.2.9	<code>onSingleTapUp(MotionEvent motionEvent)</code>	60
8.15.2.10	<code>onTouch(View view, MotionEvent motionEvent)</code>	61
8.15.3	Member Data Documentation	61
8.15.3.1	<code>drawingmode</code>	61
8.15.3.2	<code>MODE_DRAW</code>	61
8.15.3.3	<code>MODE_ERASE</code>	61
8.15.3.4	<code>MODE_SIMULATE</code>	61

9	File Documentation	63
9.1	app/src/main/cpp/blocks/SWE_Block.cpp File Reference	63
9.1.1	Detailed Description	63
9.1.2	LICENSE	64
9.1.3	DESCRIPTION	64
9.2	app/src/main/cpp/blocks/SWE_Block.hh File Reference	64
9.2.1	Detailed Description	65
9.2.2	LICENSE	65
9.2.3	DESCRIPTION	66
9.3	app/src/main/cpp/blocks/SWE_WavePropagationBlock.cpp File Reference	66
9.3.1	Detailed Description	66
9.3.2	LICENSE	66
9.3.3	DESCRIPTION	67
9.4	app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh File Reference	67
9.4.1	Detailed Description	68
9.4.2	LICENSE	68
9.4.3	DESCRIPTION	68
9.5	app/src/main/cpp/blocks/WavePropagation.cpp File Reference	68
9.6	app/src/main/cpp/blocks/WavePropagation.hpp File Reference	69
9.7	app/src/main/cpp/scenarios/SWE_Scenario.hh File Reference	69
9.7.1	Detailed Description	70
9.7.2	LICENSE	70
9.7.3	DESCRIPTION	70
9.7.4	Typedef Documentation	70
9.7.4.1	BoundaryEdge	70
9.7.4.2	BoundaryType	71
9.7.5	Enumeration Type Documentation	71
9.7.5.1	BoundaryEdge	71
9.7.5.2	BoundaryType	71
9.8	app/src/main/cpp/tools/SWE_Scenario.hh File Reference	72

9.8.1	Typedef Documentation	73
9.8.1.1	BoundaryEdge	73
9.8.1.2	BoundaryType	73
9.8.2	Enumeration Type Documentation	73
9.8.2.1	BoundaryEdge	73
9.8.2.2	BoundaryType	73
9.9	app/src/main/cpp/scenarios/SWE_simple_scenarios.hh File Reference	74
9.9.1	Detailed Description	74
9.9.2	LICENSE	74
9.9.3	DESCRIPTION	75
9.10	app/src/main/cpp/tools/help.hh File Reference	75
9.10.1	Detailed Description	76
9.10.2	LICENSE	76
9.10.3	DESCRIPTION	76
9.10.4	Function Documentation	76
9.10.4.1	generateBaseFileName(std::string &i_baseName, int i_blockPositionX, int i_blockPositionY)	76
9.10.4.2	generateContainerFileName(std::string baseName, int timeStep)	77
9.10.4.3	generateFileName(std::string baseName, int timeStep)	77
9.10.4.4	generateFileName(std::string i_baseName, int i_blockPositionX, int i_blockPositionY, std::string i_fileExtension="" <code>.nc</code> "")	77
9.10.4.5	generateFileName(std::string baseName, int timeStep, int block_X, int block_Y, std::string i_fileExtension="" <code>.vts</code> "")	77
9.11	app/src/main/DocMain.cpp File Reference	78
9.11.1	Function Documentation	78
9.11.1.1	main()	78
9.12	app/src/main/java/Interface/MainActivity.java File Reference	78
9.13	app/src/main/java/Interface/WaveView.java File Reference	78
9.14	app/src/main/java/Interface/WaveViewTouchListener.java File Reference	78
9.15	app/src/main/java/Solver/CPPSimulator.java File Reference	79
9.16	app/src/main/java/Solver/Helper.java File Reference	79
9.17	app/src/main/java/Solver/SimulationRunner.java File Reference	79

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-↔ Height-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_fileExtension=".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:

Interface	The Interface entails all classes dedicated to the visual representation and user interaction . . .	13
solver	This namespace defines code that defines the SWE framework	13
Solver	The Solver entails all classes dedicated to the mathematical backend of our implementation . .	14

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
OnDoubleTapListener	
Interface.WaveViewTouchListener	58
OnGestureListener	
Interface.WaveViewTouchListener	58
OnTouchListener	
Interface.WaveViewTouchListener	58
Solver.SimulationRunner	25
SWE_Block	26
SWE_WavePropagationBlock	47
SWE_Block1D	40
SWE_Scenario	44
SWE_FlatScenario	42
WavePropagation	50
solver::WavePropagation< T >	50
AppCompatActivity	
Interface.MainActivity	22
View	
Interface.WaveView	56

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
Interface.MainActivity	This class handles the main-Activities	22
Solver.SimulationRunner	This class handles the Threading of the Simulation	25
SWE_Block	This file is part of SWE	26
SWE_Block1D	This file is part SWE	40
SWE_FlatScenario	Scenario is a test environment that simulates a flat water-surface	42
SWE_Scenario	This class sets the standart layout for all simulateable scenarios	44
SWE_WavePropagationBlock	File is part of SWE	47
WavePropagation	Abstract wave propagation solver for the Shallow Water Equations; T should be double or float;	50
solver::WavePropagation< T >	50
Interface.WaveView	This class is responsible for the Wave visualization	56
Interface.WaveViewTouchListener	Class implemets the recation of the displayed content to the users touch-inputs	58

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

app/src/main/DocMain.cpp	78
app/src/main/cpp/blocks/SWE_Block.cpp	63
app/src/main/cpp/blocks/SWE_Block.hh	64
app/src/main/cpp/blocks/SWE_WavePropagationBlock.cpp	66
app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh	67
app/src/main/cpp/blocks/WavePropagation.cpp	68
app/src/main/cpp/blocks/WavePropagation.hpp	69
app/src/main/cpp/scenarios/SWE_Scenario.hh	69
app/src/main/cpp/scenarios/SWE_simple_scenarios.hh	74
app/src/main/cpp/tools/help.hh	75
app/src/main/cpp/tools/SWE_Scenario.hh	72
app/src/main/java/Interface/MainActivity.java	78
app/src/main/java/Interface/WaveView.java	78
app/src/main/java/Interface/WaveViewTouchListener.java	78
app/src/main/java/Solver/CPPSimulator.java	79
app/src/main/java/Solver/Helper.java	79
app/src/main/java/Solver/SimulationRunner.java	79

Chapter 7

Namespace Documentation

7.1 Package Interface

a The [Interface](#) entails all classes dedicated to the visual representation and user interaction

Classes

- class [MainActivity](#)
This class handles the main-Activities.
- class [WaveView](#)
This class is responsible for the Wave visualization.
- class [WaveViewTouchListener](#)
class implemets the recation of the displayed content to the users touch-inputs

7.1.1 Detailed Description

a The [Interface](#) entails all classes dedicated to the visual representation and user interaction

7.2 solver Namespace Reference

This namespace defines code that defines the SWE framework.

Classes

- class [WavePropagation](#)

7.2.1 Detailed Description

This namespace defines code that defines the SWE framework.

[WavePropagation.hpp](#)

Abstract wave propagation solver for the Shallow Water Equations.

Created on: Jan 03, 2012 Last Update: Jan 05, 2012

Author: Alexander Breuer Homepage: http://www5.in.tum.de/wiki/index.php/Dipl.-Math._Alexander_Breuer E-Mail: breuera AT in.tum.de

7.3 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.

7.3.1 Detailed Description

a The [Solver](#) entails all classes dedicated to the mathematical backend of our implementation

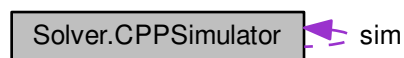
Chapter 8

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 library

< 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/destructor, 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 a two 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

<code>_cols</code>	number of columns (i.e., elements in horizontal direction)
<code>_rows</code>	number 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

<code>_cols</code>	number of columns (i.e., elements in horizontal direction)
<code>_rows</code>	number of rows (i.e., elements in vertical directions)
<code>_elem</code>	pointer to a suitably allocated region of memory to be used for the 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

<code>_cols</code>	number of columns (i.e., elements in horizontal direction)
<code>_rows</code>	number of rows (i.e., elements in vertical directions)
<code>_elem</code>	pointer to a suitably allocated region of memory to be used for the 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.

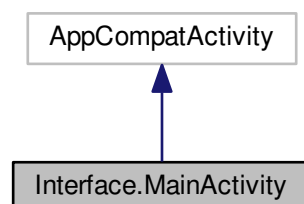
The documentation for this class was generated from the following file:

- app/src/main/java/Solver/[Helper.java](#)

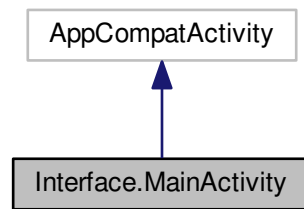
8.5 Interface.MainActivity Class Reference

This class handles the main-Activities.

Inheritance diagram for Interface.MainActivity:



Collaboration diagram for Interface.MainActivity:



Public Member Functions

- boolean [onCreateOptionsMenu](#) (Menu menu)
- boolean [onOptionsItemSelected](#) (MenuItem item)
gets called if a menu item is selected
- void [WaveHeightSeekBarChanged](#) ()
gets called when the user uses the waveheight seekbar
- float [getWaveHeightValue](#) ()
- void [onCheckedSwitch](#) (CompoundButton buttonView, boolean isChecked)
gets called when the user uses the wall boundary switch
- [WaveView](#) [getWaveView](#) ()

Static Public Member Functions

- static [SimulationRunner](#) [getSimulationRunner](#) ()

Protected Member Functions

- void [onCreate](#) (Bundle savedInstanceState)

8.5.1 Detailed Description

This class handles the main-Activities.

Definition at line 25 of file MainActivity.java.

8.5.2 Member Function Documentation

8.5.2.1 static [SimulationRunner](#) [Interface.MainActivity.getSimulationRunner](#) () `[inline], [static]`

Definition at line 178 of file MainActivity.java.

8.5.2.2 float Interface.MainActivity.getWaveHeightValue () [inline]

< maps the seekbarvalue(0,100) to the waveheight(5,15)

Definition at line 165 of file MainActivity.java.

8.5.2.3 WaveView Interface.MainActivity.getWaveView () [inline]

Definition at line 174 of file MainActivity.java.

8.5.2.4 void Interface.MainActivity.onCheckedSwitch (CompoundButton *buttonView*, boolean *isChecked*) [inline]

gets called when the user uses the wall boundary switch

Definition at line 170 of file MainActivity.java.

8.5.2.5 void Interface.MainActivity.onCreate (Bundle *savedInstanceState*) [inline],[protected]

< Defines the onClick Listener for the switch

< Defines Touchlistener of the Waveview

< set Listener of the waveheight seekbar

< add the toolbar to the main activity

< create a new simulation runner or set the current activity in the existing one

Definition at line 48 of file MainActivity.java.

8.5.2.6 boolean Interface.MainActivity.onCreateOptionsMenu (Menu *menu*) [inline]

< sets the custom menu to the actionbar

Definition at line 95 of file MainActivity.java.

8.5.2.7 boolean Interface.MainActivity.onOptionsItemSelected (MenuItem *item*) [inline]

gets called if a menu item is selected

< reset is selected

< reloads the Simulation;

< resetboundary Switch

< reset wave is selected

< brush mode is selected

if its already in the draw mode, disable draw mode

if not, disable erase mode and enable drawmode

< erase mode is selected

if its already in the erase mode, disable erase mode

if not, disable draw mode and enable erase mode

show snackbar which explains current action

Definition at line 103 of file MainActivity.java.

8.5.2.8 void Interface.MainActivity.WaveHeightSeekBarChanged () [inline]

gets called when the user uses the waveheight seekbar

< inform user about current waveheight

Definition at line 158 of file MainActivity.java.

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

- [app/src/main/java/Interface/MainActivity.java](#)

8.6 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](#) ()

8.6.1 Detailed Description

This class handles the Threading of the Simulation.

Definition at line 6 of file SimulationRunner.java.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 Solver.SimulationRunner.SimulationRunner ([MainActivity](#) currentActivity) [inline]

Definition at line 11 of file SimulationRunner.java.

8.6.3 Member Function Documentation

8.6.3.1 void Solver.SimulationRunner.changeActivity ([MainActivity](#) a) [inline]

Definition at line 47 of file SimulationRunner.java.

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

Definition at line 55 of file SimulationRunner.java.

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

starts the simulation

< get the current time

< simulate a single timestep

< redraw view

< calculate the duration of the timestep simulation

< sleep if took less time then sleeptime

< starts the created thread

Definition at line 17 of file SimulationRunner.java.

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

Definition at line 51 of file SimulationRunner.java.

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

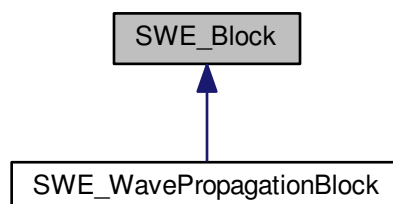
- [app/src/main/java/Solver/SimulationRunner.java](#)

8.7 SWE_Block Class Reference

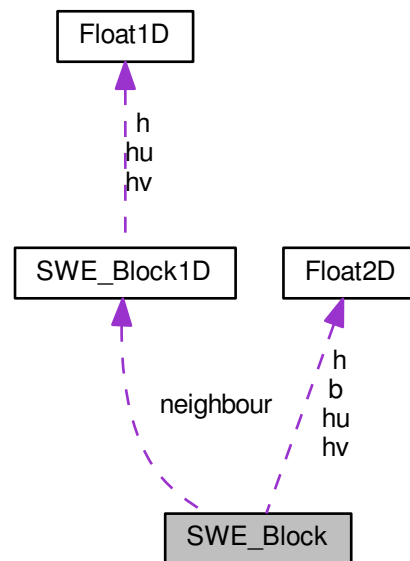
This file is part of SWE.

```
#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 additional 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 ($g = 9.81 \text{ m/s}^2$):*

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

8.7.1 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 $[0, \dots, nx+1] \times [0, \dots, ny+1]$. The computational domain is indexed with $[1, \dots, nx] \times [1, \dots, ny]$.

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 an update 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 [synch↔BeforeRead\(\)](#), 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](#).

8.7.2 Constructor & Destructor Documentation**8.7.2.1 [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](#).

8.7.2.2 SWE_Block::~SWE_Block () [protected],[virtual]

Destructor: de-allocate all variables

Definition at line 70 of file SWE_Block.cpp.

8.7.3 Member Function Documentation

8.7.3.1 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>i_dryTol</i>	dry tolerance (dry cells do not affect the time step).
<i>i_cflNumber</i>	CFL number of the used method.

Definition at line 490 of file SWE_Block.cpp.

8.7.3.2 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](#).

8.7.3.3 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.

8.7.3.4 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.

8.7.3.5 `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.

8.7.3.6 `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.

8.7.3.7 `int SWE_Block::getNx ()` `[inline]`

returns [nx](#), i.e. the grid size in x-direction

Definition at line 199 of file SWE_Block.hh.

8.7.3.8 `int SWE_Block::getNy ()` `[inline]`

returns [ny](#), i.e. the grid size in y-direction



Definition at line 201 of file SWE_Block.hh.

8.7.3.9 `const Float2D & SWE_Block::getWaterHeight ()`

provides read access to the water height array

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

Parameters

	array holding b-values in sequence return reference to water height unknown h
	
<i>b</i>	

Definition at line 235 of file SWE_Block.cpp.

8.7.3.10 `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

<i>specified</i>	edge
------------------	------

Returns

a [SWE_Block1D](#) object that contains row variables h, hu, and hv

Definition at line 398 of file SWE_Block.cpp.

8.7.3.11 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 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>i_scenario</i>	scenario, which is used during the setup.
<i>i_multipleBlocks</i>	are the multiple SWE_blocks?

Definition at line 90 of file SWE_Block.cpp.

8.7.3.12 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.

8.7.3.13 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.

8.7.3.14 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.

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

Definition at line 131 of file SWE_Block.cpp.

8.7.3.16 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.

8.7.3.17 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.

8.7.3.18 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>i_edge</i>	location of the edge relative to the SWE_block.
<i>i_boundaryType</i>	type of the boundary condition.
<i>i_inflow</i>	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.

8.7.3.19 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.

8.7.3.20 void SWE_Block::setGhostLayer ()

set values in ghost layers

set the values of all ghost cells depending on the specified 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.

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

Definition at line 135 of file SWE_Block.cpp.

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

Definition at line 139 of file SWE_Block.cpp.

8.7.3.23 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.

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

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

Definition at line 127 of file SWE_Block.cpp.

8.7.3.25 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

<i>tStart</i>	time where the simulation is started
<i>tEnd</i>	time of the next checkpoint

Returns

actual end time reached

Definition at line 293 of file SWE_Block.cpp.

8.7.3.26 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

<i>dt</i>	time step width of the update
-----------	-------------------------------

Definition at line 276 of file SWE_Block.cpp.

8.7.3.27 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.

8.7.3.28 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.

8.7.3.29 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.

8.7.3.30 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.

8.7.3.31 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.

8.7.3.32 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.

8.7.3.33 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.

8.7.3.34 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.

8.7.3.35 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.

8.7.3.36 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.

8.7.3.37 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

<i>dt</i>	size of the time step
-----------	-----------------------

Implemented in [SWE_WavePropagationBlock](#).

8.7.4 Member Data Documentation

8.7.4.1 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.

8.7.4.2 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.

8.7.4.3 float SWE_Block::dx [protected]

mesh size of the Cartesian grid in x-direction

Definition at line 236 of file SWE_Block.hh.

8.7.4.4 float SWE_Block::dy [protected]

mesh size of the Cartesian grid in y-direction

Definition at line 237 of file SWE_Block.hh.

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

static variable that holds the gravity constant ($g = 9.81 \text{ m/s}^2$):

Definition at line 205 of file SWE_Block.hh.

8.7.4.6 Float2D SWE_Block::h [protected]

array that holds the water height for each element

Definition at line 242 of file SWE_Block.hh.

8.7.4.7 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.

8.7.4.8 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.

8.7.4.9 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.

8.7.4.10 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.

8.7.4.11 int SWE_Block::nx [protected]

size of Cartesian arrays in x-direction

Definition at line 233 of file SWE_Block.hh.

8.7.4.12 int SWE_Block::ny [protected]

size of Cartesian arrays in y-direction

Definition at line 234 of file SWE_Block.hh.

8.7.4.13 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.

8.7.4.14 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:

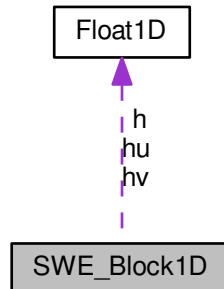
- [app/src/main/cpp/blocks/SWE_Block.hh](#)
- [app/src/main/cpp/blocks/SWE_Block.cpp](#)

8.8 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

8.8.1 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](#).

8.8.2 Constructor & Destructor Documentation

8.8.2.1 [SWE_Block1D::SWE_Block1D](#) (const [Float1D](#) & *_h*, const [Float1D](#) & *_hu*, const [Float1D](#) & *_hv*) `[inline]`

Definition at line 272 of file [SWE_Block.hh](#).

8.8.2.2 [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](#).

8.8.3 Member Data Documentation

8.8.3.1 [Float1D](#) [SWE_Block1D::h](#)

Definition at line 275 of file [SWE_Block.hh](#).

8.8.3.2 [Float1D](#) [SWE_Block1D::hu](#)

Definition at line 278 of file [SWE_Block.hh](#).

8.8.3.3 [Float1D](#) [SWE_Block1D::hv](#)

Definition at line 279 of file [SWE_Block.hh](#).

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

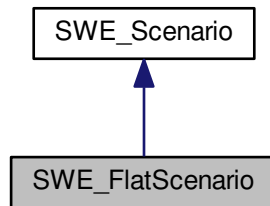
- [app/src/main/cpp/blocks/SWE_Block.hh](#)

8.9 SWE_FlatScenario Class Reference

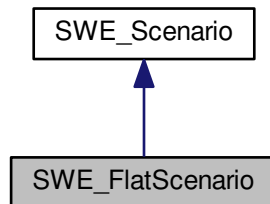
scenario is a test environment that simulates a flat water-surface

```
#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)

8.9.1 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.

8.9.2 Member Function Documentation

8.9.2.1 `virtual float SWE_FlatScenario::endSimulation () [inline],[virtual]`

Reimplemented from [SWE_Scenario](#).

Definition at line 54 of file `SWE_simple_scenarios.hh`.

8.9.2.2 `float SWE_FlatScenario::getBathymetry (float x, float y) [inline],[virtual]`

Reimplemented from [SWE_Scenario](#).

Definition at line 45 of file `SWE_simple_scenarios.hh`.

8.9.2.3 `float SWE_FlatScenario::getBoundaryPos (BoundaryEdge i_edge) [inline],[virtual]`

Get the boundary positions

Parameters

<code>i_edge</code>	which edge
---------------------	------------

Returns

value in the corresponding dimension

Reimplemented from [SWE_Scenario](#).

Definition at line 63 of file `SWE_simple_scenarios.hh`.

8.9.2.4 `virtual BoundaryType SWE_FlatScenario::getBoundaryType (BoundaryEdge edge) [inline],[virtual]`

Reimplemented from [SWE_Scenario](#).

Definition at line 56 of file `SWE_simple_scenarios.hh`.

8.9.2.5 `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:

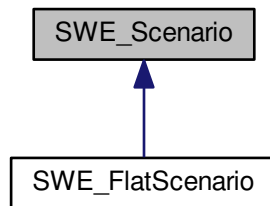
- `app/src/main/cpp/scenarios/SWE_simple_scenarios.hh`

8.10 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](#) ()

8.10.1 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 bathymetry 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.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 `virtual SWE_Scenario::~~SWE_Scenario () [inline],[virtual]`

Definition at line 76 of file SWE_Scenario.hh.

8.10.2.2 `virtual SWE_Scenario::~~SWE_Scenario () [inline],[virtual]`

Definition at line 75 of file SWE_Scenario.hh.

8.10.3 Member Function Documentation

8.10.3.1 `virtual float SWE_Scenario::endSimulation () [inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 65 of file SWE_Scenario.hh.

8.10.3.2 `virtual float SWE_Scenario::endSimulation () [inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 66 of file SWE_Scenario.hh.

8.10.3.3 `virtual float SWE_Scenario::getBathymetry (float x, float y) [inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 61 of file SWE_Scenario.hh.

8.10.3.4 `virtual float SWE_Scenario::getBathymetry (float x, float y) [inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 62 of file SWE_Scenario.hh.

8.10.3.5 `virtual float SWE_Scenario::getBoundaryPos (BoundaryEdge edge) [inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 68 of file SWE_Scenario.hh.

8.10.3.6 `virtual float SWE_Scenario::getBoundaryPos (BoundaryEdge edge)` `[inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 69 of file SWE_Scenario.hh.

8.10.3.7 `virtual BoundaryType SWE_Scenario::getBoundaryType (BoundaryEdge edge)` `[inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 67 of file SWE_Scenario.hh.

8.10.3.8 `virtual BoundaryType SWE_Scenario::getBoundaryType (BoundaryEdge edge)` `[inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 68 of file SWE_Scenario.hh.

8.10.3.9 `virtual float SWE_Scenario::getVeloc_u (float x, float y)` `[inline],[virtual]`

Definition at line 59 of file SWE_Scenario.hh.

8.10.3.10 `virtual float SWE_Scenario::getVeloc_u (float x, float y)` `[inline],[virtual]`

Definition at line 60 of file SWE_Scenario.hh.

8.10.3.11 `virtual float SWE_Scenario::getVeloc_v (float x, float y)` `[inline],[virtual]`

Definition at line 60 of file SWE_Scenario.hh.

8.10.3.12 `virtual float SWE_Scenario::getVeloc_v (float x, float y)` `[inline],[virtual]`

Definition at line 61 of file SWE_Scenario.hh.

8.10.3.13 `virtual float SWE_Scenario::getWaterHeight (float x, float y)` `[inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 58 of file SWE_Scenario.hh.

8.10.3.14 `virtual float SWE_Scenario::getWaterHeight (float x, float y) [inline],[virtual]`

Reimplemented in [SWE_FlatScenario](#).

Definition at line 59 of file SWE_Scenario.hh.

8.10.3.15 `virtual float SWE_Scenario::waterHeightAtRest () [inline],[virtual]`

Definition at line 63 of file SWE_Scenario.hh.

8.10.3.16 `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:

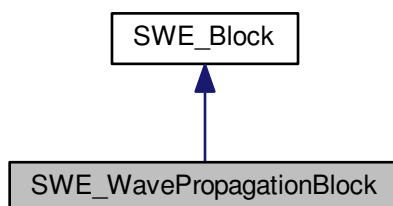
- [app/src/main/cpp/scenarios/SWE_Scenario.hh](#)

8.11 SWE_WavePropagationBlock Class Reference

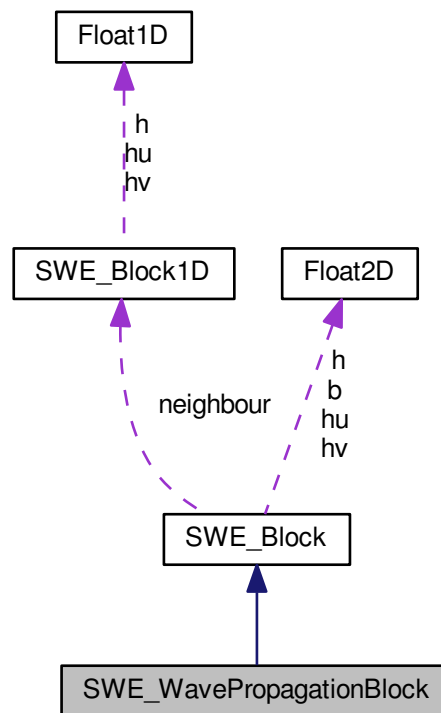
file is part of SWE

```
#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

8.11.1 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`.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 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 $[0, \dots, nx+1] \times [0, \dots, ny+1]$ (-> Abstract class [SWE_Block](#)) -> computational domain is $[1, \dots, nx] \times [1, \dots, ny]$ -> plus ghost cell layer

net-updates are defined for edges with indices $[0, \dots, nx] \times [0, \dots, ny-1]$ or $[0, \dots, nx-1] \times 0, \dots, ny$

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](#).

8.11.2.2 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](#).

8.11.3 Member Function Documentation

8.11.3.1 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.

8.11.3.2 void SWE_WavePropagationBlock::updateUnknowns (float dt) [virtual]

Updates the unknowns with the already computed net-updates.

Parameters

<i>dt</i>	time step width used in the update.
-----------	-------------------------------------

Implements [SWE_Block](#).

Definition at line 171 of file SWE_WavePropagationBlock.cpp.

8.11.3.3 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](#)

8.12 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](#)

8.13 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_huRight, 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)

8.13.1 Detailed Description

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

Definition at line 23 of file WavePropagation.hpp.

8.13.2 Member Enumeration Documentation

8.13.2.1 `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.

8.13.3 Constructor & Destructor Documentation

8.13.3.1 `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

<i>gravity</i>	gravity constant.
<i>dryTolerance</i>	numerical definition of "dry".
<i>zeroTolerance</i>	numerical definition of zero.

Definition at line 109 of file WavePropagation.hpp.

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

Definition at line 206 of file WavePropagation.hpp.

8.13.4 Member Function Documentation

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

8.13.4.2 `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.

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

Sets the dry tolerance of the solver.

Parameters

<i>i_dryTolerance</i>	dry tolerance.
-----------------------	----------------

Definition at line 202 of file WavePropagation.hpp.

8.13.4.4 `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>i_hLeft</i>	height on the left side of the edge.
<i>i_hRight</i>	height on the right side of the edge.
<i>i_huLeft</i>	momentum on the left side of the edge.
<i>i_huRight</i>	momentum on the right side of the edge.
<i>i_bLeft</i>	bathymetry on the left side of the edge.
<i>i_bRight</i>	bathymetry on the right side of the edge.

Definition at line 126 of file WavePropagation.hpp.

8.13.4.5 `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>i_hLeft</i>	height on the left side of the edge.
<i>i_hRight</i>	height on the right side of the edge.
<i>i_huLeft</i>	momentum on the left side of the edge.
<i>i_huRight</i>	momentum on the right side of the edge.
<i>i_bLeft</i>	bathymetry on the left side of the edge.
<i>i_bRight</i>	bathymetry on the right side of the edge.
<i>i_uLeft</i>	velocity on the left side of the edge.
<i>i_uRight</i>	velocity on the right side of the edge.

Definition at line 152 of file WavePropagation.hpp.

8.13.5 Member Data Documentation

8.13.5.1 `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.

8.13.5.2 `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.

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

numerical definition of "dry".

Definition at line 31 of file WavePropagation.hpp.

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

gravity constant

Definition at line 33 of file WavePropagation.hpp.

8.13.5.5 `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.

8.13.5.6 `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.

8.13.5.7 `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.

8.13.5.8 `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.

8.13.5.9 `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.

8.13.5.10 `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.

8.13.5.11 `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`.

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

numerical definition of zero.

Definition at line 35 of file `WavePropagation.hpp`.

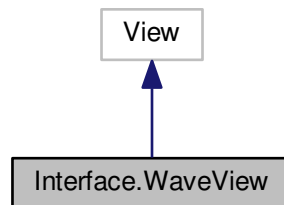
The documentation for this class was generated from the following file:

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

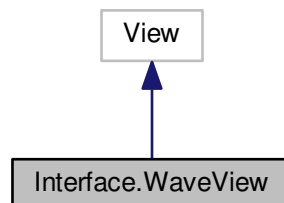
8.14 Interface.WaveView Class Reference

This class is responsible for the Wave visualization.

Inheritance diagram for `Interface.WaveView`:



Collaboration diagram for `Interface.WaveView`:



Public Member Functions

- [WaveView](#) (Context context, AttributeSet attrs)
- void [onDraw](#) (Canvas canvas)
called when something changes in the domain

Protected Member Functions

- void [onSizeChanged](#) (int w, int h, int old_w, int old_h)
called when the viewsize gets changed

8.14.1 Detailed Description

This class is responsible for the Wave visualization.

Definition at line 15 of file WaveView.java.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Interface.WaveView.WaveView (Context context, AttributeSet attrs) [inline]

if there is no Simulator running, instantiate a new one

initiate the rectangles

< calculate the position and size of the drawingrects

Definition at line 21 of file WaveView.java.

8.14.3 Member Function Documentation

8.14.3.1 void Interface.WaveView.onDraw (Canvas canvas) [inline]

called when something changes in the domain

< Check if there is an obstacle

if not draw the water

determine new color

< update nomoremovement

< set lasheight to currentheight

stops the simualtion if there is no more movement

Definition at line 62 of file WaveView.java.

8.14.3.2 `void Interface.WaveView.onSizeChanged (int w, int h, int old_w, int old_h)` `[inline],[protected]`

called when the viewsize gets changed

< calculate the position and size of the drawingrects

Definition at line 45 of file WaveView.java.

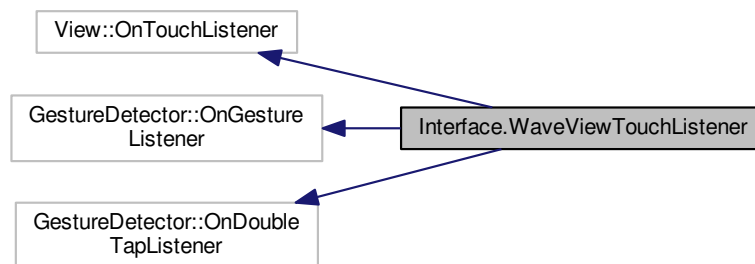
The documentation for this class was generated from the following file:

- [app/src/main/java/Interface/WaveView.java](#)

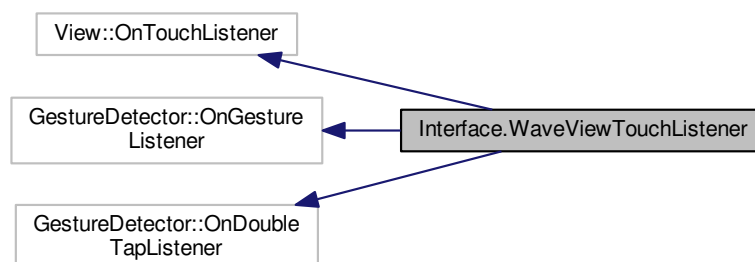
8.15 Interface.WaveViewTouchListener Class Reference

class implemets the recation of the displayed content to the users touch-inputs

Inheritance diagram for Interface.WaveViewTouchListener:



Collaboration diagram for Interface.WaveViewTouchListener:



Public Member Functions

- boolean `onTouch` (View view, MotionEvent motionEvent)
gets called if the waveview is touched
- boolean `onSingleTapConfirmed` (MotionEvent motionEvent)
adaption at single tap
- boolean `onDoubleTap` (MotionEvent motionEvent)
adaption at double-tap
- boolean `onDown` (MotionEvent motionEvent)
adaption at down-mm
- void `onShowPress` (MotionEvent motionEvent)
- boolean `onSingleTapUp` (MotionEvent motionEvent)
- boolean `onDoubleTapEvent` (MotionEvent motionEvent)
- boolean `onScroll` (MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1)
- void `onLongPress` (MotionEvent motionEvent)
adaption at press and hold
- boolean `onFling` (MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1)

Public Attributes

- int `drawingmode`

Static Public Attributes

- static final int `MODE_SIMULATE` = 0
- static final int `MODE_DRAW` = 1
- static final int `MODE_ERASE` = 2

8.15.1 Detailed Description

class implemets the recation of the displayed content to the users touch-inputs

Definition at line 10 of file WaveViewTouchListener.java.

8.15.2 Member Function Documentation

8.15.2.1 boolean Interface.WaveViewTouchListener.onDoubleTap (MotionEvent *motionEvent*) `[inline]`

adaption at double-tap

< starts or stops the simulation

Definition at line 123 of file WaveViewTouchListener.java.

8.15.2.2 boolean Interface.WaveViewTouchListener.onDoubleTapEvent (MotionEvent *motionEvent*) `[inline]`

Definition at line 150 of file WaveViewTouchListener.java.

8.15.2.3 `boolean Interface.WaveViewTouchListener.onDown (MotionEvent motionEvent)` `[inline]`

adaption at down-mm

Definition at line 136 of file WaveViewTouchListener.java.

8.15.2.4 `boolean Interface.WaveViewTouchListener.onFling (MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1)` `[inline]`

Definition at line 173 of file WaveViewTouchListener.java.

8.15.2.5 `void Interface.WaveViewTouchListener.onLongPress (MotionEvent motionEvent)` `[inline]`

adaption at press and hold

< places a circle at the pressed location

Definition at line 161 of file WaveViewTouchListener.java.

8.15.2.6 `boolean Interface.WaveViewTouchListener.onScroll (MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1)` `[inline]`

Definition at line 155 of file WaveViewTouchListener.java.

8.15.2.7 `void Interface.WaveViewTouchListener.onShowPress (MotionEvent motionEvent)` `[inline]`

Definition at line 142 of file WaveViewTouchListener.java.

8.15.2.8 `boolean Interface.WaveViewTouchListener.onSingleTapConfirmed (MotionEvent motionEvent)` `[inline]`

adaption at single tap

< if you increase the waveheights you also have to adapt the drawing

Definition at line 109 of file WaveViewTouchListener.java.

8.15.2.9 `boolean Interface.WaveViewTouchListener.onSingleTapUp (MotionEvent motionEvent)` `[inline]`

Definition at line 146 of file WaveViewTouchListener.java.

8.15.2.10 `boolean Interface.WaveViewTouchListener.onTouch (View view, MotionEvent motionEvent)` `[inline]`

gets called if the waveview is touched

-> Actions when drawing

< If a Moveevent is detected

< calculate the current finger position in the domain

if there is no last fingerpoint position, initialize with the currentposition

calculate the vector between last and current position

calculate unit vector

calculate the steps

place a circle at every step

redraw view

-> Actions when erasing

< If a Moveevent is detected

> calculate the current finger position in the domain

if there is no last fingerpoint position, initialize with the currentposition

calculate the vector between last and current position

calculate unit vector

place a circle at every step

redraw view

Definition at line 30 of file WaveViewTouchListener.java.

8.15.3 Member Data Documentation

8.15.3.1 `int Interface.WaveViewTouchListener.drawingmode`

Definition at line 15 of file WaveViewTouchListener.java.

8.15.3.2 `final int Interface.WaveViewTouchListener.MODE_DRAW = 1` `[static]`

Definition at line 12 of file WaveViewTouchListener.java.

8.15.3.3 `final int Interface.WaveViewTouchListener.MODE_ERASE = 2` `[static]`

Definition at line 13 of file WaveViewTouchListener.java.

8.15.3.4 `final int Interface.WaveViewTouchListener.MODE_SIMULATE = 0` `[static]`

Definition at line 11 of file WaveViewTouchListener.java.

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

- [app/src/main/java/Interface/WaveViewTouchListener.java](#)

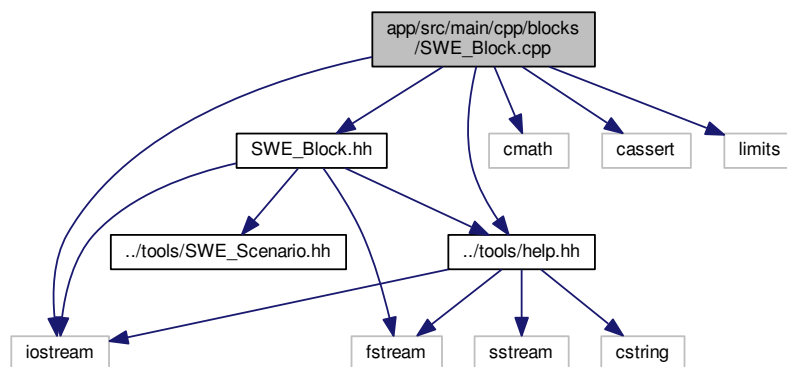
Chapter 9

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/>.

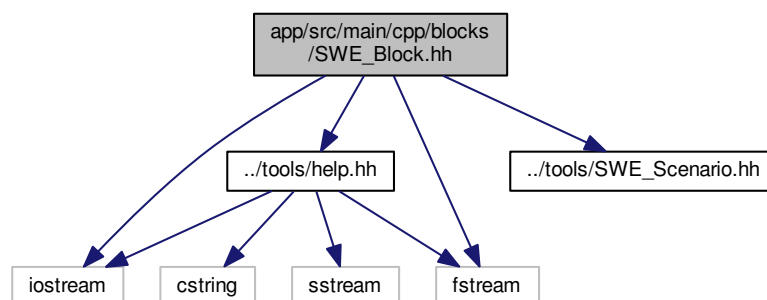
9.1.3 DESCRIPTION

TODO

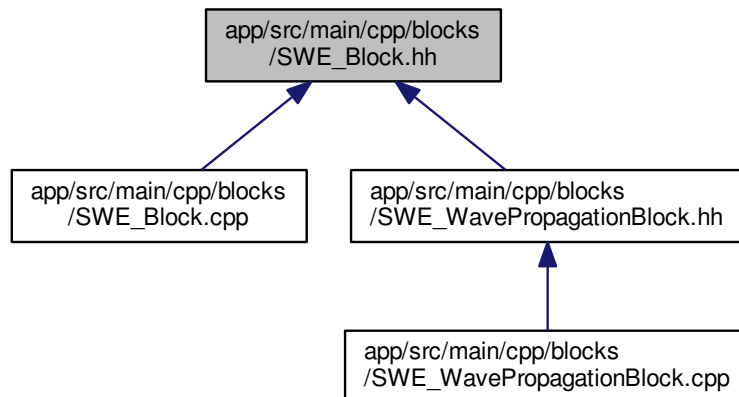
9.2 app/src/main/cpp/blocks/SWE_Block.hh File Reference

```
#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.

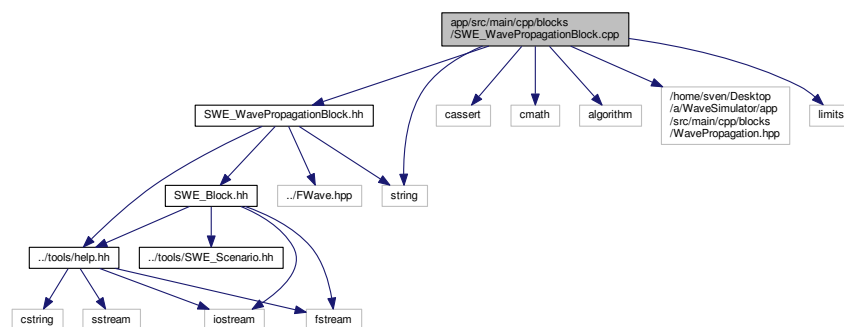
You should have received a copy of the GNU General Public License along with SWE. If not, see <http://www.↔gnu.org/licenses/>.

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/>.

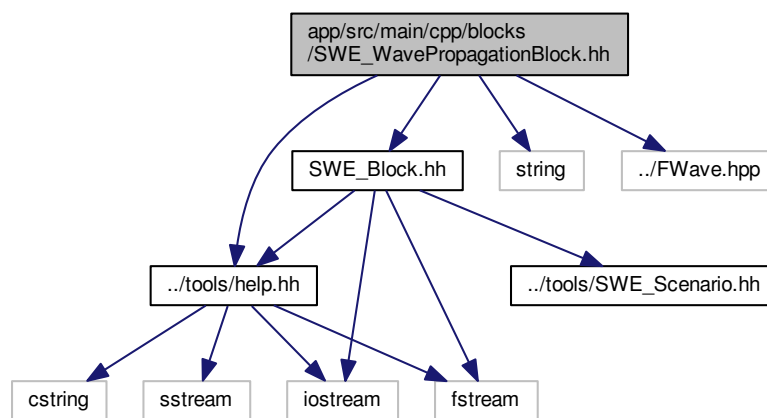
9.3.3 DESCRIPTION

Implementation of [SWE_Block](#) that uses solvers in the wave propagation formulation.

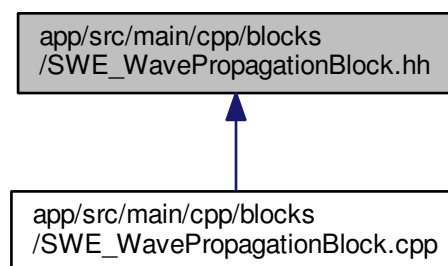
9.4 app/src/main/cpp/blocks/SWE_WavePropagationBlock.hh File Reference

```
#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/>.

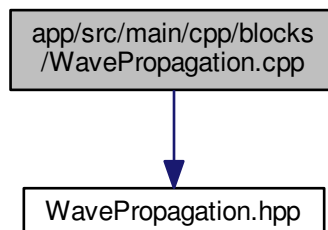
9.4.3 DESCRIPTION

Implementation of [SWE_Block](#) that uses solvers in the wave propagation formulation.

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

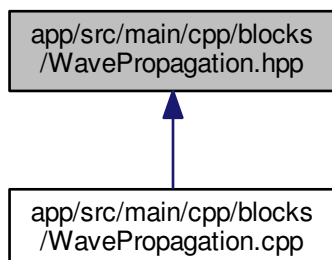
```
#include "WavePropagation.hpp"
```

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 >](#)

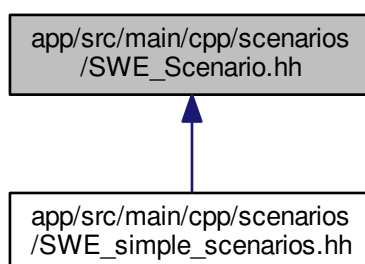
Namespaces

- [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

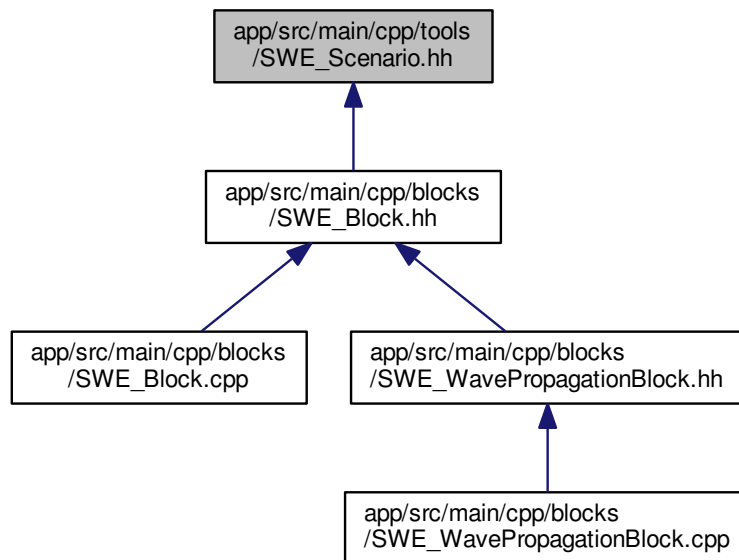
Enumerator

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

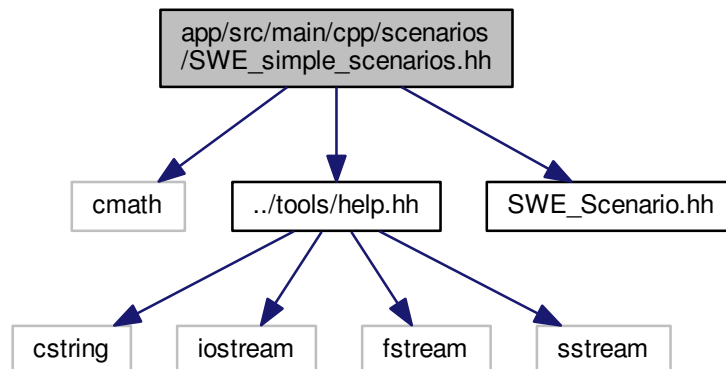
Enumerator

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/>.

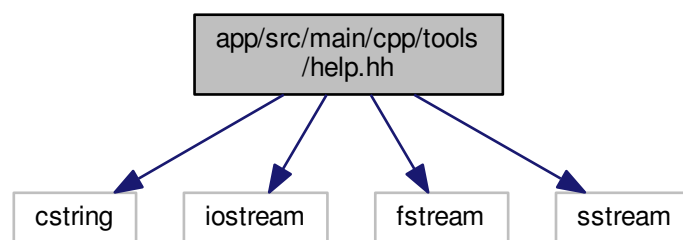
9.9.3 DESCRIPTION

TODO

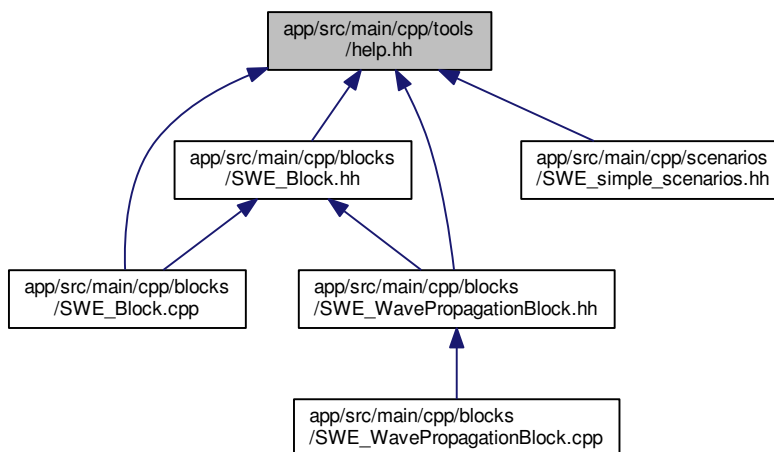
9.10 app/src/main/cpp/tools/help.hh File Reference

```
#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 a two 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_fileExtension=".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

<code>i_baseName</code>	base name of the output.
<code>i_blockPositionX</code>	position of the SWE_Block in x-direction.
<code>i_blockPositionY</code>	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>i_baseName</i>	base name of the output.
<i>i_blockPositionX</i>	position of the SWE_Block in x-direction.
<i>i_blockPositionY</i>	position of the SWE_Block in y-direction.
<i>i_fileExtension</i>	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/Interface/MainActivity.java File Reference

Classes

- class [Interface.MainActivity](#)
This class handles the main-Activities.

Packages

- package [Interface](#)
a The [Interface](#) entails all classes dedicated to the visual representation and user interaction

9.13 app/src/main/java/Interface/WaveView.java File Reference

Classes

- class [Interface.WaveView](#)
This class is responsible for the Wave visualization.

Packages

- package [Interface](#)
a The [Interface](#) entails all classes dedicated to the visual representation and user interaction

9.14 app/src/main/java/Interface/WaveViewTouchListener.java File Reference

Classes

- class [Interface.WaveViewTouchListener](#)
class implemets the recation of the displayed content to the users touch-inputs

Packages

- package [Interface](#)
a The [Interface](#) entails all classes dedicated to the visual representation and user interaction

9.15 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

9.16 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

9.17 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, [30](#)
- ~SWE_Scenario
 - SWE_Scenario, [45](#)
- ~SWE_WavePropagationBlock
 - SWE_WavePropagationBlock, [49](#)
- ~WavePropagation
 - solver::WavePropagation, [52](#)
-
- app/src/main/DocMain.cpp, [78](#)
- app/src/main/cpp/blocks/SWE_Block.cpp, [63](#)
- app/src/main/cpp/blocks/SWE_Block.hh, [64](#)
- app/src/main/cpp/blocks/SWE_WavePropagation↵
Block.cpp, [66](#)
- app/src/main/cpp/blocks/SWE_WavePropagation↵
Block.hh, [67](#)
- app/src/main/cpp/blocks/WavePropagation.cpp, [68](#)
- app/src/main/cpp/blocks/WavePropagation.hpp, [69](#)
- app/src/main/cpp/scenarios/SWE_Scenario.hh, [69](#)
- app/src/main/cpp/scenarios/SWE_simple_scenarios.hh,
[74](#)
- app/src/main/cpp/tools/SWE_Scenario.hh, [72](#)
- app/src/main/cpp/tools/help.hh, [75](#)
- app/src/main/java/Interface/MainActivity.java, [78](#)
- app/src/main/java/Interface/WaveView.java, [78](#)
- app/src/main/java/Interface/WaveViewTouchListener.↵
java, [78](#)
- app/src/main/java/Solver/CPPSimulator.java, [79](#)
- app/src/main/java/Solver/Helper.java, [79](#)
- app/src/main/java/Solver/SimulationRunner.java, [79](#)
-
- b
 - SWE_Block, [38](#)
- bLeft
 - solver::WavePropagation, [54](#)
- BND_BOTTOM
 - scenarios/SWE_Scenario.hh, [71](#)
 - tools/SWE_Scenario.hh, [73](#)
- BND_LEFT
 - scenarios/SWE_Scenario.hh, [71](#)
 - tools/SWE_Scenario.hh, [73](#)
- BND_RIGHT
 - scenarios/SWE_Scenario.hh, [71](#)
 - tools/SWE_Scenario.hh, [73](#)
- BND_TOP
 - scenarios/SWE_Scenario.hh, [71](#)
-
- tools/SWE_Scenario.hh, [73](#)
- bRight
 - solver::WavePropagation, [54](#)
- boundary
 - SWE_Block, [38](#)
- BoundaryEdge
 - scenarios/SWE_Scenario.hh, [70, 71](#)
 - tools/SWE_Scenario.hh, [73](#)
- BoundaryType
 - scenarios/SWE_Scenario.hh, [70, 71](#)
 - tools/SWE_Scenario.hh, [73](#)
-
- CONNECT
 - scenarios/SWE_Scenario.hh, [71](#)
 - tools/SWE_Scenario.hh, [73](#)
- CPPSimulator
 - Solver::CPPSimulator, [16](#)
- cell_count
 - Solver::CPPSimulator, [17](#)
- changeActivity
 - Solver::SimulationRunner, [25](#)
- computeMaxTimestep
 - SWE_Block, [31](#)
- computeNetUpdates
 - solver::WavePropagation, [53](#)
- computeNumericalFluxes
 - SWE_Block, [31](#)
 - SWE_WavePropagationBlock, [50](#)
-
- delete
 - Solver::CPPSimulator, [16](#)
- determineWetDryState
 - solver::WavePropagation, [53](#)
- DocMain.cpp
 - main, [78](#)
- drawingmode
 - Interface::WaveViewTouchListener, [61](#)
- DryDry
 - solver::WavePropagation, [52](#)
- dryTol
 - solver::WavePropagation, [54](#)
- DryWetInundation
 - solver::WavePropagation, [52](#)
- DryWetWall
 - solver::WavePropagation, [52](#)
- DryWetWallInundation
 - solver::WavePropagation, [52](#)
- dx
 - SWE_Block, [38](#)
- dy

- SWE_Block, 38
- elemVector
 - Float1D, 19
 - Float2D, 21
- endSimulation
 - SWE_FlatScenario, 43
 - SWE_Scenario, 45
- 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, 38
 - solver::WavePropagation, 55
- generateBaseFileName
 - help.hh, 76
- generateContainerFileName
 - help.hh, 77
- generateFileName
 - help.hh, 77
- getBathymetry
 - SWE_Block, 31
 - SWE_FlatScenario, 43
 - SWE_Scenario, 45
 - Solver::CPPSimulator, 16
- getBoundaryPos
 - SWE_FlatScenario, 43
 - SWE_Scenario, 45
- getBoundaryType
 - SWE_FlatScenario, 43
 - SWE_Scenario, 46
- getColProxy
 - Float2D, 21
- getCols
 - Float2D, 21
- getDischarge_hu
 - SWE_Block, 31
- getDischarge_hv
 - SWE_Block, 31
- getHeight
 - Solver::CPPSimulator, 16
- getMaxTimestep
 - SWE_Block, 32
- getNx
 - SWE_Block, 32
- getNy
 - SWE_Block, 32
- getRowProxy
 - Float2D, 21
- getRows
 - Float2D, 21
- getSimulationRunner
 - Interface::MainActivity, 23
- getSize
 - Float1D, 19
- getVeloc_u
 - SWE_Scenario, 46
- getVeloc_v
 - SWE_Scenario, 46
- getWaterHeight
 - SWE_Block, 32
 - SWE_FlatScenario, 43
 - SWE_Scenario, 46
- getWaveHeightValue
 - Interface::MainActivity, 23
- getWaveView
 - Interface::MainActivity, 24
- grabGhostLayer
 - SWE_Block, 32
- h
 - SWE_Block, 38
 - SWE_Block1D, 41
- hLeft
 - solver::WavePropagation, 55
- hRight
 - solver::WavePropagation, 55
- help.hh
 - generateBaseFileName, 76
 - generateContainerFileName, 77
 - generateFileName, 77
- hu
 - SWE_Block, 38
 - SWE_Block1D, 41
- huLeft
 - solver::WavePropagation, 55
- huRight
 - solver::WavePropagation, 55
- hv
 - SWE_Block, 39
 - SWE_Block1D, 41
- INFLOW
 - scenarios/SWE_Scenario.hh, 71
 - tools/SWE_Scenario.hh, 73
- initScenario
 - SWE_Block, 33
- Interface, 13
 - Interface.MainActivity, 22
 - Interface.WaveView, 56
 - Interface.WaveViewTouchListener, 58
 - Interface::MainActivity

- getSimulationRunner, 23
- getWaveHeightValue, 23
- getWaveView, 24
- onCheckedSwitch, 24
- onCreate, 24
- onCreateOptionsMenu, 24
- onOptionsItemSelected, 24
- WaveHeightSeekBarChanged, 24
- Interface::WaveView
 - onDraw, 57
 - onSizeChanged, 57
 - WaveView, 57
- Interface::WaveViewTouchListener
 - drawingmode, 61
 - MODE_DRAW, 61
 - MODE_ERASE, 61
 - MODE_SIMULATE, 61
 - onDoubleTap, 59
 - onDoubleTapEvent, 59
 - onDown, 59
 - onFling, 60
 - onLongPress, 60
 - onScroll, 60
 - onShowPress, 60
 - onSingleTapConfirmed, 60
 - onSingleTapUp, 60
 - onTouch, 60
- isStarted
 - Solver::SimulationRunner, 25
- linear_map
 - Solver::Helper, 22
- MODE_DRAW
 - Interface::WaveViewTouchListener, 61
- MODE_ERASE
 - Interface::WaveViewTouchListener, 61
- MODE_SIMULATE
 - Interface::WaveViewTouchListener, 61
- main
 - DocMain.cpp, 78
- maxTimestep
 - SWE_Block, 39
- neighbour
 - SWE_Block, 39
- nx
 - SWE_Block, 39
- ny
 - SWE_Block, 39
- OUTFLOW
 - scenarios/SWE_Scenario.hh, 71
 - tools/SWE_Scenario.hh, 73
- offsetX
 - SWE_Block, 39
- offsetY
 - SWE_Block, 39
- onCheckedSwitch
 - Interface::MainActivity, 24
- onCreate
 - Interface::MainActivity, 24
- onCreateOptionsMenu
 - Interface::MainActivity, 24
- onDoubleTap
 - Interface::WaveViewTouchListener, 59
- onDoubleTapEvent
 - Interface::WaveViewTouchListener, 59
- onDown
 - Interface::WaveViewTouchListener, 59
- onDraw
 - Interface::WaveView, 57
- onFling
 - Interface::WaveViewTouchListener, 60
- onLongPress
 - Interface::WaveViewTouchListener, 60
- onOptionsItemSelected
 - Interface::MainActivity, 24
- onScroll
 - Interface::WaveViewTouchListener, 60
- onShowPress
 - Interface::WaveViewTouchListener, 60
- onSingleTapConfirmed
 - Interface::WaveViewTouchListener, 60
- onSingleTapUp
 - Interface::WaveViewTouchListener, 60
- onSizeChanged
 - Interface::WaveView, 57
- onTouch
 - Interface::WaveViewTouchListener, 60
- operator[]
 - Float1D, 19
 - Float2D, 21
- PASSIVE
 - scenarios/SWE_Scenario.hh, 71
 - tools/SWE_Scenario.hh, 73
- placeCircle
 - Solver::CPPSimulator, 17
- registerCopyLayer
 - SWE_Block, 33
- reset
 - Solver::CPPSimulator, 17
- resetWaves
 - Solver::CPPSimulator, 17
- SWE_Block, 26
 - ~SWE_Block, 30
 - b, 38
 - boundary, 38
 - computeMaxTimestep, 31
 - computeNumericalFluxes, 31
 - dx, 38
 - dy, 38
 - g, 38
 - getBathymetry, 31
 - getDischarge_hu, 31

- getDischarge_hv, 31
- getMaxTimestep, 32
- getNx, 32
- getNy, 32
- getWaterHeight, 32
- grabGhostLayer, 32
- h, 38
- hu, 38
- hv, 39
- initScenario, 33
- maxTimestep, 39
- neighbour, 39
- nx, 39
- ny, 39
- offsetX, 39
- offsetY, 39
- registerCopyLayer, 33
- SWE_Block, 30
- setBathymetry, 33
- setBathymetryXY, 34
- setBoundaryBathymetry, 34
- setBoundaryConditions, 34
- setBoundaryType, 34
- setDischarge, 34
- setGhostLayer, 35
- setHuXY, 35
- setHvXY, 35
- setWaterHeight, 35
- setWaterHeightXY, 35
- simulate, 35
- simulateTimestep, 36
- synchAfterWrite, 36
- synchBathymetryAfterWrite, 36
- synchBathymetryBeforeRead, 36
- synchBeforeRead, 36
- synchCopyLayerBeforeRead, 37
- synchDischargeAfterWrite, 37
- synchDischargeBeforeRead, 37
- synchGhostLayerAfterWrite, 37
- synchWaterHeightAfterWrite, 37
- synchWaterHeightBeforeRead, 37
- updateUnknowns, 37
- SWE_Block1D, 40
 - h, 41
 - hu, 41
 - hv, 41
 - SWE_Block1D, 41
- SWE_FlatScenario, 42
 - endSimulation, 43
 - getBathymetry, 43
 - getBoundaryPos, 43
 - getBoundaryType, 43
 - getWaterHeight, 43
- SWE_Scenario, 44
 - ~SWE_Scenario, 45
 - endSimulation, 45
 - getBathymetry, 45
 - getBoundaryPos, 45
 - getBoundaryType, 46
 - getVeloc_u, 46
 - getVeloc_v, 46
 - getWaterHeight, 46
 - waterHeightAtRest, 47
- SWE_WavePropagationBlock, 47
 - ~SWE_WavePropagationBlock, 49
 - computeNumericalFluxes, 50
 - SWE_WavePropagationBlock, 49
 - updateUnknowns, 50
 - updateUnknownsRow, 50
- scenarios/SWE_Scenario.hh
 - BND_BOTTOM, 71
 - BND_LEFT, 71
 - BND_RIGHT, 71
 - BND_TOP, 71
 - BoundaryEdge, 70, 71
 - BoundaryType, 70, 71
 - CONNECT, 71
 - INFLOW, 71
 - OUTFLOW, 71
 - PASSIVE, 71
 - WALL, 71
- setBathymetry
 - SWE_Block, 33
- setBathymetryXY
 - SWE_Block, 34
- setBoundaryBathymetry
 - SWE_Block, 34
- setBoundaryConditions
 - SWE_Block, 34
- setBoundaryType
 - SWE_Block, 34
 - Solver::CPPSimulator, 17
- setDischarge
 - SWE_Block, 34
- setDryTolerance
 - solver::WavePropagation, 53
- setGhostLayer
 - SWE_Block, 35
- setHuXY
 - SWE_Block, 35
- setHvXY
 - SWE_Block, 35
- setWaterHeight
 - SWE_Block, 35
- setWaterHeightXY
 - SWE_Block, 35
- setWave
 - Solver::CPPSimulator, 17
- sim
 - Solver::CPPSimulator, 17
- simulate
 - SWE_Block, 35
- simulateTimestep
 - SWE_Block, 36
- simulatetimestep
 - Solver::CPPSimulator, 17

- SimulationRunner
 - Solver::SimulationRunner, 25
- Solver, 14
- solver, 13
- Solver.CPPSimulator, 15
- Solver.Helper, 22
- Solver.SimulationRunner, 25
- 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, 25
 - isStarted, 25
 - SimulationRunner, 25
 - start, 26
 - stop, 26
- solver::WavePropagation
 - ~WavePropagation, 52
 - bLeft, 54
 - bRight, 54
 - computeNetUpdates, 53
 - determineWetDryState, 53
 - DryDry, 52
 - dryTol, 54
 - DryWetInundation, 52
 - DryWetWall, 52
 - DryWetWallInundation, 52
 - g, 55
 - hLeft, 55
 - hRight, 55
 - huLeft, 55
 - huRight, 55
 - setDryTolerance, 53
 - storeParameters, 53, 54
 - uLeft, 55
 - uRight, 55
 - WavePropagation, 52
 - WetDryInundation, 52
 - WetDryState, 52
 - wetDryState, 55
 - WetDryWall, 52
 - WetDryWallInundation, 52
 - WetWet, 52
 - zeroTol, 56
- solver::WavePropagation< T >, 50
- start
 - Solver::SimulationRunner, 26
- stop
 - Solver::SimulationRunner, 26
- storeParameters
 - solver::WavePropagation, 53, 54
- synchAfterWrite
 - SWE_Block, 36
- synchBathymetryAfterWrite
 - SWE_Block, 36
- synchBathymetryBeforeRead
 - SWE_Block, 36
- synchBeforeRead
 - SWE_Block, 36
- synchCopyLayerBeforeRead
 - SWE_Block, 37
- synchDischargeAfterWrite
 - SWE_Block, 37
- synchDischargeBeforeRead
 - SWE_Block, 37
- synchGhostLayerAfterWrite
 - SWE_Block, 37
- synchWaterHeightAfterWrite
 - SWE_Block, 37
- synchWaterHeightBeforeRead
 - SWE_Block, 37
- tools/SWE_Scenario.hh
 - BND_BOTTOM, 73
 - BND_LEFT, 73
 - BND_RIGHT, 73
 - BND_TOP, 73
 - BoundaryEdge, 73
 - BoundaryType, 73
 - CONNECT, 73
 - INFLOW, 73
 - OUTFLOW, 73
 - PASSIVE, 73
 - WALL, 73
- uLeft
 - solver::WavePropagation, 55
- uRight
 - solver::WavePropagation, 55
- updateUnknowns
 - SWE_Block, 37
 - SWE_WavePropagationBlock, 50
- updateUnknownsRow
 - SWE_WavePropagationBlock, 50
- WALL
 - scenarios/SWE_Scenario.hh, 71
 - tools/SWE_Scenario.hh, 73
- waterHeightAtRest
 - SWE_Scenario, 47
- waterlevel
 - Solver::CPPSimulator, 18
- WaveHeightSeekBarChanged

- Interface::MainActivity, [24](#)
- WavePropagation, [50](#)
 - solver::WavePropagation, [52](#)
- WaveView
 - Interface::WaveView, [57](#)
- WetDryInundation
 - solver::WavePropagation, [52](#)
- WetDryState
 - solver::WavePropagation, [52](#)
- wetDryState
 - solver::WavePropagation, [55](#)
- WetDryWall
 - solver::WavePropagation, [52](#)
- WetDryWallInundation
 - solver::WavePropagation, [52](#)
- WetWet
 - solver::WavePropagation, [52](#)
- zeroTol
 - solver::WavePropagation, [56](#)