

SWOCS

Version 1.0

Author: Javier Burguete Tolosa

Departamento de Suelo y Agua, Estación Experimental de Aula Dei, CSIC

Avda. Montañana 1005, 50059 Zaragoza, Spain

Generated by Doxygen 1.8.1

Wed Jun 20 2012 09:28:58

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	Channel Struct Reference	5
3.1.1	Detailed Description	6
3.2	Hydrogram Struct Reference	6
3.2.1	Detailed Description	6
3.3	Mesh Struct Reference	6
3.3.1	Detailed Description	7
3.4	Model Struct Reference	7
3.4.1	Detailed Description	8
3.5	Node Struct Reference	8
3.5.1	Detailed Description	10
3.6	Probes Struct Reference	10
3.6.1	Detailed Description	11
4	File Documentation	13
4.1	channel.c File Reference	13
4.1.1	Detailed Description	13
4.1.2	Function Documentation	14
4.1.2.1	channel_diffusion_read_Rutherford	14
4.1.2.2	channel_friction_read_Manning	14
4.1.2.3	channel_infiltration_read_KostiakovLewis	15
4.1.2.4	channel_read	15
4.1.2.5	hydrogram_discharge	17
4.1.2.6	hydrogram_integrate	17
4.1.2.7	hydrogram_read	18
4.1.2.8	interpolate	19

4.2	channel.c	19
4.3	channel.h File Reference	22
4.3.1	Detailed Description	23
4.3.2	Function Documentation	23
4.3.2.1	channel_diffusion_read_Rutherford	23
4.3.2.2	channel_friction_read_Manning	24
4.3.2.3	channel_infiltration_read_KostiakovLewis	24
4.3.2.4	channel_read	25
4.3.2.5	hydrogram_discharge	26
4.3.2.6	hydrogram_integrate	27
4.3.2.7	hydrogram_read	28
4.3.2.8	interpolate	28
4.4	channel.h	29
4.5	config.h File Reference	30
4.5.1	Detailed Description	30
4.6	config.h	30
4.7	main.c File Reference	31
4.7.1	Detailed Description	31
4.8	main.c	32
4.9	mesh.c File Reference	35
4.9.1	Detailed Description	36
4.9.2	Function Documentation	36
4.9.2.1	mesh_initial_conditions_dry	36
4.9.2.2	mesh_initial_conditions_profile	36
4.9.2.3	mesh_open	37
4.9.2.4	mesh_read	38
4.9.2.5	mesh_solute_mass	39
4.9.2.6	mesh_water_mass	39
4.9.2.7	mesh_write_flows	40
4.9.2.8	mesh_write_variables	40
4.10	mesh.c	41
4.11	mesh.h File Reference	44
4.11.1	Detailed Description	44
4.11.2	Function Documentation	44
4.11.2.1	mesh_open	44
4.11.2.2	mesh_read	45
4.11.2.3	mesh_solute_mass	46
4.11.2.4	mesh_water_mass	46
4.11.2.5	mesh_write_flows	47
4.11.2.6	mesh_write_variables	47

4.12	mesh.h	48
4.13	model.c File Reference	48
4.13.1	Detailed Description	49
4.13.2	Function Documentation	50
4.13.2.1	model_diffusion_explicit	50
4.13.2.2	model_diffusion_implicit	50
4.13.2.3	model_infiltration	51
4.13.2.4	model_node_diffusion_1dt_max	51
4.13.2.5	model_parameters	51
4.13.2.6	model_print	52
4.13.2.7	model_probes_read	52
4.13.2.8	model_read	53
4.13.2.9	model_step	54
4.13.2.10	model_write_advance	54
4.13.2.11	model_write_probes	55
4.14	model.c	55
4.15	model.h File Reference	59
4.15.1	Detailed Description	60
4.15.2	Function Documentation	60
4.15.2.1	model_diffusion_explicit	60
4.15.2.2	model_diffusion_implicit	60
4.15.2.3	model_infiltration	61
4.15.2.4	model_node_diffusion_1dt_max	61
4.15.2.5	model_parameters	62
4.15.2.6	model_print	62
4.15.2.7	model_probes_read	62
4.15.2.8	model_read	63
4.15.2.9	model_step	64
4.15.2.10	model_write_advance	65
4.15.2.11	model_write_probes	65
4.16	model.h	66
4.17	model_complete.c File Reference	67
4.17.1	Detailed Description	67
4.17.2	Function Documentation	67
4.17.2.1	model_inlet_dtmax_complete	67
4.17.2.2	model_node_parameters_complete	68
4.17.2.3	node_1dt_max_complete	68
4.17.2.4	node_flows_complete	69
4.18	model_complete.c	69
4.19	model_complete.h File Reference	70

4.19.1 Detailed Description	71
4.19.2 Function Documentation	71
4.19.2.1 model_inlet_dtmax_complete	71
4.19.2.2 model_node_parameters_complete	71
4.19.2.3 node_1dt_max_complete	72
4.19.2.4 node_flows_complete	72
4.20 model_complete.h	73
4.21 model_complete_LaxFriedrichs.c File Reference	73
4.21.1 Detailed Description	74
4.21.2 Function Documentation	74
4.21.2.1 model_surface_flow_complete_LaxFriedrichs	74
4.22 model_complete_LaxFriedrichs.c	75
4.23 model_complete_LaxFriedrichs.h File Reference	76
4.23.1 Detailed Description	76
4.23.2 Function Documentation	76
4.23.2.1 model_surface_flow_complete_LaxFriedrichs	76
4.24 model_complete_LaxFriedrichs.h	77
4.25 model_diffusive.c File Reference	78
4.25.1 Detailed Description	78
4.25.2 Function Documentation	79
4.25.2.1 model_inlet_dtmax_diffusive	79
4.25.2.2 model_node_parameters_centre_diffusive	79
4.25.2.3 model_node_parameters_left_diffusive	80
4.25.2.4 model_node_parameters_right_diffusive	80
4.25.2.5 node_1dt_max_diffusive	81
4.25.2.6 node_discharge_centre_diffusive_Manning	81
4.25.2.7 node_discharge_left_diffusive_Manning	81
4.25.2.8 node_discharge_right_diffusive_Manning	82
4.25.2.9 node_flows_diffusive	82
4.26 model_diffusive.c	82
4.27 model_diffusive.h File Reference	84
4.27.1 Detailed Description	85
4.27.2 Function Documentation	85
4.27.2.1 model_inlet_dtmax_diffusive	85
4.27.2.2 model_node_parameters_centre_diffusive	86
4.27.2.3 model_node_parameters_left_diffusive	86
4.27.2.4 model_node_parameters_right_diffusive	87
4.27.2.5 node_1dt_max_diffusive	87
4.27.2.6 node_discharge_centre_diffusive_Manning	88
4.27.2.7 node_discharge_left_diffusive_Manning	88

4.27.2.8	node_discharge_right_diffusive_Manning	88
4.27.2.9	node_flows_diffusive	88
4.28	model_diffusive.h	89
4.29	model_diffusive_upwind.c File Reference	89
4.29.1	Detailed Description	90
4.29.2	Function Documentation	90
4.29.2.1	model_surface_flow_diffusive_upwind	90
4.30	model_diffusive_upwind.c	91
4.31	model_diffusive_upwind.h File Reference	91
4.31.1	Detailed Description	92
4.31.2	Function Documentation	92
4.31.2.1	model_surface_flow_diffusive_upwind	92
4.32	model_diffusive_upwind.h	92
4.33	model_kinematic.c File Reference	93
4.33.1	Detailed Description	93
4.33.2	Function Documentation	94
4.33.2.1	model_inlet_dtmax_kinematic	94
4.33.2.2	model_node_parameters_centre_kinematic	94
4.33.2.3	model_node_parameters_left_kinematic	95
4.33.2.4	model_node_parameters_right_kinematic	95
4.33.2.5	node_1dt_max_kinematic	96
4.33.2.6	node_discharge_centre_kinematic_Manning	96
4.33.2.7	node_discharge_left_kinematic_Manning	96
4.33.2.8	node_discharge_right_kinematic_Manning	97
4.33.2.9	node_flows_kinematic	97
4.34	model_kinematic.c	97
4.35	model_kinematic.h File Reference	99
4.35.1	Detailed Description	100
4.35.2	Function Documentation	100
4.35.2.1	model_inlet_dtmax_kinematic	100
4.35.2.2	model_node_parameters_centre_kinematic	101
4.35.2.3	model_node_parameters_left_kinematic	101
4.35.2.4	model_node_parameters_right_kinematic	102
4.35.2.5	node_1dt_max_kinematic	102
4.35.2.6	node_discharge_centre_kinematic_Manning	102
4.35.2.7	node_discharge_left_kinematic_Manning	103
4.35.2.8	node_discharge_right_kinematic_Manning	103
4.35.2.9	node_flows_kinematic	103
4.36	model_kinematic.h	104
4.37	model_kinematic_upwind.c File Reference	104

4.37.1 Detailed Description	105
4.37.2 Function Documentation	105
4.37.2.1 model_surface_flow_kinematic_upwind	105
4.38 model_kinematic_upwind.c	105
4.39 model_kinematic_upwind.h File Reference	106
4.39.1 Detailed Description	106
4.39.2 Function Documentation	107
4.39.2.1 model_surface_flow_kinematic_upwind	107
4.40 model_kinematic_upwind.h	107
4.41 model_zero_inertia.c File Reference	108
4.41.1 Detailed Description	108
4.41.2 Function Documentation	108
4.41.2.1 model_inlet_dtmax_zero_inertia	108
4.41.2.2 model_node_parameters_zero_inertia	109
4.41.2.3 node_1dt_max_zero_inertia	109
4.41.2.4 node_flows_zero_inertia	110
4.42 model_zero_inertia.c	110
4.43 model_zero_inertia.h File Reference	111
4.43.1 Detailed Description	111
4.43.2 Function Documentation	112
4.43.2.1 model_inlet_dtmax_zero_inertia	112
4.43.2.2 model_node_parameters_zero_inertia	112
4.43.2.3 node_1dt_max_zero_inertia	113
4.43.2.4 node_flows_zero_inertia	113
4.44 model_zero_inertia.h	113
4.45 model_zero_inertia_LaxFriedrichs.c File Reference	114
4.45.1 Detailed Description	114
4.45.2 Function Documentation	115
4.45.2.1 model_surface_flow_zero_inertia_LaxFriedrichs	115
4.46 model_zero_inertia_LaxFriedrichs.c	115
4.47 model_zero_inertia_LaxFriedrichs.h File Reference	117
4.47.1 Detailed Description	117
4.47.2 Function Documentation	117
4.47.2.1 model_surface_flow_zero_inertia_LaxFriedrichs	117
4.48 model_zero_inertia_LaxFriedrichs.h	118
4.49 node.c File Reference	119
4.49.1 Detailed Description	119
4.49.2 Function Documentation	120
4.49.2.1 node_critical_depth	120
4.49.2.2 node_critical_velocity	120

4.49.2.3	node_depth	121
4.49.2.4	node_diffusion_Rutherford	121
4.49.2.5	node_friction_Manning	121
4.49.2.6	node_infiltration_KostiakovLewis	121
4.49.2.7	node_inlet	122
4.49.2.8	node_outlet_closed	122
4.49.2.9	node_outlet_open	122
4.49.2.10	node_perimeter	123
4.49.2.11	node_subcritical_discharge	123
4.49.2.12	node_width	123
4.50	node.c	123
4.51	node.h File Reference	125
4.51.1	Detailed Description	126
4.51.2	Function Documentation	126
4.51.2.1	node_critical_depth	126
4.51.2.2	node_critical_velocity	127
4.51.2.3	node_depth	127
4.51.2.4	node_diffusion_Rutherford	128
4.51.2.5	node_friction_Manning	128
4.51.2.6	node_infiltration_KostiakovLewis	128
4.51.2.7	node_inlet	129
4.51.2.8	node_outlet_closed	129
4.51.2.9	node_outlet_open	129
4.51.2.10	node_perimeter	130
4.51.2.11	node_subcritical_discharge	130
4.51.2.12	node_width	130
4.52	node.h	130

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Channel	Struct to define a channel	5
Hydrogram	Struct to define a hydrogram	6
Mesh	Struct to define a mesh	6
Model	Struct to define a numerical model	7
Node	Struct to define a mesh node	8
Probes	Struct to define probes to save the evolution of the variables at a mesh cell	10

Chapter 2

File Index

2.1 File List

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

channel.c	Source file to define a channel	13
channel.h	Header file to define a channel	22
config.h	Configuration header file	30
main.c	Main source code	31
mesh.c	Source file to define a mesh	35
mesh.h	Header file to define a mesh	44
model.c	Source file to define the numerical model	48
model.h	Header file to define the numerical model	59
model_complete.c	Source file to define the complete model	67
model_complete.h	Header file to define the complete model	70
model_complete_LaxFriedrichs.c	Source file to define the Lax-Friedrichs numerical model applied to the complete model	73
model_complete_LaxFriedrichs.h	Header file to define the Lax-Friedrichs numerical model applied to the complete model	76
model_diffusive.c	Source file to define the diffusive model	78
model_diffusive.h	Header file to define the diffusive model	84
model_diffusive_upwind.c	Source file to define the upwind numerical model applied to the diffusive model	89
model_diffusive_upwind.h	Header file to define the upwind numerical model applied to the diffusive model	91
model_kinematic.c	Source file to define the kinematic model	93
model_kinematic.h	Header file to define the kinematic model	99
model_kinematic_upwind.c	Source file to define the upwind numerical model applied to the kinematic model	104

model_kinematic_upwind.h	
Header file to define the upwind numerical model applied to the kinematic model	106
model_zero_inertia.c	
Source file to define the zero inertia model	108
model_zero_inertia.h	
Header file to define the zero inertia model	111
model_zero_inertia_LaxFriedrichs.c	
Source file to define the Lax-Friedrichs numerical model applied to the zero-inertia model . . .	114
model_zero_inertia_LaxFriedrichs.h	
Header file to define the Lax-Friedrichs numerical model applied to the zero-inertia model . . .	117
node.c	
Source file to define a mesh node	119
node.h	
Header file to define a mesh node	125

Chapter 3

Data Structure Documentation

3.1 Channel Struct Reference

Struct to define a channel.

```
#include <channel.h>
```

Data Fields

- Hydrogram `water_inlet` [1]
hydrogram of water inlet.
- Hydrogram `solute_inlet` [1]
hydrogram of solute inlet.
- double `friction_coefficient` [3]
array of friction coefficients.
- double `infiltration_coefficient` [4]
array of infiltration coefficients.
- double `diffusion_coefficient` [1]
array of diffusion coefficients.
- double `slope`
channel slope.
- double `length`
channel length.
- double `bottom_width`
bottom width.
- double `wall_slope`
slope of the lateral walls.
- double `height`
channel height.
- int `type_outlet`
type of outlet (1 closed, 2 open).
- int `friction_model`
type of friction model (1 Gauckler-Manning).
- int `infiltration_model`
type of infiltration model (1 Kostiakov-Lewis).
- int `diffusion_model`
type of diffusion model (1 Rutherford).

3.1.1 Detailed Description

Struct to define a channel.

Definition at line 67 of file [channel.h](#).

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

- [channel.h](#)

3.2 Hydrogram Struct Reference

Struct to define a hydrogram.

```
#include <channel.h>
```

Data Fields

- double * [t](#)
array of times.
- double * [Q](#)
array of discharges.
- int [n](#)
number of points defining the hydrogram.

3.2.1 Detailed Description

Struct to define a hydrogram.

Definition at line 44 of file [channel.h](#).

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

- [channel.h](#)

3.3 Mesh Struct Reference

Struct to define a mesh.

```
#include <mesh.h>
```

Data Fields

- Node * [node](#)
array of node structs.
- int [n](#)
number of nodes.
- int [type](#)
initial conditions type (1 dry, 2 longitudinal profile).

3.3.1 Detailed Description

Struct to define a mesh.

Definition at line 44 of file [mesh.h](#).

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

- [mesh.h](#)

3.4 Model Struct Reference

Struct to define a numerical model.

```
#include <model.h>
```

Data Fields

- Mesh [mesh](#) [1]
mesh struct.
- Channel [channel](#) [1]
channel struct.
- Probes [probes](#) [1]
probes struct.
- double [t](#)
actual time.
- double [t2](#)
next time.
- double [dt](#)
time step size.
- double [tfinal](#)
final time.
- double [cfl](#)
CFL number.
- double [interval](#)
time interval to save the data.
- double [minimum_depth](#)
minimum depth allowing the water movement.
- void(* [model_node_parameters_centre](#))(struct [_Model](#) *model, Node *node)
pointer to the function calculating the node parameters in a centred form.
- void(* [model_node_parameters_right](#))(struct [_Model](#) *model, Node *node)
pointer to the function calculating the node parameters in an right upwind form.
- void(* [model_node_parameters_left](#))(struct [_Model](#) *model, Node *node)
pointer to the function calculating the node parameters in an left upwind form.
- double(* [node_1dt_max](#))(Node *node)
pointer to the function calculating the maximum allowed time at a node.
- double(* [model_inlet_dtmx](#))(struct [_Model](#) *model)
pointer to the function calculating the maximum allowed time at the inlet
- void(* [node_flows](#))(Node *node1)
pointer to the function calculating the node flows.
- void(* [node_discharge_centre](#))(Node *node)
pointer to the function calculating the node discharge in a centred form.

- `void(* node_discharge_right)(Node *node)`
pointer to the function calculating the node discharge in an right upwind form.
- `void(* node_discharge_left)(Node *node)`
pointer to the function calculating the node discharge in an left upwind form.
- `void(* node_friction)(Node *node)`
pointer to the function calculating the node friction.
- `void(* node_infiltration)(Node *node)`
pointer to the function calculating the node infiltration.
- `void(* node_diffusion)(Node *node)`
pointer to the function calculating the node diffusion.
- `void(* node_inlet)(Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2)`
pointer to the function calculating the inlet.
- `void(* node_outlet)(Node *node)`
pointer to the function calculating the outlet.
- `void(* model_surface_flow)(struct _Model *model)`
pointer to the function defining the numerical surface flow scheme.
- `void(* model_diffusion)(struct _Model *model)`
pointer to the function defining the numerical diffusion scheme.
- `int type_surface_flow`
type of numerical surface flow scheme (1 McCormack, 2 upwind).
- `int type_diffusion`
type of numerical diffusion scheme (1 explicit, 2 implicit).
- `int type_model`
type of model (1 complete, 2 zero-inertia, 3 diffusive, 4 kinematic).

3.4.1 Detailed Description

Struct to define a numerical model.

Definition at line 68 of file [model.h](#).

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

- [model.h](#)

3.5 Node Struct Reference

Struct to define a mesh node.

```
#include <node.h>
```

Data Fields

- `double friction_coefficient [3]`
array of friction coefficients.
- `double infiltration_coefficient [4]`
array of infiltration coefficients.
- `double diffusion_coefficient [1]`
array of diffusion coefficients.
- `double x`
position.

- double **dx**
cell size.
- double **ix**
cell distance.
- double **A**
wetted cross sectional area.
- double **Ai**
infiltrated cross sectional area.
- double **Q**
discharge.
- double **s**
solute concentration.
- double **si**
infiltrated solute concentration.
- double **As**
 $A * s.$
- double **Asi**
 $A * si.$
- double **h**
depth.
- double **Sf**
friction slope.
- double **zb**
bottom level.
- double **zs**
surface level.
- double **P**
wetted perimeter.
- double **B**
surface width.
- double **u**
velocity.
- double **c**
critical velocity.
- double **l1**
first eigenvalue.
- double **l2**
second eigenvalue.
- double **i**
infiltration velocity.
- double **Pi**
 $P * i.$
- double **Z**
lateral wall slope.
- double **B0**
bottom width.
- double **F**
 $A * u * u.$
- double **T**
 $Q * s.$
- double **Kx**

- diffusion coefficient.*
- double [KxA](#)
 - $Kx * A.$
- double [Kxi](#)
 - soil diffusion coefficient.*
- double [KxiA](#)
 - $Kxi * A.$
- double [dQ](#)
 - mass flux difference.*
- double [dF](#)
 - momentum flux difference.*
- double [dT](#)
 - solute mass flux difference.*
- double [dQl](#)
 - left numerical mass flux difference.*
- double [dFl](#)
 - left numerical momentum flux difference.*
- double [dTl](#)
 - left numerical solute mass flux difference.*
- double [dQr](#)
 - right numerical mass flux difference.*
- double [dFr](#)
 - right numerical momentum flux difference.*
- double [dTr](#)
 - right numerical solute mass flux difference.*
- double [nu](#)
 - artificial viscosity coefficient.*

3.5.1 Detailed Description

Struct to define a mesh node.

Definition at line [44](#) of file [node.h](#).

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

- [node.h](#)

3.6 Probes Struct Reference

Struct to define probes to save the evolution of the variables at a mesh cell.

```
#include <model.h>
```

Data Fields

- double * [x](#)
 - array of x-coordinates of the probes.*
- int * [node](#)
 - array of positions of the probes in the mesh.*
- int [n](#)
 - number of probes.*

3.6.1 Detailed Description

Struct to define probes to save the evolution of the variables at a mesh cell.

Definition at line 45 of file [model.h](#).

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

- [model.h](#)

Chapter 4

File Documentation

4.1 channel.c File Reference

Source file to define a channel.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "channel.h"
```

Functions

- double [interpolate](#) (double x, double x1, double x2, double y1, double y2)
Function to calculate an interpolation.
- int [hydrogram_read](#) (Hydrogram *hydrogram, FILE *file)
Function to read the data of a hydrogram.
- double [hydrogram_discharge](#) (Hydrogram *hydrogram, double t)
Function to calculate the discharge in a hydrogram.
- double [hydrogram_integrate](#) (Hydrogram *hydrogram, double t1, double t2)
Function to integrate the mass flux in a hydrogram.
- int [channel_friction_read_Manning](#) (Channel *channel, FILE *file)
Function to read the friction coefficient of the Manning model.
- int [channel_infiltration_read_KostiakovLewis](#) (Channel *channel, FILE *file)
function to read the infiltration coefficients of the Kostiakov-Lewis model.
- int [channel_diffusion_read_Rutherford](#) (Channel *channel, FILE *file)
Function to read the diffusion coefficient of the Rutherford model.
- int [channel_read](#) (Channel *channel, FILE *file)
function to read a channel.

4.1.1 Detailed Description

Source file to define a channel.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [channel.c](#).

4.1.2 Function Documentation

4.1.2.1 `int channel_diffusion_read_Rutherford (Channel * channel, FILE * file)`

Function to read the diffusion coefficient of the Rutherford model.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line [252](#) of file [channel.c](#).

```
{
    if (fscanf(file, "%lf", channel->diffusion_coefficient) != 1
        || channel->diffusion_coefficient[0] < 0.)
    {
        printf("channel diffusion: bad defined\n");
        return 0;;
    }
    return 1;
}
```

4.1.2.2 `int channel_friction_read_Manning (Channel * channel, FILE * file)`

Function to read the friction coefficient of the Manning model.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line [190](#) of file [channel.c](#).

```
{
    if (fscanf(file, "%lf", channel->friction_coefficient) != 1
        || channel->friction_coefficient[0] < 0.)
    {
        printf("channel friction: bad defined\n");
        return 0;;
    }
    #if DEBUG_MODEL_READ
        printf("channel friction: coefficient1=%lf\n",
            channel->friction_coefficient[0]);
    #endif
    return 1;
}
```


4.1.2.3 int channel_infiltration_read_KostiakovLewis (Channel * *channel*, FILE * *file*)

function to read the infiltration coefficients of the Kostiakov-Lewis model.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line 215 of file [channel.c](#).

```
{
    if (fscanf(file, "%lf%lf%lf%lf",
        channel->infiltration_coefficient,
        channel->infiltration_coefficient + 1,
        channel->infiltration_coefficient + 2,
        channel->infiltration_coefficient + 3) != 4
        || channel->infiltration_coefficient[0] < 0.
        || channel->infiltration_coefficient[1] < 0.
        || channel->infiltration_coefficient[3] <= 0.)
    {
        printf("channel infiltration: bad defined\n");
        return 0;
    }
    #if DEBUG_MODEL_READ
    printf("channel infiltration:\n"
        "coefficient1=%lf\n"
        "coefficient2=%lf\n"
        "coefficient3=%lf\n"
        "coefficient4=%lf\n",
        channel->infiltration_coefficient[0],
        channel->infiltration_coefficient[1],
        channel->infiltration_coefficient[2],
        channel->infiltration_coefficient[3]);
    #endif
    return 1;
}
```

4.1.2.4 int channel_read (Channel * *channel*, FILE * *file*)

function to read a channel.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success

Definition at line 272 of file [channel.c](#).

```
{
    char *msg;
    if (fscanf(file, "%lf%lf%lf%lf%lf%d%d%d",
        &channel->length,
        &channel->slope,
        &channel->bottom_width,
        &channel->wall_slope,
        &channel->height,
        &channel->type_outlet,
        &channel->friction_model,
        &channel->infiltration_model,
        &channel->diffusion_model) != 9)
```

```

    {
        msg = "channel: bad defined\n";
        goto bad;
    }
}
#ifdef DEBUG_MODEL_READ
printf("channel:\n"
    "length=%lf slope=%lf\n"
    "bottom_width=%lf wall_slope=%lf\n"
    "height=%lf type_outlet=%d\n"
    "friction_model=%d infiltration_model=%d diffusion_model=%d\n",
    channel->length,
    channel->slope,
    channel->bottom_width,
    channel->wall_slope,
    channel->height,
    channel->type_outlet,
    channel->friction_model,
    channel->infiltration_model,
    channel->diffusion_model);
#endif
if (channel->length <= 0.)
{
    msg = "channel: bad length\n";
    goto bad;
}
if (channel->bottom_width < 0.)
{
    msg = "channel: bad bottom width\n";
    goto bad;
}
if (channel->wall_slope < 0.)
{
    msg = "channel: bad wall slope\n";
    goto bad;
}
if (channel->height <= 0.)
{
    msg = "channel: bad height\n";
    goto bad;
}
switch (channel->type_outlet)
{
case 1:
case 2:
    break;
default:
    msg = "channel: bad outlet\n";
    goto bad;
}
switch (channel->friction_model)
{
case 1:
    if (!channel_friction_read_Manning(channel
        , file)) return 0;
    break;
default:
    msg = "channel: bad friction model\n";
    goto bad;
}
switch (channel->infiltration_model)
{
case 1:
    if (!channel_infiltration_read_KostiakovLewis
        (channel, file)) return 0;
    break;
default:
    msg = "channel: bad infiltration model\n";
    goto bad;
}
switch (channel->diffusion_model)
{
case 1:
    if (!channel_diffusion_read_Rutherford
        (channel, file)) return 0;
    break;
default:
    msg = "channel: bad diffusion model\n";
    goto bad;
}
if (!hydrogram_read(channel->water_inlet, file))
{
    msg = "channel: inlet\n";
    goto bad;
}
if (!hydrogram_read(channel->solute_inlet, file))
{
    msg = "channel: outlet\n";

```

```

        goto bad;
    }
    return 1;
bad:
    printf(msg);
    return 0;
}

```

4.1.2.5 double hydrogram.discharge (Hydrogram * *hydrogram*, double *t1*)

Function to calculate the discharge in a hydrogram.

Parameters

<i>hydrogram</i>	hydrogram struct.
<i>t</i>	time.

Returns

discharge.

Definition at line 115 of file [channel.c](#).

```

{
    int i, n1;
    n1 = hydrogram->n - 1;
    if (t <= hydrogram->t[0]) return hydrogram->Q[0];
    if (t >= hydrogram->t[n1]) return hydrogram->Q[n1];
    for (i = 0; t > hydrogram->t[i];) ++i;
    return interpolate(t, hydrogram->t[i], hydrogram->t[i - 1],
        hydrogram->Q[i], hydrogram->Q[i - 1]);
}

```

4.1.2.6 double hydrogram.integrate (Hydrogram * *hydrogram*, double *t1*, double *t2*)

Function to integrate the mass flux in a hydrogram.

Parameters

<i>hydrogram</i>	hydrogram struct.
<i>t1</i>	initial time.
<i>t2</i>	final time.

Returns

integral of the mass flux.

Definition at line 137 of file [channel.c](#).

```

{
    int i, j, n1;
    double Q1, Q2, I;
    n1 = hydrogram->n - 1;
    if (t2 <= hydrogram->t[0]) return hydrogram->Q[0] * (t2 - t1);
    if (t1 >= hydrogram->t[n1]) return hydrogram->Q[n1] * (t2 - t1);
    for (i = 0; t1 > hydrogram->t[i];) ++i;
    for (j = i; j < hydrogram->n && t2 > hydrogram->t[j];) ++j;
    if (i == j)
    {
        Q1 = interpolate(t1, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
        Q2 = interpolate(t2, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
        return 0.5 * (Q1 + Q2) * (t2 - t1);
    }
}

```

```

    }
    if (i == 0)
    {
        Q1 = hydrogram->Q[0];
    }
    else
    {
        Q1 = interpolate(t1, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
    }
    I = 0.5 * (Q1 + hydrogram->Q[i]) * (hydrogram->t[i] - t1);
    while (++i < j)
    {
        I += 0.5 * (hydrogram->Q[i] - hydrogram->Q[i - 1])
            * (hydrogram->t[i] - hydrogram->t[i - 1]);
    }
    if (i == hydrogram->n)
    {
        Q2 = hydrogram->Q[n1];
    }
    else
    {
        Q2 = interpolate(t2, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
    }
    return I + 0.5 * (Q2 + hydrogram->Q[i - 1]) * (t2 - hydrogram->t[i - 1]);
}

```

4.1.2.7 int hydrogram_read (Hydrogram * *hydrogram*, FILE * *file*)

Function to read the data of a hydrogram.

Parameters

<i>hydrogram</i>	hydrogram struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line 69 of file [channel.c](#).

```

{
    int i;
    char *msg;
    if (fscanf(file, "%d", &hydrogram->n) != 1 || hydrogram->n < 1)
    {
        msg = "hydrogram: bad points number\n";
        goto bad;
    }
    #if DEBUG_MODEL_READ
    printf("hydrogram: n=%d\n", hydrogram->n);
    #endif
    hydrogram->t = (double*)malloc(hydrogram->n * sizeof(double));
    hydrogram->Q = (double*)malloc(hydrogram->n * sizeof(double));
    if (!hydrogram->t || !hydrogram->Q)
    {
        msg = "hydrogram: not enough memory\n";
        goto bad;
    }
    for (i = 0; i < hydrogram->n; ++i)
    {
        if (fscanf(file, "%lf%lf", hydrogram->t + i, hydrogram->Q + i) != 2)
        {
            msg = "hydrogram: bad defined\n";
            goto bad;
        }
    }
    #if DEBUG_MODEL_READ
    printf("hydrogram: t=%lf Q=%lf\n", hydrogram->t[i], hydrogram->Q[i]);
    #endif
    return 1;
bad:
    printf(msg);
    return 0;
}

```

4.1.2.8 double interpolate (double x, double x1, double x2, double y1, double y2)

Function to calculate an interpolation.

Parameters

x	x-coordinate of the interpolation point.
x1	x-coordinate of the first point.
x2	x-coordinate of the second point.
y1	y-coordinate of the first point.
y2	y-coordinate of the second point.

Returns

y-coordinate of the interpolation point.

Definition at line 55 of file [channel.c](#).

```
{
    return y1 + (x - x1) * (y2 - y1) / (x2 - x1);
}
```

4.2 channel.c

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
00015         notice,
00016         this list of conditions and the following disclaimer in the
00017         documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00020 OR
00021 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00023 EVENT
00024 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00025 INDIRECT,
00026 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00027 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00030 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00031 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033 #include <stdio.h>
00034 #include <stdlib.h>
00035 #include "config.h"
00036 #include "channel.h"
00037
00038 double interpolate(double x, double x1, double x2, double y1, double
00039     y2)
00040 {
00041     return y1 + (x - x1) * (y2 - y1) / (x2 - x1);
00042 }
00043
00044 int hydrogram_read(Hydrogram *hydrogram, FILE *file)
00045 {
00046     int i;
00047     char *msg;
00048     if (fscanf(file, "%d", &hydrogram->n) != 1 || hydrogram->n < 1)
00049     {
```

```

00075         msg = "hydrogram: bad points number\n";
00076         goto bad;
00077     }
00078     #if DEBUG_MODEL_READ
00079     printf("hydrogram: n=%d\n", hydrogram->n);
00080     #endif
00081     hydrogram->t = (double*)malloc(hydrogram->n * sizeof(double));
00082     hydrogram->Q = (double*)malloc(hydrogram->n * sizeof(double));
00083     if (!hydrogram->t || !hydrogram->Q)
00084     {
00085         msg = "hydrogram: not enough memory\n";
00086         goto bad;
00087     }
00088     for (i = 0; i < hydrogram->n; ++i)
00089     {
00090         if (fscanf(file, "%lf%lf", hydrogram->t + i, hydrogram->Q + i) != 2)
00091         {
00092             msg = "hydrogram: bad defined\n";
00093             goto bad;
00094         }
00095         #if DEBUG_MODEL_READ
00096         printf("hydrogram: t=%lf Q=%lf\n", hydrogram->t[i], hydrogram->Q[i]);
00097         #endif
00098     }
00099     return 1;
00100
00101 bad:
00102     printf(msg);
00103     return 0;
00104 }
00105
00115 double hydrogram_discharge(Hydrogram *hydrogram, double t)
00116 {
00117     int i, n1;
00118     n1 = hydrogram->n - 1;
00119     if (t <= hydrogram->t[0]) return hydrogram->Q[0];
00120     if (t >= hydrogram->t[n1]) return hydrogram->Q[n1];
00121     for (i = 0; t > hydrogram->t[i];) ++i;
00122     return interpolate(t, hydrogram->t[i], hydrogram->t[i - 1],
00123         hydrogram->Q[i], hydrogram->Q[i - 1]);
00124 }
00125
00137 double hydrogram_integrate(Hydrogram *hydrogram, double t1,
00138     double t2)
00139 {
00140     int i, j, n1;
00141     double Q1, Q2, I;
00142     n1 = hydrogram->n - 1;
00143     if (t2 <= hydrogram->t[0]) return hydrogram->Q[0] * (t2 - t1);
00144     if (t1 >= hydrogram->t[n1]) return hydrogram->Q[n1] * (t2 - t1);
00145     for (i = 0; t1 > hydrogram->t[i];) ++i;
00146     for (j = i; j < hydrogram->n && t2 > hydrogram->t[j];) ++j;
00147     if (i == j)
00148     {
00149         Q1 = interpolate(t1, hydrogram->t[i], hydrogram->t[i - 1],
00150             hydrogram->Q[i], hydrogram->Q[i - 1]);
00151         Q2 = interpolate(t2, hydrogram->t[i], hydrogram->t[i - 1],
00152             hydrogram->Q[i], hydrogram->Q[i - 1]);
00153         return 0.5 * (Q1 + Q2) * (t2 - t1);
00154     }
00155     if (i == 0)
00156     {
00157         Q1 = hydrogram->Q[0];
00158     }
00159     else
00160     {
00161         Q1 = interpolate(t1, hydrogram->t[i], hydrogram->t[i - 1],
00162             hydrogram->Q[i], hydrogram->Q[i - 1]);
00163     }
00164     I = 0.5 * (Q1 + hydrogram->Q[i]) * (hydrogram->t[i] - t1);
00165     while (++i < j)
00166     {
00167         I += 0.5 * (hydrogram->Q[i] - hydrogram->Q[i - 1])
00168             * (hydrogram->t[i] - hydrogram->t[i - 1]);
00169     }
00170     if (i == hydrogram->n)
00171     {
00172         Q2 = hydrogram->Q[n1];
00173     }
00174     else
00175     {
00176         Q2 = interpolate(t2, hydrogram->t[i], hydrogram->t[i - 1],
00177             hydrogram->Q[i], hydrogram->Q[i - 1]);
00178     }
00179     return I + 0.5 * (Q2 + hydrogram->Q[i - 1]) * (t2 - hydrogram->t[i - 1]);
00180 }

```

```

00190 int channel_friction_read_Manning(Channel *channel
, FILE *file)
00191 {
00192     if (fscanf(file, "%lf", channel->friction_coefficient) != 1
00193         || channel->friction_coefficient[0] < 0.)
00194     {
00195         printf("channel friction: bad defined\n");
00196         return 0;;
00197     }
00198 #if DEBUG_MODEL_READ
00199     printf("channel friction: coefficient1=%lf\n",
00200         channel->friction_coefficient[0]);
00201 #endif
00202     return 1;
00203 }
00204
00215 int channel_infiltration_read_KostiakovLewis
(Channel *channel, FILE *file)
00216 {
00217     if (fscanf(file, "%lf%lf%lf%lf",
00218         channel->infiltration_coefficient,
00219         channel->infiltration_coefficient + 1,
00220         channel->infiltration_coefficient + 2,
00221         channel->infiltration_coefficient + 3) != 4
00222         || channel->infiltration_coefficient[0] < 0.
00223         || channel->infiltration_coefficient[1] < 0.
00224         || channel->infiltration_coefficient[3] <= 0.)
00225     {
00226         printf("channel infiltration: bad defined\n");
00227         return 0;;
00228     }
00229 #if DEBUG_MODEL_READ
00230     printf("channel infiltration:\n"
00231         "coefficient1=%lf\n"
00232         "coefficient2=%lf\n"
00233         "coefficient3=%lf\n"
00234         "coefficient4=%lf\n",
00235         channel->infiltration_coefficient[0],
00236         channel->infiltration_coefficient[1],
00237         channel->infiltration_coefficient[2],
00238         channel->infiltration_coefficient[3]);
00239 #endif
00240     return 1;
00241 }
00242
00252 int channel_diffusion_read_Rutherford(Channel
*channel, FILE *file)
00253 {
00254     if (fscanf(file, "%lf", channel->diffusion_coefficient) != 1
00255         || channel->diffusion_coefficient[0] < 0.)
00256     {
00257         printf("channel diffusion: bad defined\n");
00258         return 0;;
00259     }
00260     return 1;
00261 }
00262
00272 int channel_read(Channel *channel, FILE *file)
00273 {
00274     char *msg;
00275     if (fscanf(file, "%lf%lf%lf%lf%lf%d%d%d",
00276         &channel->length,
00277         &channel->slope,
00278         &channel->bottom_width,
00279         &channel->wall_slope,
00280         &channel->height,
00281         &channel->type_outlet,
00282         &channel->friction_model,
00283         &channel->infiltration_model,
00284         &channel->diffusion_model) != 9)
00285     {
00286         msg = "channel: bad defined\n";
00287         goto bad;
00288     }
00289 #if DEBUG_MODEL_READ
00290     printf("channel:\n"
00291         "length=%lf slope=%lf\n"
00292         "bottom_width=%lf wall_slope=%lf\n"
00293         "height=%lf type_outlet=%d\n"
00294         "friction_model=%d infiltration_model=%d diffusion_model=%d\n",
00295         channel->length,
00296         channel->slope,
00297         channel->bottom_width,
00298         channel->wall_slope,
00299         channel->height,
00300         channel->type_outlet,
00301         channel->friction_model,

```

```

00302         channel->infiltration_model,
00303         channel->diffusion_model);
00304 #endif
00305     if (channel->length <= 0.)
00306     {
00307         msg = "channel: bad length\n";
00308         goto bad;
00309     }
00310     if (channel->bottom_width < 0.)
00311     {
00312         msg= "channel: bad bottom width\n";
00313         goto bad;
00314     }
00315     if (channel->wall_slope < 0.)
00316     {
00317         msg = "channel: bad wall slope\n";
00318         goto bad;
00319     }
00320     if (channel->height <= 0.)
00321     {
00322         msg = "channel: bad height\n";
00323         goto bad;
00324     }
00325     switch (channel->type_outlet)
00326     {
00327     case 1:
00328     case 2:
00329         break;
00330     default:
00331         msg = "channel: bad outlet\n";
00332         goto bad;
00333     }
00334     switch (channel->friction_model)
00335     {
00336     case 1:
00337         if (!channel_friction_read_Manning(channel
, file)) return 0;
00338         break;
00339     default:
00340         msg = "channel: bad friction model\n";
00341         goto bad;
00342     }
00343     switch (channel->infiltration_model)
00344     {
00345     case 1:
00346         if (!channel_infiltration_read_KostiakovLewis
(channel, file)) return 0;
00347         break;
00348     default:
00349         msg = "channel: bad infiltration model\n";
00350         goto bad;
00351     }
00352     switch (channel->diffusion_model)
00353     {
00354     case 1:
00355         if (!channel_diffusion_read_Rutherford
(channel, file)) return 0;
00356         break;
00357     default:
00358         msg = "channel: bad diffusion model\n";
00359         goto bad;
00360     }
00361     if (!hydrogram_read(channel->water_inlet, file))
00362     {
00363         msg = "channel: inlet\n";
00364         goto bad;
00365     }
00366     if (!hydrogram_read(channel->solute_inlet, file))
00367     {
00368         msg = "channel: outlet\n";
00369         goto bad;
00370     }
00371     return 1;
00372 bad:
00373     printf(msg);
00374     return 0;
00375 }
00376 }
00377

```

4.3 channel.h File Reference

Header file to define a channel.

Data Structures

- struct [Hydrogram](#)
Struct to define a hydrogram.
- struct [Channel](#)
Struct to define a channel.

Functions

- double [interpolate](#) (double x, double x1, double x2, double y1, double y2)
Function to calculate an interpolation.
- int [hydrogram_read](#) (Hydrogram *hydrogram, FILE *file)
Function to read the data of a hydrogram.
- double [hydrogram_discharge](#) (Hydrogram *hydrogram, double t)
Function to calculate the discharge in a hydrogram.
- double [hydrogram_integrate](#) (Hydrogram *hydrogram, double t1, double t2)
Function to integrate the mass flux in a hydrogram.
- int [channel_friction_read_Manning](#) (Channel *channel, FILE *file)
Function to read the friction coefficient of the Manning model.
- int [channel_infiltration_read_KostiakovLewis](#) (Channel *channel, FILE *file)
function to read the infiltration coefficients of the Kostiakov-Lewis model.
- int [channel_diffusion_read_Rutherford](#) (Channel *channel, FILE *file)
Function to read the diffusion coefficient of the Rutherford model.
- int [channel_read](#) (Channel *channel, FILE *file)
function to read a channel.

4.3.1 Detailed Description

Header file to define a channel.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [channel.h](#).

4.3.2 Function Documentation

4.3.2.1 int channel_diffusion_read_Rutherford (Channel * *channel*, FILE * *file*)

Function to read the diffusion coefficient of the Rutherford model.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line 252 of file [channel.c](#).

```
{
    if (fscanf(file, "%lf", channel->diffusion_coefficient) != 1
        || channel->diffusion_coefficient[0] < 0.)
    {
        printf("channel diffusion: bad defined\n");
        return 0;;
    }
    return 1;
}
```

4.3.2.2 int channel_friction_read_Manning (Channel * *channel*, FILE * *file*)

Function to read the friction coefficient of the Manning model.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line 190 of file [channel.c](#).

```
{
    if (fscanf(file, "%lf", channel->friction_coefficient) != 1
        || channel->friction_coefficient[0] < 0.)
    {
        printf("channel friction: bad defined\n");
        return 0;;
    }
    #if DEBUG_MODEL_READ
    printf("channel friction: coefficient1=%lf\n",
        channel->friction_coefficient[0]);
    #endif
    return 1;
}
```

4.3.2.3 int channel_infiltration_read_KostiakovLewis (Channel * *channel*, FILE * *file*)

function to read the infiltration coefficients of the Kostiakov-Lewis model.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line 215 of file [channel.c](#).

```
{
    if (fscanf(file, "%lf%lf%lf%lf",
        channel->infiltration_coefficient,
```

```

        channel->infiltration_coefficient + 1,
        channel->infiltration_coefficient + 2,
        channel->infiltration_coefficient + 3) != 4
        || channel->infiltration_coefficient[0] < 0.
        || channel->infiltration_coefficient[1] < 0.
        || channel->infiltration_coefficient[3] <= 0.)
    {
        printf("channel infiltration: bad defined\n");
        return 0;;
    }
#ifdef DEBUG_MODEL_READ
    printf("channel infiltration:\n"
        "coefficient1=%lf\n"
        "coefficient2=%lf\n"
        "coefficient3=%lf\n"
        "coefficient4=%lf\n",
        channel->infiltration_coefficient[0],
        channel->infiltration_coefficient[1],
        channel->infiltration_coefficient[2],
        channel->infiltration_coefficient[3]);
#endif
    return 1;
}

```

4.3.2.4 int channel_read (Channel * *channel*, FILE * *file*)

function to read a channel.

Parameters

<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success

Definition at line 272 of file [channel.c](#).

```

{
    char *msg;
    if (fscanf(file, "%lf%lf%lf%lf%lf%d%d%d",
        &channel->length,
        &channel->slope,
        &channel->bottom_width,
        &channel->wall_slope,
        &channel->height,
        &channel->type_outlet,
        &channel->friction_model,
        &channel->infiltration_model,
        &channel->diffusion_model) != 9)
    {
        msg = "channel: bad defined\n";
        goto bad;
    }
#ifdef DEBUG_MODEL_READ
    printf("channel:\n"
        "length=%lf slope=%lf\n"
        "bottom_width=%lf wall_slope=%lf\n"
        "height=%lf type_outlet=%d\n"
        "friction_model=%d infiltration_model=%d diffusion_model=%d\n",
        channel->length,
        channel->slope,
        channel->bottom_width,
        channel->wall_slope,
        channel->height,
        channel->type_outlet,
        channel->friction_model,
        channel->infiltration_model,
        channel->diffusion_model);
#endif
    if (channel->length <= 0.)
    {
        msg = "channel: bad length\n";
        goto bad;
    }
    if (channel->bottom_width < 0.)

```

```

    {
        msg= "channel: bad bottom width\n";
        goto bad;
    }
    if (channel->wall_slope < 0.)
    {
        msg = "channel: bad wall slope\n";
        goto bad;
    }
    if (channel->height <= 0.)
    {
        msg = "channel: bad height\n";
        goto bad;
    }
    switch (channel->type_outlet)
    {
    case 1:
    case 2:
        break;
    default:
        msg = "channel: bad outlet\n";
        goto bad;
    }
    switch (channel->friction_model)
    {
    case 1:
        if (!channel_friction_read_Manning(channel
        , file)) return 0;
        break;
    default:
        msg = "channel: bad friction model\n";
        goto bad;
    }
    switch (channel->infiltration_model)
    {
    case 1:
        if (!channel_infiltration_read_KostiakovLewis
        (channel, file)) return 0;
        break;
    default:
        msg = "channel: bad infiltration model\n";
        goto bad;
    }
    switch (channel->diffusion_model)
    {
    case 1:
        if (!channel_diffusion_read_Rutherford
        (channel, file)) return 0;
        break;
    default:
        msg = "channel: bad diffusion model\n";
        goto bad;
    }
    if (!hydrogram_read(channel->water_inlet, file))
    {
        msg = "channel: inlet\n";
        goto bad;
    }
    if (!hydrogram_read(channel->solute_inlet, file))
    {
        msg = "channel: outlet\n";
        goto bad;
    }
    return 1;
bad:
    printf(msg);
    return 0;
}

```

4.3.2.5 double hydrogram_discharge (Hydrogram * hydrogram, double t)

Function to calculate the discharge in a hydrogram.

Parameters

<i>hydrogram</i>	hydrogram struct.
<i>t</i>	time.

Returns

discharge.

Definition at line 115 of file [channel.c](#).

```
{
    int i, n1;
    n1 = hydrogram->n - 1;
    if (t <= hydrogram->t[0]) return hydrogram->Q[0];
    if (t >= hydrogram->t[n1]) return hydrogram->Q[n1];
    for (i = 0; t > hydrogram->t[i];) ++i;
    return interpolate(t, hydrogram->t[i], hydrogram->t[i - 1],
        hydrogram->Q[i], hydrogram->Q[i - 1]);
}
```

4.3.2.6 double hydrogram.integrate (Hydrogram * *hydrogram*, double *t1*, double *t2*)

Function to integrate the mass flux in a hydrogram.

Parameters

<i>hydrogram</i>	hydrogram struct.
<i>t1</i>	initial time.
<i>t2</i>	final time.

Returns

integral of the mass flux.

Definition at line 137 of file [channel.c](#).

```
{
    int i, j, n1;
    double Q1, Q2, I;
    n1 = hydrogram->n - 1;
    if (t2 <= hydrogram->t[0]) return hydrogram->Q[0] * (t2 - t1);
    if (t1 >= hydrogram->t[n1]) return hydrogram->Q[n1] * (t2 - t1);
    for (i = 0; t1 > hydrogram->t[i];) ++i;
    for (j = i; j < hydrogram->n && t2 > hydrogram->t[j];) ++j;
    if (i == j)
    {
        Q1 = interpolate(t1, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
        Q2 = interpolate(t2, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
        return 0.5 * (Q1 + Q2) * (t2 - t1);
    }
    if (i == 0)
    {
        Q1 = hydrogram->Q[0];
    }
    else
    {
        Q1 = interpolate(t1, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
    }
    I = 0.5 * (Q1 + hydrogram->Q[i]) * (hydrogram->t[i] - t1);
    while (++i < j)
    {
        I += 0.5 * (hydrogram->Q[i] - hydrogram->Q[i - 1])
            * (hydrogram->t[i] - hydrogram->t[i - 1]);
    }
    if (i == hydrogram->n)
    {
        Q2 = hydrogram->Q[n1];
    }
    else
    {
        Q2 = interpolate(t2, hydrogram->t[i], hydrogram->t[i - 1],
            hydrogram->Q[i], hydrogram->Q[i - 1]);
    }
    return I + 0.5 * (Q2 + hydrogram->Q[i - 1]) * (t2 - hydrogram->t[i - 1]);
}
```

4.3.2.7 int hydrogram_read (Hydrogram * *hydrogram*, FILE * *file*)

Function to read the data of a hydrogram.

Parameters

<i>hydrogram</i>	hydrogram struct.
<i>file</i>	input file.

Returns

0 on error, 1 on success.

Definition at line 69 of file [channel.c](#).

```
{
    int i;
    char *msg;
    if (fscanf(file, "%d", &hydrogram->n) != 1 || hydrogram->n < 1)
    {
        msg = "hydrogram: bad points number\n";
        goto bad;
    }
    #if DEBUG_MODEL_READ
    printf("hydrogram: n=%d\n", hydrogram->n);
    #endif
    hydrogram->t = (double*)malloc(hydrogram->n * sizeof(double));
    hydrogram->Q = (double*)malloc(hydrogram->n * sizeof(double));
    if (!hydrogram->t || !hydrogram->Q)
    {
        msg = "hydrogram: not enough memory\n";
        goto bad;
    }
    for (i = 0; i < hydrogram->n; ++i)
    {
        if (fscanf(file, "%lf%lf", hydrogram->t + i, hydrogram->Q + i) != 2)
        {
            msg = "hydrogram: bad defined\n";
            goto bad;
        }
    }
    #if DEBUG_MODEL_READ
    printf("hydrogram: t=%lf Q=%lf\n", hydrogram->t[i], hydrogram->Q[i]);
    #endif
    return 1;
bad:
    printf(msg);
    return 0;
}
```

4.3.2.8 double interpolate (double *x*, double *x1*, double *x2*, double *y1*, double *y2*)

Function to calculate an interpolation.

Parameters

<i>x</i>	x-coordinate of the interpolation point.
<i>x1</i>	x-coordinate of the first point.
<i>x2</i>	x-coordinate of the second point.
<i>y1</i>	y-coordinate of the first point.
<i>y2</i>	y-coordinate of the second point.

Returns

y-coordinate of the interpolation point.

Definition at line 55 of file [channel.c](#).

```
{
    return y1 + (x - x1) * (y2 - y1) / (x2 - x1);
}
```

4.4 channel.h

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 // in order to prevent multiple definitions
00033 #ifndef CHANNEL_H
00034 #define CHANNEL_H 1
00035
00036 struct _Hydrogram
00037 {
00038     double *t, *Q;
00039     int n;
00040 };
00041
00042 typedef struct _Hydrogram Hydrogram;
00043
00044 struct _Channel
00045 {
00046     Hydrogram water_inlet[1], solute_inlet[1];
00047     double friction_coefficient[3],
00048     infiltration_coefficient[4],
00049     diffusion_coefficient[1], slope, length
00050     , bottom_width, wall_slope,
00051     height;
00052     int type_outlet, friction_model,
00053     infiltration_model, diffusion_model;
00054 };
00055
00056 typedef struct _Channel Channel;
00057
00058 // member functions
00059
00060 double interpolate(double x, double x1, double x2, double y1, double
00061     y2);
00062
00063 int hydrogram_read(Hydrogram *hydrogram, FILE *file);
00064 double hydrogram_discharge(Hydrogram *hydrogram, double t);
00065 double hydrogram_integrate(Hydrogram *hydrogram, double t1,
00066     double t2);
00067
00068 int channel_friction_read_Manning(Channel *channel
00069     , FILE *file);
00070 int channel_infiltration_read_KostiakovLewis
00071     (Channel *channel, FILE *file);
00072 int channel_diffusion_read_Rutherford(Channel
00073     *channel, FILE *file);
00074 int channel_read(Channel *channel, FILE *file);
00075
```

```
00124 #endif
```

4.5 config.h File Reference

Configuration header file.

Macros

- `#define G 9.81`
Gravitational constant.
- `#define DEBUG_MODEL_READ 0`
Macro to debug the function `model_read()`.
- `#define DEBUG_MESH_OPEN 0`
Macro to debug the function `mesh_open()`.

4.5.1 Detailed Description

Configuration header file.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [config.h](#).

4.6 config.h

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018 OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021 EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023 INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
```



```
00028
00036 // in order to prevent multiple definitions
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00044 #define G 9.81
00045
00046 // debug defines
00047
00052 #define DEBUG_MODEL_READ    0
00053
00057 #define DEBUG_MESH_OPEN    0
00058
00059 #endif
```

4.7 main.c File Reference

Main source code.

```
#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_complete.h"
#include "model_zero_inertia.h"
#include "model_diffusive.h"
#include "model_kinematic.h"
#include "model_complete_upwind.h"
#include "model_zero_inertia_upwind.h"
#include "model_diffusive_upwind.h"
#include "model_kinematic_upwind.h"
#include "model_complete_LaxFriedrichs.h"
#include "model_zero_inertia_LaxFriedrichs.h"
```

Functions

- int [main](#) (int argn, char **argc)

Main function.

Variables

- double [critical_depth_tolerance](#) = 0.001

Accuracy calculating the critical depth.

4.7.1 Detailed Description

Main source code.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [main.c](#).

4.8 main.c

```

00001  /*
00002  SWOCS: a software to check the numerical performance of different models in
00003  channel or furrow flows
00004
00005  Copyright 2011, Javier Burguete Tolosa.
00006
00007  Redistribution and use in source and binary forms, with or without
00008  modification,
00009  are permitted provided that the following conditions are met:
00010      1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012      2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017  THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018  OR
00019  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021  EVENT
00022  SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023  INDIRECT,
00024  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028  OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029  ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030  */
00031
00032 #include <stdio.h>
00033 #include <math.h>
00034 #include "config.h"
00035 #include "channel.h"
00036 #include "node.h"
00037 #include "mesh.h"
00038 #include "model.h"
00039 #include "model_complete.h"
00040 #include "model_zero_inertia.h"
00041 #include "model_diffusive.h"
00042 #include "model_kinematic.h"
00043 #include "model_complete_upwind.h"
00044 #include "model_zero_inertia_upwind.h"
00045 #include "model_diffusive_upwind.h"
00046 #include "model_kinematic_upwind.h"
00047 #include "model_complete_LaxFriedrichs.h"
00048 #include "model_zero_inertia_LaxFriedrichs.h"
00049 // #include "model_diffusive_LaxFriedrichs.h"
00050 // #include "model_kinematic_LaxFriedrichs.h"
00051
00052 double critical_depth_tolerance = 0.001;
00053
00054 int main(int argn, char **argc)
00055 {
00056     int i;
00057     FILE *file, *file_advance, *file_probes;
00058     Model model[1];
00059     if (argn < 3 || argn == 6 || argn > 7)
00060     {
00061         printf("the sintaxis is:\n./SWOCS input_file "
00062             "output_variables_file "
00063             "[output_flows_file] [output_advance_file]"
00064             "[input_probes_file output_probes_file]\n");
00065         return 1;
00066     }
00067     if (!model_read(model, argc[1])) return 2;
00068     switch (model->type_model)
00069     {
00070     case 1:

```

```

00084     model->model_node_parameters_centre
00085         = model->model_node_parameters_right
00086         = model->model_node_parameters_left
00087         = model_node_parameters_complete;
00088     model->node_ldt_max = node_ldt_max_complete;
00089     model->node_flows = node_flows_complete;
00090     model->model_inlet_dtmax = model_inlet_dtmax_complete
00091 ;
00092     goto complete;
00093 case 2:
00094     model->model_node_parameters_centre
00095         = model->model_node_parameters_right
00096         = model->model_node_parameters_left
00097         = model_node_parameters_zero_inertia
00098 ;
00099     model->node_ldt_max = node_ldt_max_zero_inertia
00100 ;
00101     model->node_flows = node_flows_zero_inertia;
00102     model->model_inlet_dtmax = model_inlet_dtmax_zero_inertia
00103 ;
00104     goto zero_inertia;
00105 case 3:
00106     switch (model->channel->friction_model)
00107     {
00108     case 1:
00109         model->node_discharge_centre
00110             = node_discharge_centre_diffusive_Manning
00111 ;
00112         model->node_discharge_right
00113             = node_discharge_right_diffusive_Manning
00114 ;
00115         model->node_discharge_left
00116             = node_discharge_left_diffusive_Manning
00117 ;
00118         model->model_node_parameters_centre
00119             = model_node_parameters_centre_diffusive
00120 ;
00121         model->model_node_parameters_right
00122             = model_node_parameters_right_diffusive
00123 ;
00124         model->model_node_parameters_left
00125             = model_node_parameters_left_diffusive
00126 ;
00127     }
00128     model->node_ldt_max = node_ldt_max_diffusive;
00129     model->node_flows = node_flows_diffusive;
00130     model->model_inlet_dtmax = model_inlet_dtmax_diffusive
00131 ;
00132     goto diffusive;
00133 case 4:
00134     switch (model->channel->friction_model)
00135     {
00136     case 1:
00137         model->node_discharge_centre
00138             = node_discharge_centre_kinematic_Manning
00139 ;
00140         model->node_discharge_right
00141             = node_discharge_right_kinematic_Manning
00142 ;
00143         model->node_discharge_left
00144             = node_discharge_left_kinematic_Manning
00145 ;
00146         model->model_node_parameters_centre
00147             = model_node_parameters_centre_kinematic
00148 ;
00149         model->model_node_parameters_right
00150             = model_node_parameters_right_kinematic
00151 ;
00152         model->model_node_parameters_left
00153             = model_node_parameters_left_kinematic
00154 ;
00155     }
00156     model->node_ldt_max = node_ldt_max_kinematic;
00157     model->node_flows = node_flows_kinematic;
00158     model->model_inlet_dtmax = model_inlet_dtmax_kinematic
00159 ;
00160     goto kinematic;
00161 default:
00162     printf("model: bad type\n");
00163     return 2;
00164 }
00165 complete:
00166 switch (model->type_surface_flow)
00167 {
00168 case 1:
00169     model->model_surface_flow = model_surface_flow_complete_upwind;

```

```

00153         goto calculate;
00154     case 2:
00155         model->model_surface_flow = model_surface_flow_complete_LaxFriedrichs
;
00156         goto calculate;
00157     default:
00158         printf("model: bad surface flow type\n");
00159         return 2;
00160     }
00161
00162 zero_inertia:
00163     switch (model->type_surface_flow)
00164     {
00165     case 1:
00166         model->model_surface_flow = model_surface_flow_zero_inertia_upwind;
00167         goto calculate;
00168     case 2:
00169         model->model_surface_flow =
00170             model_surface_flow_zero_inertia_LaxFriedrichs
;
00171         goto calculate;
00172     default:
00173         printf("model: bad surface flow type\n");
00174         return 2;
00175     }
00176
00177 diffusive:
00178     switch (model->type_surface_flow)
00179     {
00180     case 1:
00181         model->model_surface_flow = model_surface_flow_diffusive_upwind
;
00182         break;
00183     // case 2:
00184     //     model->model_surface_flow = model_surface_flow_diffusive_LaxFriedrichs;
00185     //     break;
00186     default:
00187         printf("model: bad surface flow type\n");
00188         return 2;
00189     }
00190
00191 kinematic:
00192     switch (model->type_surface_flow)
00193     {
00194     case 1:
00195         model->model_surface_flow = model_surface_flow_kinematic_upwind
;
00196         break;
00197     // case 2:
00198     //     model->model_surface_flow = model_surface_flow_kinematic_LaxFriedrichs;
00199     //     break;
00200     default:
00201         printf("model: bad surface flow type\n");
00202         return 2;
00203     }
00204
00205 calculate:
00206     switch (model->type_diffusion)
00207     {
00208     case 1:
00209         model->model_diffusion = model_diffusion_explicit
;
00210         break;
00211     case 2:
00212         model->model_diffusion = model_diffusion_implicit
;
00213         break;
00214     default:
00215         printf("model: bad diffusion type\n");
00216         return 2;
00217     }
00218
00219 if (argn > 4)
00220 {
00221     // opening the advance file
00222     file_advance = fopen(argc[4], "w");
00223
00224     if (argn > 6)
00225     {
00226         // opening the probes files
00227         if (!model_probes_read(model, argc[5])) return 2;
00228         file_probes = fopen(argc[6], "w");
00229         if (!file_probes)
00230         {
00231             printf("model: unable to open the probes output file\n");
00232             return 2;
00233         }
00234     }
}

```

```

00234     }
00235 }
00236
00237 // init model parameters
00238 model_parameters(model);
00239
00240 // main calculation bucle
00241 for (model->t = 0, i = 0; model->t < model->tfinal; ++i)
00242 {
00243     if (argn > 4)
00244     {
00245         // writing the advance
00246         model_write_advance(model, file_advance);
00247
00248         // writing the probes
00249         if (argn > 6) model_write_probes(model,
file_probes);
00250     }
00251
00252     // model step
00253     model_step(model);
00254 //     model_print(model, i);
00255 }
00256 model_print(model, i);
00257
00258 // writing result variables
00259 file = fopen(argc[2], "w");
00260 mesh_write_variables(model->mesh, file);
00261 fclose(file);
00262
00263 // writing result flows
00264 if (argn > 3)
00265 {
00266     file = fopen(argc[3], "w");
00267     mesh_write_flows(model->mesh, file);
00268     fclose(file);
00269
00270     if (argn > 4)
00271     {
00272         // closing the advance
00273         model_write_advance(model, file_advance);
00274         fclose(file_advance);
00275
00276         if (argn > 6)
00277         {
00278             // closing the probes
00279             model_write_probes(model, file_probes);
00280             fclose(file_probes);
00281         }
00282     }
00283 }
00284
00285 return 0;
00286 }

```

4.9 mesh.c File Reference

Source file to define a mesh.

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"

```

Functions

- int [mesh_open](#) (Mesh *mesh, Channel *channel)
Function to open a mesh.
- void [mesh_initial_conditions_dry](#) (Mesh *mesh)

Function to read dry initial conditions.

- int [mesh_initial_conditions_profile](#) (Mesh *mesh, FILE *file)

Function to read an initial conditions profile.

- int [mesh_read](#) (Mesh *mesh, Channel *channel, FILE *file)

Function to read a mesh.

- void [mesh_write_variables](#) (Mesh *mesh, FILE *file)

Function to write the variables of a mesh in a file.

- void [mesh_write_flows](#) (Mesh *mesh, FILE *file)

Function to write the flows of a mesh in a file.

- double [mesh_water_mass](#) (Mesh *mesh)

Function to calculate the water mass in a mesh.

- double [mesh_solute_mass](#) (Mesh *mesh)

Function to calculate the solute mass in a mesh.

4.9.1 Detailed Description

Source file to define a mesh.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [mesh.c](#).

4.9.2 Function Documentation

4.9.2.1 void mesh_initial_conditions_dry (Mesh * mesh)

Function to read dry initial conditions.

Parameters

<i>mesh</i>	mesh struct.
-------------	--------------

Definition at line 101 of file [mesh.c](#).

```
{
    int i;
    Node *node = mesh->node;
    for (i = 0; i < mesh->n; ++i)
        node[i].A = node[i].Q = node[i].As = node[i].Ai = node[i].Asi
        = 0.;
}
```

4.9.2.2 int mesh_initial_conditions_profile (Mesh * mesh, FILE * file)

Function to read an initial conditions profile.

Parameters

<i>mesh</i>	mesh struct.
<i>file</i>	input file.

Definition at line 117 of file [mesh.c](#).

```
{
    int i, j, n;
    double dx, *x, *A, *Q, *s;
    char *msg;
    Node *node = mesh->node;
    if (fscanf(file, "%d", &n) != 1 || n < 1)
    {
        msg = "mesh initial conditions profile: bad points number\n";
        goto bad2;
    }
    x = (double*)malloc(n * sizeof(double));
    A = (double*)malloc(n * sizeof(double));
    Q = (double*)malloc(n * sizeof(double));
    s = (double*)malloc(n * sizeof(double));
    if (!x || !A || !Q || !s)
    {
        msg = "mesh initial conditions profile: not enough memory\n";
        goto bad;
    }
    for (i = 0; i < n; ++i)
    {
        if (fscanf(file, "%lf%lf%lf%lf", x + i, A + i, Q + i, s + i) != 4 ||
            A[i] < 0. || s[i] < 0.)
        {
            msg = "mesh initial conditions profile: bad defined\n";
            goto bad;
        }
        if (i > 0 && x[i] < x[i - 1])
        {
            msg = "mesh initial conditions profile: bad order\n";
            goto bad;
        }
    }
    --n;
    for (i = j = 0; i < mesh->n; ++i)
    {
        while (node[i].x > x[j])
        {
            if (j < n) ++j; else break;
        }
        if (node[i].x <= x[0])
        {
            node[i].A = A[0];
            node[i].Q = Q[0];
            node[i].As = A[0] * s[0];
        }
        else if (node[i].x >= x[n])
        {
            node[i].A = A[n];
            node[i].Q = Q[n];
            node[i].As = A[n] * s[n];
        }
        else
        {
            dx = (node[i].x - x[j]) / (x[j + 1] - x[j]);
            node[i].A = A[j] + dx * (A[j+1] - A[j]);
            node[i].Q = Q[j] + dx * (Q[j+1] - Q[j]);
            node[i].As = node[i].A * (s[j] + dx * (s[j+1] - s[j]));
        }
        node[i].Ai = node[i].Asi = 0.;
    }
    return 1;

bad:
    free(x), free(A), free(Q), free(s);

bad2:
    printf(msg);
    return 0;
}
```

4.9.2.3 int mesh_open (Mesh * *mesh*, Channel * *channel*)

Function to open a mesh.

Parameters

<i>mesh</i>	mesh struct.
<i>channel</i>	channel struct.

Returns

0 on error, 1 on success.

Definition at line 53 of file [mesh.c](#).

```
{
    int i;
    double ix, Z;
    Node *node;
    mesh->node = node = (Node*)malloc(mesh->n * sizeof(Node));
    if (!mesh->node)
    {
        printf("mesh: not enough memory\n");
        return 0;
    }
    ix = channel->length / (mesh->n - 1);
    Z = channel->wall_slope;
    for (i = 0; i < mesh->n; ++i)
    {
        node[i].ix = ix;
        node[i].x = i * ix;
        node[i].zb = (channel->length - node[i].x) * channel->slope;
        node[i].B0 = channel->bottom_width;
        node[i].Z = Z;
        memcpy(node[i].friction_coefficient, channel->
            friction_coefficient,
            3 * sizeof(double));
        memcpy(node[i].infiltration_coefficient,
            channel->infiltration_coefficient, 4 * sizeof(double));
        memcpy(node[i].diffusion_coefficient, channel->
            diffusion_coefficient,
            sizeof(double));
    }
    node[0].dx = node[mesh->n - 1].dx = 0.5 * ix;
    for (i = 0; ++i < mesh->n - 1;) node[i].dx = ix;
    #if DEBUG_MESH_OPEN
    for (i=0; i < mesh->n; ++i)
        printf("node:\nx=%lf ix=%lf dx=%lf\nzb=%lf B0=%lf Z=%lf\n",
            node[i].x,
            node[i].ix,
            node[i].dx,
            node[i].zb,
            node[i].B0,
            node[i].Z);
    #endif
    return 1;
}
```

4.9.2.4 int mesh_read (Mesh * *mesh*, Channel * *channel*, FILE * *file*)

Function to read a mesh.

Parameters

<i>mesh</i>	mesh struct.
<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on succes.

Definition at line 200 of file [mesh.c](#).

```
{
    char *msg;
    if (fscanf(file, "%d%d", &mesh->n, &mesh->type) != 2)
    {
        msg = "mesh: bad defined\n";
        goto bad;
    }
    if (mesh->n < 3)
    {
```



```

        msg = "mesh: bad nodes number\n";
        goto bad;
    }
#ifdef DEBUG_MODEL_READ
    printf("mesh: n=%d type=%d\n", mesh->n, mesh->type);
#endif
    if (!mesh_open(mesh, channel)) return 0;
    switch (mesh->type)
    {
        case 1:
            mesh_initial_conditions_dry(mesh);
            break;
        case 2:
            if (!mesh_initial_conditions_profile(
                mesh, file)) return 0;
            break;
        default:
            msg = "mesh: bad type\n";
            goto bad;
    }
    return 1;
bad:
    printf(msg);
    return 0;
}

```

4.9.2.5 double mesh_solute_mass (Mesh * *mesh*)

Function to calculate the solute mass in a mesh.

Parameters

<i>mesh</i>	mesh struct.
-------------	--------------

Returns

solute mass.

Definition at line 315 of file [mesh.c](#).

```

{
    int i;
    double mass = 0.;
    Node *node = mesh->node;
    for (i = 0; i < mesh->n; ++i)
        mass += node[i].dx * (node[i].As + node[i].Asi);
    return mass;
}

```

4.9.2.6 double mesh_water_mass (Mesh * *mesh*)

Function to calculate the water mass in a mesh.

Parameters

<i>mesh</i>	mesh struct.
-------------	--------------

Returns

water mass

Definition at line 297 of file [mesh.c](#).

```

{
    int i;
    double mass = 0.;

```

```

Node *node = mesh->node;
for (i = 0; i < mesh->n; ++i)
    mass += node[i].dx * (node[i].A + node[i].Ai);
for (i=0; i<mesh->n; ++i) printf("i=%d A=%.14le Ai=%.14le\n", i, node[i].A,
    node[i].Ai);
return mass;
}

```

4.9.2.7 int mesh_write_flows (Mesh * *mesh*, FILE * *file*)

Function to write the flows of a mesh in a file.

Parameters

<i>mesh</i>	mesh struct.
<i>file</i>	input file.

Definition at line 270 of file [mesh.c](#).

```

{
    int i, n1;
    Node *node = mesh->node;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        fprintf(file, "%.14le %.14le %.14le %.14le %.14le\n",
            0.5 * (node[i].x + node[i + 1].x),
            (node[i + 1].Q * node[i + 1].u - node[i].Q * node[i].u)
            / node[i].ix,
            0.5 * G * (node[i + 1].A + node[i].A)
            * (node[i + 1].zb - node[i].zb) / node[i].ix,
            0.5 * G * (node[i + 1].A + node[i].A)
            * (node[i + 1].h - node[i].h) / node[i].ix,
            0.25 * G * (node[i + 1].A + node[i].A)
            * (node[i + 1].Sf + node[i].Sf));
    }
}

```

4.9.2.8 int mesh_write_variables (Mesh * *mesh*, FILE * *file*)

Function to write the variables of a mesh in a file.

Parameters

<i>mesh</i>	mesh struct.
<i>file</i>	input file.

Definition at line 244 of file [mesh.c](#).

```

{
    int i;
    Node *node;
    for (i = 0; i < mesh->n; ++i)
    {
        node = mesh->node + i;
        printf("i=%d A=%.14le\n", i, node->A);
        fprintf(file, "%.14le %.14le %.14le %.14le %.14le %.14le\n",
            node->x,
            node->A,
            node->Q,
            node->As,
            node->Ai,
            node->Asi);
    }
}

```

4.10 mesh.c

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
00015         notice,
00016         this list of conditions and the following disclaimer in the
00017         documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00020 OR
00021 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00023 EVENT
00024 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00025 INDIRECT,
00026 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00027 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00030 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00031 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033 #include <stdio.h>
00034 #include <stdlib.h>
00035 #include <math.h>
00036 #include <string.h>
00037 #include "config.h"
00038 #include "channel.h"
00039 #include "node.h"
00040 #include "mesh.h"
00041
00042 int mesh_open(Mesh *mesh, Channel *channel)
00043 {
00044     int i;
00045     double ix, Z;
00046     Node *node;
00047     mesh->node = node = (Node*)malloc(mesh->n * sizeof(Node));
00048     if (!mesh->node)
00049     {
00050         printf("mesh: not enough memory\n");
00051         return 0;
00052     }
00053     ix = channel->length / (mesh->n - 1);
00054     Z = channel->wall_slope;
00055     for (i = 0; i < mesh->n; ++i)
00056     {
00057         node[i].ix = ix;
00058         node[i].x = i * ix;
00059         node[i].zb = (channel->length - node[i].x) * channel->slope;
00060         node[i].B0 = channel->bottom_width;
00061         node[i].Z = Z;
00062         memcpy(node[i].friction_coefficient, channel->
00063             friction_coefficient,
00064             3 * sizeof(double));
00065         memcpy(node[i].infiltration_coefficient,
00066             channel->infiltration_coefficient, 4 * sizeof(double));
00067         memcpy(node[i].diffusion_coefficient, channel->
00068             diffusion_coefficient,
00069             sizeof(double));
00070     }
00071     node[0].dx = node[mesh->n - 1].dx = 0.5 * ix;
00072     for (i = 0; ++i < mesh->n - 1;) node[i].dx = ix;
00073 #if DEBUG_MESH_OPEN
00074     for (i=0; i < mesh->n; ++i)
00075         printf("node: nx=%lf ix=%lf dx=%lf nzb=%lf B0=%lf Z=%lf\n",
00076             node[i].x,
00077             node[i].ix,
00078             node[i].dx,
00079             node[i].zb,
00080             node[i].B0,
00081             node[i].Z);
00082 #endif
00083     return 1;
00084 }

```

```

00093 }
00094
00101 void mesh_initial_conditions_dry(Mesh *mesh)
00102 {
00103     int i;
00104     Node *node = mesh->node;
00105     for (i = 0; i < mesh->n; ++i)
00106         node[i].A = node[i].Q = node[i].As = node[i].Ai = node[i].Asi = 0.;
00107 }
00108
00117 int mesh_initial_conditions_profile(Mesh *mesh,
FILE *file)
00118 {
00119     int i, j, n;
00120     double dx, *x, *A, *Q, *s;
00121     char *msg;
00122     Node *node = mesh->node;
00123     if (fscanf(file, "%d", &n) != 1 || n < 1)
00124     {
00125         msg = "mesh initial conditions profile: bad points number\n";
00126         goto bad2;
00127     }
00128     x = (double*)malloc(n * sizeof(double));
00129     A = (double*)malloc(n * sizeof(double));
00130     Q = (double*)malloc(n * sizeof(double));
00131     s = (double*)malloc(n * sizeof(double));
00132     if (!x || !A || !Q || !s)
00133     {
00134         msg = "mesh initial conditions profile: not enough memory\n";
00135         goto bad;
00136     }
00137     for (i = 0; i < n; ++i)
00138     {
00139         if (fscanf(file, "%lf%lf%lf%lf", x + i, A + i, Q + i, s + i) != 4 ||
00140             A[i] < 0. || s[i] < 0.)
00141         {
00142             msg = "mesh initial conditions profile: bad defined\n";
00143             goto bad;
00144         }
00145         if (i > 0 && x[i] < x[i - 1])
00146         {
00147             msg = "mesh initial conditions profile: bad order\n";
00148             goto bad;
00149         }
00150     }
00151     --n;
00152     for (i = j = 0; i < mesh->n; ++i)
00153     {
00154         while (node[i].x > x[j])
00155         {
00156             if (j < n) ++j; else break;
00157         }
00158         if (node[i].x <= x[0])
00159         {
00160             node[i].A = A[0];
00161             node[i].Q = Q[0];
00162             node[i].As = A[0] * s[0];
00163         }
00164         else if (node[i].x >= x[n])
00165         {
00166             node[i].A = A[n];
00167             node[i].Q = Q[n];
00168             node[i].As = A[n] * s[n];
00169         }
00170         else
00171         {
00172             dx = (node[i].x - x[j]) / (x[j + 1] - x[j]);
00173             node[i].A = A[j] + dx * (A[j+1] - A[j]);
00174             node[i].Q = Q[j] + dx * (Q[j+1] - Q[j]);
00175             node[i].As = node[i].A * (s[j] + dx * (s[j+1] - s[j]));
00176         }
00177         node[i].Ai = node[i].Asi = 0.;
00178     }
00179     return 1;
00180
00181 bad:
00182     free(x), free(A), free(Q), free(s);
00183
00184 bad2:
00185     printf(msg);
00186     return 0;
00187 }
00188
00200 int mesh_read(Mesh *mesh, Channel *channel, FILE *file)
00201 {
00202     char *msg;
00203     if (fscanf(file, "%d%d", &mesh->n, &mesh->type) != 2)

```

```

00204     {
00205         msg = "mesh: bad defined\n";
00206         goto bad;
00207     }
00208     if (mesh->n < 3)
00209     {
00210         msg = "mesh: bad nodes number\n";
00211         goto bad;
00212     }
00213 #if DEBUG_MODEL_READ
00214     printf("mesh: n=%d type=%d\n", mesh->n, mesh->type);
00215 #endif
00216     if (!mesh_open(mesh, channel)) return 0;
00217     switch (mesh->type)
00218     {
00219     case 1:
00220         mesh_initial_conditions_dry(mesh);
00221         break;
00222     case 2:
00223         if (!mesh_initial_conditions_profile(
00224             mesh, file)) return 0;
00225         break;
00226     default:
00227         msg = "mesh: bad type\n";
00228         goto bad;
00229     }
00230     return 1;
00231 bad:
00232     printf(msg);
00233     return 0;
00234 }
00235
00244 void mesh_write_variables(Mesh *mesh, FILE *file)
00245 {
00246     int i;
00247     Node *node;
00248     for (i = 0; i < mesh->n; ++i)
00249     {
00250         node = mesh->node + i;
00251         printf("i=%d A=%.14le\n", i, node->A);
00252         fprintf(file, "%.14le %.14le %.14le %.14le %.14le %.14le\n",
00253             node->x,
00254             node->A,
00255             node->Q,
00256             node->As,
00257             node->Ai,
00258             node->Asi);
00259     }
00260 }
00261
00270 void mesh_write_flows(Mesh *mesh, FILE *file)
00271 {
00272     int i, n1;
00273     Node *node = mesh->node;
00274     n1 = mesh->n - 1;
00275     for (i = 0; i < n1; ++i)
00276     {
00277         fprintf(file, "%.14le %.14le %.14le %.14le %.14le\n",
00278             0.5 * (node[i].x + node[i + 1].x),
00279             (node[i + 1].Q * node[i + 1].u - node[i].Q * node[i].u)
00280             / node[i].ix,
00281             0.5 * G * (node[i + 1].A + node[i].A)
00282             * (node[i + 1].zb - node[i].zb) / node[i].ix,
00283             0.5 * G * (node[i + 1].A + node[i].A)
00284             * (node[i + 1].h - node[i].h) / node[i].ix,
00285             0.25 * G * (node[i + 1].A + node[i].A)
00286             * (node[i + 1].Sf + node[i].Sf));
00287     }
00288 }
00289
00297 double mesh_water_mass(Mesh *mesh)
00298 {
00299     int i;
00300     double mass = 0.;
00301     Node *node = mesh->node;
00302     for (i = 0; i < mesh->n; ++i)
00303         mass += node[i].dx * (node[i].A + node[i].Ai);
00304     for (i=0; i<mesh->n; ++i) printf("i=%d A=%.14le Ai=%.14le\n", i, node[i].A,
00305         node[i].Ai);
00306     return mass;
00307 }
00315 double mesh_solute_mass(Mesh *mesh)
00316 {
00317     int i;
00318     double mass = 0.;

```

```
00319     Node *node = mesh->node;
00320     for (i = 0; i < mesh->n; ++i)
00321         mass += node[i].dx * (node[i].As + node[i].Asi);
00322     return mass;
00323 }
00324
```

4.11 mesh.h File Reference

Header file to define a mesh.

Data Structures

- struct [Mesh](#)
Struct to define a mesh.

Functions

- int [mesh_open](#) (Mesh *mesh, Channel *channel)
Function to open a mesh.
- int [mesh_read](#) (Mesh *mesh, Channel *channel, FILE *file)
Function to read a mesh.
- void [mesh_write_variables](#) (Mesh *mesh, FILE *file)
Function to write the variables of a mesh in a file.
- void [mesh_write_flows](#) (Mesh *mesh, FILE *file)
Function to write the flows of a mesh in a file.
- double [mesh_water_mass](#) (Mesh *mesh)
Function to calculate the water mass in a mesh.
- double [mesh_solute_mass](#) (Mesh *mesh)
Function to calculate the solute mass in a mesh.

4.11.1 Detailed Description

Header file to define a mesh.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [mesh.h](#).

4.11.2 Function Documentation

4.11.2.1 int mesh_open (Mesh * *mesh*, Channel * *channel*)

Function to open a mesh.

Parameters

<i>mesh</i>	mesh struct.
<i>channel</i>	channel struct.

Returns

0 on error, 1 on success.

Definition at line 53 of file [mesh.c](#).

```
{
    int i;
    double ix, Z;
    Node *node;
    mesh->node = node = (Node*)malloc(mesh->n * sizeof(Node));
    if (!mesh->node)
    {
        printf("mesh: not enough memory\n");
        return 0;
    }
    ix = channel->length / (mesh->n - 1);
    Z = channel->wall_slope;
    for (i = 0; i < mesh->n; ++i)
    {
        node[i].ix = ix;
        node[i].x = i * ix;
        node[i].zb = (channel->length - node[i].x) * channel->slope;
        node[i].B0 = channel->bottom_width;
        node[i].Z = Z;
        memcpy(node[i].friction_coefficient, channel->
            friction_coefficient,
            3 * sizeof(double));
        memcpy(node[i].infiltration_coefficient,
            channel->infiltration_coefficient, 4 * sizeof(double));
        memcpy(node[i].diffusion_coefficient, channel->
            diffusion_coefficient,
            sizeof(double));
    }
    node[0].dx = node[mesh->n - 1].dx = 0.5 * ix;
    for (i = 0; ++i < mesh->n - 1;) node[i].dx = ix;
    #if DEBUG_MESH_OPEN
    for (i=0; i < mesh->n; ++i)
        printf("node:\nx=%lf ix=%lf dx=%lf\nzb=%lf B0=%lf Z=%lf\n",
            node[i].x,
            node[i].ix,
            node[i].dx,
            node[i].zb,
            node[i].B0,
            node[i].Z);
    #endif
    return 1;
}
```

4.11.2.2 int mesh_read (Mesh * *mesh*, Channel * *channel*, FILE * *file*)

Function to read a mesh.

Parameters

<i>mesh</i>	mesh struct.
<i>channel</i>	channel struct.
<i>file</i>	input file.

Returns

0 on error, 1 on succes.

Definition at line 200 of file [mesh.c](#).

```
{
```

```

char *msg;
if (fscanf(file, "%d%d", &mesh->n, &mesh->type) != 2)
{
    msg = "mesh: bad defined\n";
    goto bad;
}
if (mesh->n < 3)
{
    msg = "mesh: bad nodes number\n";
    goto bad;
}
#ifdef DEBUG_MODEL_READ
printf("mesh: n=%d type=%d\n", mesh->n, mesh->type);
#endif
if (!mesh_open(mesh, channel)) return 0;
switch (mesh->type)
{
    case 1:
        mesh_initial_conditions_dry(mesh);
        break;
    case 2:
        if (!mesh_initial_conditions_profile(
            mesh, file)) return 0;
        break;
    default:
        msg = "mesh: bad type\n";
        goto bad;
}
return 1;

bad:
printf(msg);
return 0;
}

```

4.11.2.3 double mesh_solute_mass (Mesh * *mesh*)

Function to calculate the solute mass in a mesh.

Parameters

<i>mesh</i>	mesh struct.
-------------	--------------

Returns

solute mass.

Definition at line 315 of file [mesh.c](#).

```

{
    int i;
    double mass = 0.;
    Node *node = mesh->node;
    for (i = 0; i < mesh->n; ++i)
        mass += node[i].dx * (node[i].As + node[i].Asi);
    return mass;
}

```

4.11.2.4 double mesh_water_mass (Mesh * *mesh*)

Function to calculate the water mass in a mesh.

Parameters

<i>mesh</i>	mesh struct.
-------------	--------------

Returns

water mass

Definition at line 297 of file [mesh.c](#).

```
{
    int i;
    double mass = 0.;
    Node *node = mesh->node;
    for (i = 0; i < mesh->n; ++i)
        mass += node[i].dx * (node[i].A + node[i].Ai);
    for (i=0; i<mesh->n; ++i) printf("i=%d A=%.14le Ai=%.14le\n", i, node[i].A,
        node[i].Ai);
    return mass;
}
```

4.11.2.5 void mesh_write_flows (Mesh * *mesh*, FILE * *file*)

Function to write the flows of a mesh in a file.

Parameters

<i>mesh</i>	mesh struct.
<i>file</i>	input file.

Definition at line 270 of file [mesh.c](#).

```
{
    int i, n1;
    Node *node = mesh->node;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        fprintf(file, "%.14le %.14le %.14le %.14le %.14le\n",
            0.5 * (node[i].x + node[i + 1].x),
            (node[i + 1].Q * node[i + 1].u - node[i].Q * node[i].u)
            / node[i].ix,
            0.5 * G * (node[i + 1].A + node[i].A)
            * (node[i + 1].zb - node[i].zb) / node[i].ix,
            0.5 * G * (node[i + 1].A + node[i].A)
            * (node[i + 1].h - node[i].h) / node[i].ix,
            0.25 * G * (node[i + 1].A + node[i].A)
            * (node[i + 1].Sf + node[i].Sf));
    }
}
```

4.11.2.6 void mesh_write_variables (Mesh * *mesh*, FILE * *file*)

Function to write the variables of a mesh in a file.

Parameters

<i>mesh</i>	mesh struct.
<i>file</i>	input file.

Definition at line 244 of file [mesh.c](#).

```
{
    int i;
    Node *node;
    for (i = 0; i < mesh->n; ++i)
    {
        node = mesh->node + i;
        printf("i=%d A=%.14le\n", i, node->A);
        fprintf(file, "%.14le %.14le %.14le %.14le %.14le\n",
            node->x,
            node->A,
            node->A,
```

```

        node->Q,
        node->As,
        node->Ai,
        node->Asi);
    }
}

```

4.12 mesh.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010         1. Redistributions of source code must retain the above copyright notice,
00011             this list of conditions and the following disclaimer.
00012         2. Redistributions in binary form must reproduce the above copyright
00013             notice,
00014             this list of conditions and the following disclaimer in the
00015             documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022     SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 // in order to prevent multiple definitions
00033 #ifndef MESH__H
00034 #define MESH__H 1
00035
00036 struct _Mesh
00037 {
00038     Node *node;
00039     int n, type;
00040 };
00041
00042 typedef struct _Mesh Mesh;
00043
00044 // member functions
00045 int mesh_open(Mesh *mesh, Channel *channel);
00046 int mesh_read(Mesh *mesh, Channel *channel, FILE *file);
00047 void mesh_write_variables(Mesh *mesh, FILE *file);
00048 void mesh_write_flows(Mesh *mesh, FILE *file);
00049 double mesh_water_mass(Mesh *mesh);
00050 double mesh_solute_mass(Mesh *mesh);
00051
00052 #endif

```

4.13 model.c File Reference

Source file to define the numerical model.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
```

Functions

- void [model_parameters](#) (Model *model)
Function to calculate the model parameters.
- void [model_infiltration](#) (Model *model)
Function to make the infiltration model.
- void [model_diffusion_explicit](#) (Model *model)
Function to make the explicit diffusion model.
- void [model_diffusion_implicit](#) (Model *model)
Function to make the implicit diffusion model.
- double [model_node_diffusion_1dt_max](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the diffusion model.
- void [model_step](#) (Model *model)
Function to make a step of the numerical model.
- int [model_read](#) (Model *model, char *file_name)
Function to read the numerical model.
- void [model_print](#) (Model *model, int nsteps)
Function to print a model stat.
- void [model_write_advance](#) (Model *model, FILE *file)
Function to write in a file the channel water advance.
- int [model_probes_read](#) (Model *model, char *name)
Function to read the model probes in a file.
- void [model_write_probes](#) (Model *model, FILE *file)
Function to write the model probes in a file.

4.13.1 Detailed Description

Source file to define the numerical model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model.c](#).

4.13.2 Function Documentation

4.13.2.1 void model_diffusion_explicit (Model * model)

Function to make the explicit diffusion model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 92 of file [model.c](#).

```
{
    int i, n1;
    double dD;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        dD = model->dt * fmin(node[i + 1].KxA, node[i].KxA)
            * (node[i + 1].s - node[i].s) / node[i].ix;
        node[i].As += dD / node[i].dx;
        node[i + 1].As -= dD / node[i + 1].dx;
    }
}
```

4.13.2.2 void model_diffusion_implicit (Model * model)

Function to make the implicit diffusion model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 114 of file [model.c](#).

```
{
    int i, n1;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    double k, C[mesh->n], D[mesh->n], E[mesh->n], H[mesh->n];
    for (i = 0; i < mesh->n; ++i)
    {
        D[i] = node[i].A * node[i].dx;
        H[i] = node[i].As * node[i].dx;
    }
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        k = model->dt * fmin(node[i + 1].KxA, node[i].KxA) / node[i].ix;
        C[i] = E[i] = -k;
        D[i] += k;
        D[i + 1] += k;
    }
    for (i = 0; i < n1; ++i)
    {
        if (D[i] == 0.) k = 0; else k = C[i] / D[i];
        D[i + 1] -= k * E[i];
        H[i + 1] -= k * H[i];
    }
    if (D[i] == 0.) H[i] = 0; else H[i] /= D[i];
    node[i].As = H[i] * node[i].A;
    while (--i >= 0)
    {
        if (D[i] == 0.) H[i] = 0; else H[i] = (H[i] - E[i] * H[i+1]) / D[i];
        ;
        node[i].As = H[i] * node[i].A;
    }
}
```

4.13.2.3 void model_infiltration (Model * *model*)

Function to make the infiltration model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 68 of file [model.c](#).

```
{
    int i;
    double Piddt;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    printf("t=%lg dt=%lg Pi=%lg A=%lg Ai=%lg\n", model->t, model->dt, node[0].Pi,
        node[0].A, node[0].Ai);
    for (i = 0; i < mesh->n; ++i)
    {
        Piddt = fmin(node[i].Pi * model->dt, node[i].A);
        node[i].A -= Piddt;
        node[i].Ai += Piddt;
        Piddt *= node[i].S;
        node[i].As -= Piddt;
        node[i].Asi += Piddt;
    }
}
```

4.13.2.4 double model_node_diffusion_1dt_max (Node * *node*)

Function to calculate the allowed maximum time step size in a node with the diffusion model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 156 of file [model.c](#).

```
{
    return (2 * node->Kx + fabs(node->u) * node->dx) / (node->dx * node->dx);
}
```

4.13.2.5 void model_parameters (Model * *model*)

Function to calculate the model parameters.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 50 of file [model.c](#).

```
{
    int i, n1;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    model->model_node_parameters_right(model, node);
    n1 = mesh->n - 1;
    for (i = 0; ++i < n1;)
        model->model_node_parameters_centre(model, node + i);
}
```

```

    model->model_node_parameters_left(model, node + i);
}

```

4.13.2.6 void model_print (Model * *model*, int *nsteps*)

Function to print a model stat.

Parameters

<i>model</i>	model struct.
<i>nsteps</i>	number of time steps.

Definition at line 284 of file [model.c](#).

```

{
    printf(
        "main: steps number=%d t=%.14lg water mass=%.14lg solute mass=%.14lg\n"
        ,
        nsteps,
        model->t,
        mesh_water_mass(model->mesh),
        mesh_solute_mass(model->mesh));
}

```

4.13.2.7 int model_probes_read (Model * *model*, char * *name*)

Function to read the model probes in a file.

Parameters

<i>model</i>	model struct.
<i>name</i>	input file name.

Returns

0 on error, 1 on success.

Definition at line 325 of file [model.c](#).

```

{
    int i, j, k;
    double d, dmin, *x;
    char *msg;
    FILE *file;
    Probes *probes = model->probes;
    Node *node = model->mesh->node;
    file = fopen(name, "r");
    if (!file)
    {
        msg = "probes: unable to open the input file\n";
        goto bad2;
    }
    if (fscanf(file, "%d", &probes->n) != 1) goto bad;
    probes->x = x = (double*)malloc(probes->n * sizeof(double));
    probes->node = (int*)malloc(probes->n * sizeof(int));
    for (i = 0; i < probes->n; ++i)
    {
        if (fscanf(file, "%lf", x + i) != 1) goto bad;
        k = 0;
        dmin = fabs(x[i] - node[0].x);
        for (j = 0; ++j < model->mesh->n;)
        {
            d = fabs(x[i] - node[j].x);
            if (d < dmin)
            {
                dmin = d;
                k = j;
            }
        }
    }
}

```

```

        }
        probes->node[i] = k;
    }
    return 1;

bad:
    msg = "probes: bad data\n";
    fclose(file);

bad2:
    printf(msg);
    return 0;
}

```

4.13.2.8 int model_read (Model * model, char * file_name)

Function to read the numerical model.

Parameters

<i>model</i>	model struct. file_name = name of the input data file
--------------	---

Returns

0 on error, 1 on success.

Definition at line 203 of file [model.c](#).

```

{
    char *msg;
    FILE *file;

    file = fopen(file_name, "r");
    if (!file)
    {
        msg = "model: unable to open the input file\n";
        goto bad;
    }

    if (!channel_read(model->channel, file)) goto bad;
    switch (model->channel->friction_model)
    {
    case 1:
        model->node_friction = node_friction_Manning;
    }
    switch (model->channel->infiltration_model)
    {
    case 1:
        model->node_infiltration = node_infiltration_KostiakovLewis
        ;
    }
    switch (model->channel->diffusion_model)
    {
    case 1:
        model->node_diffusion = node_diffusion_Rutherford
        ;
    }
    if (!mesh_read(model->mesh, model->channel, file)) goto bad;

    model->node_inlet = node_inlet;
    switch (model->channel->type_outlet)
    {
    case 1:
        model->node_outlet = node_outlet_closed;
        break;
    case 2:
        model->node_outlet = node_outlet_open;
    }

    if (fscanf(file, "%lf%lf%lf%lf%d%d",
        &model->tfinal,
        &model->interval,
        &model->cfl,
        &model->minimum_depth,
        &model->type_surface_flow,
        &model->type_diffusion,

```

```

        &model->type_model) != 7)
    {
        msg = "model: bad data\n";
        goto bad;
    }
#ifdef DEBUG_MODEL_READ
    printf("model:\n"
           "tfinal=%lf interval=%lf cfl=%lf\n"
           "type_surface_flow=%d type_model=%d\n",
           model->tfinal,
           model->interval,
           model->cfl,
           model->type_surface_flow,
           model->type_model);
#endif

    fclose(file);
    return 1;

bad:
    if (file) fclose(file);
    printf(msg);
    return 0;
}

```

4.13.2.9 void model_step (Model * model)

Function to make a step of the numerical model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 167 of file [model.c](#).

```

{
    int i;
    double dtmax;
    Mesh *mesh = model->mesh;
    Channel *channel = model->channel;
    Node *node = mesh->node;
    for (i = 0, dtmax = 0; i < mesh->n; ++i)
        dtmax = fmax(dtmax, model->node_ldt_max(node + i));
    if (model->type_diffusion == 1)
    {
        for (i = 0, dtmax = 0; i < mesh->n; ++i)
            dtmax = fmax(dtmax, model_node_diffusion_ldt_max
                (node + i));
    }
    dtmax = model->cfl / dtmax;
    dtmax = fmin(dtmax, model->model_inlet_dtmax(model));
    model->t2 = fmin(model->tfinal, model->t + dtmax);
    model->dt = model->t2 - model->t;
    model->model_surface_flow(model);
    model->model_diffusion(model);
    model_infiltration(model);
    model->node_inlet(node, channel->water_inlet, channel->solute_inlet,
        model->t, model->t2);
    model->node_outlet(node + mesh->n - 1);
    model_parameters(model);
    model->t = model->t2;
}

```

4.13.2.10 void model_write_advance (Model * model, FILE * file)

Function to write in a file the channel water advance.

Parameters

<i>model</i>	model struct.
<i>file</i>	output file.

Definition at line 302 of file [model.c](#).

```
{
    int i;
    Mesh *mesh = model->mesh;
    Node *node;
    for (i = 0; i < mesh->n; ++i)
    {
        node = mesh->node + i;
        if (node->A == 0) break;
    }
    if (i) --i;
    fprintf(file, "%lg %lg\n", model->t, mesh->node[i].x);
}
```

4.13.2.11 void model_write_probes (Model * model, FILE * file)

Function to write the model probes in a file.

Parameters

<i>model</i>	model struct.
<i>file</i>	output file.

Definition at line 377 of file [model.c](#).

```
{
    int i;
    Probes *probes = model->probes;
    Node *node;

    // writing the time
    fprintf(file, "%lg ", model->t);
    for (i = 0; i < probes->n; ++i)
    {
        //writing the depth and the concentration of the i-th probe
        node = model->mesh->node + probes->node[i];
        fprintf(file, "%lg %lg ", node->h, node->s);
    }
    // writing a new row
    fprintf(file, "\n");
}
```

4.14 model.c

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012
00013     2. Redistributions in binary form must reproduce the above copyright
00014         notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
```

```

00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00035 #include <stdio.h>
00036 #include <math.h>
00037 #include <stdlib.h>
00038 #include "config.h"
00039 #include "channel.h"
00040 #include "node.h"
00041 #include "mesh.h"
00042 #include "model.h"
00043
00050 void model_parameters(Model *model)
00051 {
00052     int i, nl;
00053     Mesh *mesh = model->mesh;
00054     Node *node = mesh->node;
00055     model->model_node_parameters_right(model, node);
00056     nl = mesh->n - 1;
00057     for (i = 0; ++i < nl;)
00058         model->model_node_parameters_centre(model, node + i);
00059     model->model_node_parameters_left(model, node + i);
00060 }
00061
00068 void model_infiltration(Model *model)
00069 {
00070     int i;
00071     double Pidt;
00072     Mesh *mesh = model->mesh;
00073     Node *node = mesh->node;
00074     printf("t=%lg dt=%lg Pi=%lg A=%lg Ai=%lg\n", model->t, model->dt, node[0].Pi,
00075           node[0].A, node[0].Ai);
00076     for (i = 0; i < mesh->n; ++i)
00077     {
00078         Pidt = fmin(node[i].Pi * model->dt, node[i].A);
00079         node[i].A -= Pidt;
00080         node[i].Ai += Pidt;
00081         Pidt *= node[i].s;
00082         node[i].As -= Pidt;
00083         node[i].Asi += Pidt;
00084     }
00085
00092 void model_diffusion_explicit(Model *model)
00093 {
00094     int i, nl;
00095     double dD;
00096     Mesh *mesh = model->mesh;
00097     Node *node = mesh->node;
00098     nl = mesh->n - 1;
00099     for (i = 0; i < nl; ++i)
00100     {
00101         dD = model->dt * fmin(node[i + 1].KxA, node[i].KxA)
00102             * (node[i + 1].s - node[i].s) / node[i].ix;
00103         node[i].As += dD / node[i].dx;
00104         node[i + 1].As -= dD / node[i + 1].dx;
00105     }
00106 }
00107
00114 void model_diffusion_implicit(Model *model)
00115 {
00116     int i, nl;
00117     Mesh *mesh = model->mesh;
00118     Node *node = mesh->node;
00119     double k, C[mesh->n], D[mesh->n], E[mesh->n], H[mesh->n];
00120     for (i = 0; i < mesh->n; ++i)
00121     {
00122         D[i] = node[i].A * node[i].dx;
00123         H[i] = node[i].As * node[i].dx;
00124     }
00125     nl = mesh->n - 1;
00126     for (i = 0; i < nl; ++i)
00127     {
00128         k = model->dt * fmin(node[i + 1].KxA, node[i].KxA) / node[i].ix;
00129         C[i] = E[i] = -k;
00130         D[i] += k;
00131         D[i + 1] += k;
00132     }
00133     for (i = 0; i < nl; ++i)
00134     {
00135         if (D[i] == 0.) k = 0; else k = C[i] / D[i];
00136         D[i + 1] -= k * E[i];
00137         H[i + 1] -= k * H[i];
00138     }
00139     if (D[i] == 0.) H[i] = 0; else H[i] /= D[i];
00140     node[i].As = H[i] * node[i].A;

```

```

00141     while (--i >= 0)
00142     {
00143         if (D[i] == 0.) H[i] = 0; else H[i] = (H[i] - E[i] * H[i+1]) / D[i];
00144         node[i].As = H[i] * node[i].A;
00145     }
00146 }
00147
00156 double model_node_diffusion_ldt_max(Node *node)
00157 {
00158     return (2 * node->Kx + fabs(node->u) * node->dx) / (node->dx * node->dx);
00159 }
00160
00167 void model_step(Model *model)
00168 {
00169     int i;
00170     double dtmax;
00171     Mesh *mesh = model->mesh;
00172     Channel *channel = model->channel;
00173     Node *node = mesh->node;
00174     for (i = 0, dtmax = 0; i < mesh->n; ++i)
00175         dtmax = fmax(dtmax, model->node_ldt_max(node + i));
00176     if (model->type_diffusion == 1)
00177     {
00178         for (i = 0, dtmax = 0; i < mesh->n; ++i)
00179             dtmax = fmax(dtmax, model_node_diffusion_ldt_max
00180 (node + i));
00181     }
00182     dtmax = model->cfl / dtmax;
00183     dtmax = fmin(dtmax, model->model_inlet_dtmax(model));
00184     model->t2 = fmin(model->tfinal, model->t + dtmax);
00185     model->dt = model->t2 - model->t;
00186     model->model_surface_flow(model);
00187     model->model_diffusion(model);
00188     model_infiltration(model);
00189     model->node_inlet(node, channel->water_inlet, channel->solute_inlet,
00190 model->t, model->t2);
00191     model->node_outlet(node + mesh->n - 1);
00192     model_parameters(model);
00193     model->t = model->t2;
00194 }
00203 int model_read(Model *model, char *file_name)
00204 {
00205     char *msg;
00206     FILE *file;
00207
00208     file = fopen(file_name, "r");
00209     if (!file)
00210     {
00211         msg = "model: unable to open the input file\n";
00212         goto bad;
00213     }
00214
00215     if (!channel_read(model->channel, file)) goto bad;
00216     switch (model->channel->friction_model)
00217     {
00218     case 1:
00219         model->node_friction = node_friction_Manning;
00220     }
00221     switch (model->channel->infiltration_model)
00222     {
00223     case 1:
00224         model->node_infiltration = node_infiltration_KostiakovLewis
00225 ;
00226     }
00227     switch (model->channel->diffusion_model)
00228     {
00229     case 1:
00230         model->node_diffusion = node_diffusion_Rutherford
00231 ;
00232     }
00233
00234     if (!mesh_read(model->mesh, model->channel, file)) goto bad;
00235
00236     model->node_inlet = node_inlet;
00237     switch (model->channel->type_outlet)
00238     {
00239     case 1:
00240         model->node_outlet = node_outlet_closed;
00241         break;
00242     case 2:
00243         model->node_outlet = node_outlet_open;
00244     }
00245
00246     if (fscanf(file, "%lf%lf%lf%lf%d%d",
00247 &model->tfinal,
00248 &model->interval,

```

```

00247         &model->cfl,
00248         &model->minimum_depth,
00249         &model->type_surface_flow,
00250         &model->type_diffusion,
00251         &model->type_model) != 7)
00252     {
00253         msg = "model: bad data\n";
00254         goto bad;
00255     }
00256 #if DEBUG_MODEL_READ
00257     printf("model:\n"
00258           "tfinal=%lf interval=%lf cfl=%lf\n"
00259           "type_surface_flow=%d type_model=%d\n",
00260           model->tfinal,
00261           model->interval,
00262           model->cfl,
00263           model->type_surface_flow,
00264           model->type_model);
00265 #endif
00266     fclose(file);
00267     return 1;
00269 bad:
00271     if (file) fclose(file);
00272     printf(msg);
00273     return 0;
00274 }
00275
00284 void model_print(Model *model, int nsteps)
00285 {
00286     printf(
00287         "main: steps number=%d t=%.14lg water mass=%.14lg solute mass=%.14lg\n"
00288         ,
00289         nsteps,
00290         model->t,
00291         mesh_water_mass(model->mesh),
00292         mesh_solute_mass(model->mesh));
00293 }
00302 void model_write_advance(Model *model, FILE *file)
00303 {
00304     int i;
00305     Mesh *mesh = model->mesh;
00306     Node *node;
00307     for (i = 0; i < mesh->n; ++i)
00308     {
00309         node = mesh->node + i;
00310         if (node->A == 0) break;
00311     }
00312     if (i) --i;
00313     fprintf(file, "%lg %lg\n", model->t, mesh->node[i].x);
00314 }
00315
00325 int model_probes_read(Model *model, char *name)
00326 {
00327     int i, j, k;
00328     double d, dmin, *x;
00329     char *msg;
00330     FILE *file;
00331     Probes *probes = model->probes;
00332     Node *node = model->mesh->node;
00333     file = fopen(name, "r");
00334     if (!file)
00335     {
00336         msg = "probes: unable to open the input file\n";
00337         goto bad2;
00338     }
00339     if (fscanf(file, "%d", &probes->n) != 1) goto bad;
00340     probes->x = x = (double*)malloc(probes->n * sizeof(double));
00341     probes->node = (int*)malloc(probes->n * sizeof(int));
00342     for (i = 0; i < probes->n; ++i)
00343     {
00344         if (fscanf(file, "%lf", x + i) != 1) goto bad;
00345         k = 0;
00346         dmin = fabs(x[i] - node[0].x);
00347         for (j = 0; ++j < model->mesh->n;)
00348         {
00349             d = fabs(x[i] - node[j].x);
00350             if (d < dmin)
00351             {
00352                 dmin = d;
00353                 k = j;
00354             }
00355         }
00356         probes->node[i] = k;
00357     }

```

```

00358     return 1;
00359
00360 bad:
00361     msg = "probes: bad data\n";
00362     fclose(file);
00363
00364 bad2:
00365     printf(msg);
00366     return 0;
00367 }
00368
00377 void model_write_probes(Model *model, FILE *file)
00378 {
00379     int i;
00380     Probes *probes = model->probes;
00381     Node *node;
00382
00383     // writing the time
00384     fprintf(file, "%lg ", model->t);
00385     for (i = 0; i < probes->n; ++i)
00386     {
00387         //writing the depth and the concentration of the i-th probe
00388         node = model->mesh->node + probes->node[i];
00389         fprintf(file, "%lg %lg ", node->h, node->s);
00390     }
00391     // writing a new row
00392     fprintf(file, "\n");
00393 }

```

4.15 model.h File Reference

Header file to define the numerical model.

Data Structures

- struct [Probes](#)
Struct to define probes to save the evolution of the variables at a mesh cell.
- struct [Model](#)
Struct to define a numerical model.

Functions

- void [model_parameters](#) (Model *model)
Function to calculate the model parameters.
- void [model_infiltration](#) (Model *model)
Function to make the infiltration model.
- void [model_diffusion_explicit](#) (Model *model)
Function to make the explicit diffusion model.
- void [model_diffusion_implicit](#) (Model *model)
Function to make the implicit diffusion model.
- double [model_node_diffusion_1dt_max](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the diffusion model.
- void [model_step](#) (Model *model)
Function to make a step of the numerical model.
- int [model_read](#) (Model *model, char *file_name)
Function to read the numerical model.
- void [model_print](#) (Model *model, int nsteps)
Function to print a model stat.
- void [model_write_advance](#) (Model *model, FILE *file)
Function to write in a file the channel water advance.
- int [model_probes_read](#) (Model *model, char *name)

Function to read the model probes in a file.

- void [model_write_probes](#) (Model *model, FILE *file)

Function to write the model probes in a file.

4.15.1 Detailed Description

Header file to define the numerical model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model.h](#).

4.15.2 Function Documentation

4.15.2.1 void model_diffusion_explicit (Model * model)

Function to make the explicit diffusion model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 92 of file [model.c](#).

```
{
    int i, n1;
    double dD;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        dD = model->dt * fmin(node[i + 1].KxA, node[i].KxA)
            * (node[i + 1].s - node[i].s) / node[i].ix;
        node[i].As += dD / node[i].dx;
        node[i + 1].As -= dD / node[i + 1].dx;
    }
}
```

4.15.2.2 void model_diffusion_implicit (Model * model)

Function to make the implicit diffusion model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 114 of file [model.c](#).

```
{
    int i, n1;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    double k, C[mesh->n], D[mesh->n], E[mesh->n], H[mesh->n];
    for (i = 0; i < mesh->n; ++i)
```

```

{
    D[i] = node[i].A * node[i].dx;
    H[i] = node[i].As * node[i].dx;
}
n1 = mesh->n - 1;
for (i = 0; i < n1; ++i)
{
    k = model->dt * fmin(node[i + 1].KxA, node[i].KxA) / node[i].ix;
    C[i] = E[i] = -k;
    D[i] += k;
    D[i + 1] += k;
}
for (i = 0; i < n1; ++i)
{
    if (D[i] == 0.) k = 0; else k = C[i] / D[i];
    D[i + 1] -= k * E[i];
    H[i + 1] -= k * H[i];
}
if (D[i] == 0.) H[i] = 0; else H[i] /= D[i];
node[i].As = H[i] * node[i].A;
while (--i >= 0)
{
    if (D[i] == 0.) H[i] = 0; else H[i] = (H[i] - E[i] * H[i+1]) / D[i];
    ;
    node[i].As = H[i] * node[i].A;
}
}
}

```

4.15.2.3 void model_infiltration (Model * model)

Function to make the infiltration model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 68 of file [model.c](#).

```

{
    int i;
    double Pidt;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    printf("t=%lg dt=%lg Pi=%lg A=%lg Ai=%lg\n", model->t, model->dt, node[0].Pi,
        node[0].A, node[0].Ai);
    for (i = 0; i < mesh->n; ++i)
    {
        Pidt = fmin(node[i].Pi * model->dt, node[i].A);
        node[i].A -= Pidt;
        node[i].Ai += Pidt;
        Pidt *= node[i].s;
        node[i].As -= Pidt;
        node[i].Asi += Pidt;
    }
}

```

4.15.2.4 double model_node_diffusion_1dt_max (Node * node)

Function to calculate the allowed maximum time step size in a node with the diffusion model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 156 of file [model.c](#).

```
{
    return (2 * node->Kx + fabs(node->u) * node->dx) / (node->dx * node->dx);
}
```

4.15.2.5 void model_parameters (Model * *model*)

Function to calculate the model parameters.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 50 of file [model.c](#).

```
{
    int i, n1;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    model->model_node_parameters_right(model, node);
    n1 = mesh->n - 1;
    for (i = 0; ++i < n1;)
        model->model_node_parameters_centre(model, node + i);
    model->model_node_parameters_left(model, node + i);
}
```

4.15.2.6 void model_print (Model * *model*, int *nsteps*)

Function to print a model stat.

Parameters

<i>model</i>	model struct.
<i>nsteps</i>	number of time steps.

Definition at line 284 of file [model.c](#).

```
{
    printf(
        "main: steps number=%d t=%.14lg water mass=%.14lg solute mass=%.14lg\n"
        ,
        nsteps,
        model->t,
        mesh_water_mass(model->mesh),
        mesh_solute_mass(model->mesh));
}
```

4.15.2.7 int model_probes_read (Model * *model*, char * *name*)

Function to read the model probes in a file.

Parameters

<i>model</i>	model struct.
<i>name</i>	input file name.

Returns

0 on error, 1 on success.

Definition at line 325 of file [model.c](#).


```

{
    int i, j, k;
    double d, dmin, *x;
    char *msg;
    FILE *file;
    Probes *probes = model->probes;
    Node *node = model->mesh->node;
    file = fopen(name, "r");
    if (!file)
    {
        msg = "probes: unable to open the input file\n";
        goto bad2;
    }
    if (fscanf(file, "%d", &probes->n) != 1) goto bad;
    probes->x = x = (double*)malloc(probes->n * sizeof(double));
    probes->node = (int*)malloc(probes->n * sizeof(int));
    for (i = 0; i < probes->n; ++i)
    {
        if (fscanf(file, "%lf", x + i) != 1) goto bad;
        k = 0;
        dmin = fabs(x[i] - node[0].x);
        for (j = 0; ++j < model->mesh->n;)
        {
            d = fabs(x[i] - node[j].x);
            if (d < dmin)
            {
                dmin = d;
                k = j;
            }
        }
        probes->node[i] = k;
    }
    return 1;

bad:
    msg = "probes: bad data\n";
    fclose(file);

bad2:
    printf(msg);
    return 0;
}

```

4.15.2.8 int model_read (Model * model, char * file_name)

Function to read the numerical model.

Parameters

<i>model</i>	model struct. file_name = name of the input data file
--------------	---

Returns

0 on error, 1 on success.

Definition at line 203 of file [model.c](#).

```

{
    char *msg;
    FILE *file;

    file = fopen(file_name, "r");
    if (!file)
    {
        msg = "model: unable to open the input file\n";
        goto bad;
    }

    if (!channel_read(model->channel, file)) goto bad;
    switch (model->channel->friction_model)
    {
        case 1:
            model->node_friction = node_friction_Manning;
    }
    switch (model->channel->infiltration_model)
    {
        case 1:

```

```

        model->node_infiltration = node_infiltration_KostiakovLewis
    }
    switch (model->channel->diffusion_model)
    {
    case 1:
        model->node_diffusion = node_diffusion_Rutherford
    }

    if (!mesh_read(model->mesh, model->channel, file)) goto bad;

    model->node_inlet = node_inlet;
    switch (model->channel->type_outlet)
    {
    case 1:
        model->node_outlet = node_outlet_closed;
        break;
    case 2:
        model->node_outlet = node_outlet_open;
    }

    if (fscanf(file, "%lf%lf%lf%lf%d%d",
        &model->tfinal,
        &model->interval,
        &model->cfl,
        &model->minimum_depth,
        &model->type_surface_flow,
        &model->type_diffusion,
        &model->type_model) != 7)
    {
        msg = "model: bad data\n";
        goto bad;
    }
}
#endif
DEBUG_MODEL_READ
printf("model:\n"
    "tfinal=%lf interval=%lf cfl=%lf\n"
    "type_surface_flow=%d type_model=%d\n",
    model->tfinal,
    model->interval,
    model->cfl,
    model->type_surface_flow,
    model->type_model);
#endif

fclose(file);
return 1;

bad:
if (file) fclose(file);
printf(msg);
return 0;
}

```

4.15.2.9 void model_step (Model * model)

Function to make a step of the numerical model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 167 of file [model.c](#).

```

{
    int i;
    double dtmax;
    Mesh *mesh = model->mesh;
    Channel *channel = model->channel;
    Node *node = mesh->node;
    for (i = 0, dtmax = 0; i < mesh->n; ++i)
        dtmax = fmax(dtmax, model->node_ldt_max(node + i));
    if (model->type_diffusion == 1)
    {
        for (i = 0, dtmax = 0; i < mesh->n; ++i)
            dtmax = fmax(dtmax, model_node_diffusion_ldt_max
                (node + i));
    }
    dtmax = model->cfl / dtmax;
}

```

```

    dtmax = fmin(dtmax, model->model_inlet_dtmax(model));
    model->t2 = fmin(model->tfinal, model->t + dtmax);
    model->dt = model->t2 - model->t;
    model->model_surface_flow(model);
    model->model_diffusion(model);
    model_infiltration(model);
    model->node_inlet(node, channel->water_inlet, channel->solute_inlet,
        model->t, model->t2);
    model->node_outlet(node + mesh->n - 1);
    model_parameters(model);
    model->t = model->t2;
}

```

4.15.2.10 void model_write_advance (Model * *model*, FILE * *file*)

Function to write in a file the channel water advance.

Parameters

<i>model</i>	model struct.
<i>file</i>	output file.

Definition at line 302 of file [model.c](#).

```

{
    int i;
    Mesh *mesh = model->mesh;
    Node *node;
    for (i = 0; i < mesh->n; ++i)
    {
        node = mesh->node + i;
        if (node->A == 0) break;
    }
    if (i) --i;
    fprintf(file, "%lg %lg\n", model->t, mesh->node[i].x);
}

```

4.15.2.11 void model_write_probes (Model * *model*, FILE * *file*)

Function to write the model probes in a file.

Parameters

<i>model</i>	model struct.
<i>file</i>	output file.

Definition at line 377 of file [model.c](#).

```

{
    int i;
    Probes *probes = model->probes;
    Node *node;

    // writing the time
    fprintf(file, "%lg ", model->t);
    for (i = 0; i < probes->n; ++i)
    {
        //writing the depth and the concentration of the i-th probe
        node = model->mesh->node + probes->node[i];
        fprintf(file, "%lg %lg ", node->h, node->s);
    }
    // writing a new row
    fprintf(file, "\n");
}

```

4.16 model.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010         1. Redistributions of source code must retain the above copyright notice,
00011           this list of conditions and the following disclaimer.
00012
00013         2. Redistributions in binary form must reproduce the above copyright
00014           notice,
00015           this list of conditions and the following disclaimer in the
00016           documentation and/or other materials provided with the distribution.
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022     SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031 // in order to prevent multiple definitions
00032 #ifndef MODEL__H
00033 #define MODEL__H 1
00034
00035 struct _Probes
00036 {
00037     double *x;
00038     int *node, n;
00039 };
00040 typedef struct _Probes Probes;
00041
00042 struct _Model
00043 {
00044     Mesh mesh[1];
00045     Channel channel[1];
00046     Probes probes[1];
00047     double t, t2, dt, tfinal, cfl, interval,
00048     minimum_depth;
00049     void (*model_node_parameters_centre)(struct
00050     _Model *model, Node *node);
00051     void (*model_node_parameters_right)(struct
00052     _Model *model, Node *node);
00053     void (*model_node_parameters_left)(struct _Model
00054     *model, Node *node);
00055     double (*node_ldt_max)(Node *node);
00056     double (*model_inlet_dtmax)(struct _Model *model);
00057     void (*node_flows)(Node *node);
00058     void (*node_discharge_centre)(Node *node);
00059     void (*node_discharge_right)(Node *node);
00060     void (*node_discharge_left)(Node *node);
00061     void (*node_friction)(Node *node);
00062     void (*node_infiltration)(Node *node);
00063     void (*node_diffusion)(Node *node);
00064     void (*node_inlet)
00065     (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2
00066     );
00067     void (*node_outlet)(Node *node);
00068     void (*model_surface_flow)(struct _Model *model);
00069     void (*model_diffusion)(struct _Model *model);
00070     int type_surface_flow, type_diffusion,
00071     type_model;
00072 };
00073 typedef struct _Model Model;
00074 // member functions
00075 void model_parameters(Model *model);
00076 void model_infiltration(Model *model);
00077 void model_diffusion_explicit(Model *model);

```

```

00172 void model_diffusion_implicit(Model *model);
00173 double model_node_diffusion_idt_max(Node *node);
00174 void model_step(Model *model);
00175 int model_read(Model *model, char *file_name);
00176 void model_print(Model *model, int nsteps);
00177 void model_write_advance(Model *model, FILE *file);
00178 int model_probes_read(Model *model, char *name);
00179 void model_write_probes(Model *model, FILE *file);
00180
00181 #endif

```

4.17 model_complete.c File Reference

Source file to define the complete model.

```

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_complete.h"

```

Functions

- void [model_node_parameters_complete](#) (Model *model, Node *node)
Ffunction to calculate the numerical parameters of a node with the complete model.
- double [node_1dt_max_complete](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the complete model.
- void [node_flows_complete](#) (Node *node1)
Function to calculate the flux differences in a node with the complete model.
- double [model_inlet_dtmx_complete](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the complete model.

4.17.1 Detailed Description

Source file to define the complete model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_complete.c](#).

4.17.2 Function Documentation

4.17.2.1 double model_inlet_dtmx_complete (Model * model)

Function to calculate the allowed maximum time step size at the inlet with the complete model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 124 of file [model_complete.c](#).

```
{
    double A, Q, h, B, u, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    u = Q / A;
    return node->ix / (c + fabs(u));
}
```

4.17.2.2 void model_node_parameters_complete (Model * model, Node * node)

Function to calculate the numerical parameters of a node with the complete model.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 53 of file [model_complete.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    node_critical_velocity(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->Sf = node->F = node->T = node->Kx
            = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->Sf = node->F = node->T = node->Kx
            = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        node->u = node->Q / node->A;
        node->F = node->A * node->u * node->u;
        node->T = node->Q * node->s;
        model->node_friction(node);
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    node->l1 = node->u + node->c;
    node->l2 = node->u - node->c;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.17.2.3 double node_1dt_max_complete (Node * node)

Function to calculate the allowed maximum time step size in a node with the complete model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 95 of file [model_complete.c](#).

```
{
    return (node->c + fabs(node->u)) / node->dx;
}
```

4.17.2.4 void node_flows_complete (Node * node1)

Function to calculate the flux differences in a node with the complete model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 107 of file [model_complete.c](#).

```
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dF = node2->F - node1->F + G * 0.5 * (node2->A + node1->A)
        * (node2->zs - node1->zs + 0.5 * (node2->Sf + node1->Sf) * node1->ix);
    node1->dT = node2->T - node1->T;
}
```

4.18 model_complete.c

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 #include <stdio.h>
00033 #include <math.h>
00034 #include "config.h"
00035 #include "channel.h"
```

```

00039 #include "node.h"
00040 #include "mesh.h"
00041 #include "model.h"
00042 #include "model_complete.h"
00043
00053 void model_node_parameters_complete(Model *model,
    Node *node)
00054 {
00055     node_depth(node);
00056     node_width(node);
00057     node_perimeter(node);
00058     node_critical_velocity(node);
00059     if (node->A <= 0.)
00060     {
00061         node->s = node->Q = node->u = node->Sf = node->F = node->T = node->Kx
            = node->KxA = 0.;
00062     }
00063     else if (node->h < model->minimum_depth)
00064     {
00065         node->s = node->As / node->A;
00066         node->Q = node->u = node->Sf = node->F = node->T = node->Kx
            = node->KxA = 0.;
00067     }
00068     else
00069     {
00070         node->s = node->As / node->A;
00071         node->u = node->Q / node->A;
00072         node->F = node->A * node->u * node->u;
00073         node->T = node->Q * node->s;
00074         model->node_friction(node);
00075         model->node_diffusion(node);
00076         node->KxA = node->Kx * node->A;
00077     }
00078     node->zs = node->zb + node->h;
00079     node->l1 = node->u + node->c;
00080     node->l2 = node->u - node->c;
00081     model->node_infiltration(node);
00082     node->Pi = node->P * node->i;
00083 }
00084
00085 double node_1dt_max_complete(Node *node)
00086 {
00087     return (node->c + fabs(node->u)) / node->dx;
00088 }
00089
00090 void node_flows_complete(Node *node1)
00091 {
00092     Node *node2 = node1 + 1;
00093     node1->dQ = node2->Q - node1->Q;
00094     node1->dF = node2->F - node1->F + G * 0.5 * (node2->A + node1->A)
        * (node2->zs - node1->zs + 0.5 * (node2->Sf + node1->Sf) * node1->ix);
00095     node1->dT = node2->T - node1->T;
00096 }
00097
00098 double model_inlet_dtmax_complete(Model *model)
00099 {
00100     double A, Q, h, B, u, c;
00101     Node *node = model->mesh->node;
00102     Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
00103     h = node_critical_depth(node, Q);
00104     A = h * (node->B0 + h * node->Z);
00105     B = node->B0 + 2 * h * node->Z;
00106     c = sqrt(G * A / B);
00107     u = Q / A;
00108     return node->ix / (c + fabs(u));
00109 }
00110

```

4.19 model_complete.h File Reference

Header file to define the complete model.

Functions

- void [model_node_parameters_complete](#) (Model *model, Node *node)
Ffunction to calculate the numerical parameters of a node with the complete model.
- double [node_1dt_max_complete](#) (Node *node)

Function to calculate the allowed maximum time step size in a node with the complete model.

- void [node_flows_complete](#) (Node *node1)

Function to calculate the flux differences in a node with the complete model.

- double [model_inlet_dtmax_complete](#) (Model *model)

Function to calculate the allowed maximum time step size at the inlet with the complete model.

4.19.1 Detailed Description

Header file to define the complete model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_complete.h](#).

4.19.2 Function Documentation

4.19.2.1 double model_inlet_dtmax_complete (Model * model)

Function to calculate the allowed maximum time step size at the inlet with the complete model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 124 of file [model_complete.c](#).

```
{
    double A, Q, h, B, u, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    u = Q / A;
    return node->ix / (c + fabs(u));
}
```

4.19.2.2 void model_node_parameters_complete (Model * model, Node * node)

Function to calculate the numerical parameters of a node with the complete model.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 53 of file [model_complete.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    node_critical_velocity(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->Sf = node->F = node->T = node->Kx
        = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->Sf = node->F = node->T = node->Kx
        = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        node->u = node->Q / node->A;
        node->F = node->A * node->u * node->u;
        node->T = node->Q * node->s;
        model->node_friction(node);
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    node->l1 = node->u + node->c;
    node->l2 = node->u - node->c;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.19.2.3 double node_1dt_max_complete (Node * node)

Function to calculate the allowed maximum time step size in a node with the complete model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 95 of file [model_complete.c](#).

```
{
    return (node->c + fabs(node->u)) / node->dx;
}
```

4.19.2.4 void node_flows_complete (Node * node1)

Function to calculate the flux differences in a node with the complete model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 107 of file [model_complete.c](#).

```
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dF = node2->F - node1->F + G * 0.5 * (node2->A + node1->A)
    * (node2->zs - node1->zs + 0.5 * (node2->Sf + node1->Sf) * node1->ix);
}
```

```

    model->dT = node2->T - model->T;
}

```

4.20 model_complete.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 // in order to prevent multiple definitions
00033 #ifndef MODEL_COMPLETE__H
00034 #define MODEL_COMPLETE__H 1
00035
00036 // member functions
00037
00038 void model_node_parameters_complete(Model *model,
00039     Node *node);
00040 double node_ldt_max_complete(Node *node);
00041 void node_flows_complete(Node *node);
00042 double model_inlet_dtmax_complete(Model *model);
00043
00044 #endif

```

4.21 model_complete_LaxFriedrichs.c File Reference

Source file to define the Lax-Friedrichs numerical model applied to the complete model.

```

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_complete_LaxFriedrichs.h"

```

Functions

- void [model_surface_flow_complete_LaxFriedrichs](#) (Model *model)
Function to make the surface flow with the Lax-Friedrichs numerical scheme.

4.21.1 Detailed Description

Source file to define the Lax-Friedrichs numerical model applied to the complete model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_complete_LaxFriedrichs.c](#).

4.21.2 Function Documentation

4.21.2.1 void model_surface_flow_complete_LaxFriedrichs (Model * model)

Function to make the surface flow with the Lax-Friedrichs numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 52 of file [model_complete_LaxFriedrichs.c](#).

```
{
    int i, n1;
    double k1, k2, inlet_water_contribution, inlet_solute_contribution;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        model->node_flows(node + i);
        node[i].dQr = node[i].dFr = node[i].dTr = node[i].dQl = node[i].
dFl
        = node[i].dTl = 0;
        if (node[i].h <= model->minimum_depth &&
            node[i + 1].h <= model->minimum_depth)
            continue;

        // wave decomposition

        node[i].dQr = node[i].dQl = 0.5 * node[i].dQ;
        node[i].dFr = node[i].dFl = 0.5 * node[i].dF;
        node[i].dTr = node[i].dTl = 0.5 * node[i].dT;

        // artificial viscosity

        k1 = 0.5 * fmax(node[i + 1].ll, node[i].ll);
        k2 = k1 * (node[i + 1].A - node[i].A);
        node[i].dQl += k2;
        node[i].dQr -= k2;
        k2 = k1 * node[i].dQ;
        node[i].dFl += k2;
        node[i].dFr -= k2;
        k2 = k1 * (node[i + 1].As - node[i].As);
        node[i].dTl += k2;
        node[i].dTr -= k2;
    }

    // variables actualization
    for (i = 0; i < n1; ++i)
    {
        node[i].A -= model->dt * node[i].dQr / node[i].dx;
        node[i].Q -= model->dt * node[i].dFr / node[i].dx;
        node[i].As -= model->dt * node[i].dTr / node[i].dx;
        node[i + 1].A -= model->dt * node[i].dQl / node[i + 1].dx;
        node[i + 1].Q -= model->dt * node[i].dFl / node[i + 1].dx;
```

```

        node[i + 1].As -= model->dt * node[i].dTl / node[i + 1].dx;
    }
    node[0].A -= inlet_water_contribution / node[0].dx;
    node[0].As -= inlet_solute_contribution / node[0].dx;
}

```

4.22 model_complete_LaxFriedrichs.c

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010         1. Redistributions of source code must retain the above copyright notice,
00011             this list of conditions and the following disclaimer.
00012         2. Redistributions in binary form must reproduce the above copyright
00013             notice,
00014             this list of conditions and the following disclaimer in the
00015             documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018 OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021 EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023 INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031 #include <stdio.h>
00032 #include <math.h>
00033 #include "config.h"
00034 #include "channel.h"
00035 #include "node.h"
00036 #include "mesh.h"
00037 #include "model.h"
00038 #include "model_complete_LaxFriedrichs.h"
00039
00040 void model_surface_flow_complete_LaxFriedrichs
00041 (Model *model)
00042 {
00043     int i, nl;
00044     double k1, k2, inlet_water_contribution, inlet_solute_contribution;
00045     Mesh *mesh = model->mesh;
00046     Node *node = mesh->node;
00047     inlet_water_contribution = model->dt * node[0].Q;
00048     inlet_solute_contribution = model->dt * node[0].T;
00049     nl = mesh->n - 1;
00050     for (i = 0; i < nl; ++i)
00051     {
00052         model->node_flows(node + i);
00053         node[i].dQr = node[i].dFr = node[i].dTr = node[i].dQl = node[i].dFl
00054             = node[i].dTl = 0;
00055         if (node[i].h <= model->minimum_depth &&
00056             node[i + 1].h <= model->minimum_depth)
00057             continue;
00058
00059         // wave decomposition
00060
00061         node[i].dQr = node[i].dQl = 0.5 * node[i].dQ;
00062         node[i].dFr = node[i].dFl = 0.5 * node[i].dF;
00063         node[i].dTr = node[i].dTl = 0.5 * node[i].dT;
00064
00065         // artificial viscosity
00066
00067         k1 = 0.5 * fmax(node[i + 1].ll, node[i].ll);
00068         k2 = k1 * (node[i + 1].A - node[i].A);
00069         node[i].dQl += k2;
00070         node[i].dQr -= k2;
00071         k2 = k1 * node[i].dQ;
00072         node[i].dFl += k2;
00073     }
00074 }

```

```

00084         node[i].dFr -= k2;
00085         k2 = k1 * (node[i + 1].As - node[i].As);
00086         node[i].dTl += k2;
00087         node[i].dTr -= k2;
00088     }
00089
00090     // variables actualization
00091
00092     for (i = 0; i < n1; ++i)
00093     {
00094         node[i].A -= model->dt * node[i].dQr / node[i].dx;
00095         node[i].Q -= model->dt * node[i].dFr / node[i].dx;
00096         node[i].As -= model->dt * node[i].dTr / node[i].dx;
00097         node[i + 1].A -= model->dt * node[i].dQl / node[i + 1].dx;
00098         node[i + 1].Q -= model->dt * node[i].dFl / node[i + 1].dx;
00099         node[i + 1].As -= model->dt * node[i].dTl / node[i + 1].dx;
00100     }
00101     node[0].A -= inlet_water_contribution / node[0].dx;
00102     node[0].As -= inlet_solute_contribution / node[0].dx;
00103 }

```

4.23 model_complete_LaxFriedrichs.h File Reference

Header file to define the Lax-Friedrichs numerical model applied to the complete model.

Functions

- void [model_surface_flow_complete_LaxFriedrichs](#) (Model *model)
Function to make the surface flow with the Lax-Friedrichs numerical scheme.

4.23.1 Detailed Description

Header file to define the Lax-Friedrichs numerical model applied to the complete model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_complete_LaxFriedrichs.h](#).

4.23.2 Function Documentation

4.23.2.1 void model_surface_flow_complete_LaxFriedrichs (Model * model)

Function to make the surface flow with the Lax-Friedrichs numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 52 of file [model_complete_LaxFriedrichs.c](#).

```

{
    int i, n1;
    double k1, k2, inlet_water_contribution, inlet_solute_contribution;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    inlet_water_contribution = model->dt * node[0].Q;

```

```

inlet_solute_contribution = model->dt * node[0].T;
n1 = mesh->n - 1;
for (i = 0; i < n1; ++i)
{
    model->node_flows(node + i);
    node[i].dQr = node[i].dFr = node[i].dTr = node[i].dQl = node[i].
dFl
    = node[i].dTl = 0;
    if (node[i].h <= model->minimum_depth &&
        node[i + 1].h <= model->minimum_depth)
        continue;

    // wave decomposition

    node[i].dQr = node[i].dQl = 0.5 * node[i].dQ;
    node[i].dFr = node[i].dFl = 0.5 * node[i].dF;
    node[i].dTr = node[i].dTl = 0.5 * node[i].dT;

    // artificial viscosity

    k1 = 0.5 * fmax(node[i + 1].ll, node[i].ll);
    k2 = k1 * (node[i + 1].A - node[i].A);
    node[i].dQl += k2;
    node[i].dQr -= k2;
    k2 = k1 * node[i].dQ;
    node[i].dFl += k2;
    node[i].dFr -= k2;
    k2 = k1 * (node[i + 1].As - node[i].As);
    node[i].dTl += k2;
    node[i].dTr -= k2;
}

// variables actualization

for (i = 0; i < n1; ++i)
{
    node[i].A -= model->dt * node[i].dQr / node[i].dx;
    node[i].Q -= model->dt * node[i].dFr / node[i].dx;
    node[i].As -= model->dt * node[i].dTr / node[i].dx;
    node[i + 1].A -= model->dt * node[i].dQl / node[i + 1].dx;
    node[i + 1].Q -= model->dt * node[i].dFl / node[i + 1].dx;
    node[i + 1].As -= model->dt * node[i].dTl / node[i + 1].dx;
}
node[0].A -= inlet_water_contribution / node[0].dx;
node[0].As -= inlet_solute_contribution / node[0].dx;
}

```

4.24 model_complete_LaxFriedrichs.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031 // in order to prevent multiple definitions

```

```

00038 #ifndef MODEL_COMPLETE_LAXFRIEDRICHS__H
00039 #define MODEL_COMPLETE_LAXFRIEDRICHS__H 1
00040
00041 // member functions
00042
00043 void model_surface_flow_complete_LaxFriedrichs
    (Model *model);
00044
00045 #endif

```

4.25 model_diffusive.c File Reference

Source file to define the diffusive model.

```

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_diffusive.h"

```

Functions

- void [node_discharge_centre_diffusive_Manning](#) (Node *node)
Function to calculate the diffusive discharge with the Manning model using centred derivatives.
- void [node_discharge_right_diffusive_Manning](#) (Node *node)
Function to calculate the diffusive discharge with the Manning model using right derivatives.
- void [node_discharge_left_diffusive_Manning](#) (Node *node)
Function to calculate the diffusive discharge with the Manning model using left derivatives.
- void [model_node_parameters_centre_diffusive](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the diffusive model using centred derivatives.
- void [model_node_parameters_right_diffusive](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the diffusive model using right derivatives.
- void [model_node_parameters_left_diffusive](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the diffusive model using left derivatives.
- double [node_1dt_max_diffusive](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the diffusive model.
- void [node_flows_diffusive](#) (Node *node1)
Function to calculate the flux differences in a node with the diffusive model.
- double [model_inlet_dtmax_diffusive](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the diffusive model.

4.25.1 Detailed Description

Source file to define the diffusive model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_diffusive.c](#).

4.25.2 Function Documentation

4.25.2.1 double model_inlet_dtmx_diffusive (Model * *model*)

Function to calculate the allowed maximum time step size at the inlet with the diffusive model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 246 of file [model_diffusive.c](#).

```
{
    double A, Q, h, B, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    return node->ix / c;
}
```

4.25.2.2 void model_node_parameters_centre_diffusive (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the diffusive model using centred derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 101 of file [model_diffusive.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA
            = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_centre(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_friction(node);
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.25.2.3 void model_node_parameters_left_diffusive (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the diffusive model using left derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 177 of file [model_diffusive.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.25.2.4 void model_node_parameters_right_diffusive (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the diffusive model using right derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 140 of file [model_diffusive.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_right(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.25.2.5 double node_1dt_max_diffusive (Node * node)

Function to calculate the allowed maximum time step size in a node with the diffusive model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 212 of file [model_diffusive.c](#).

```
{
    double u;
    u = 5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
        / (node->B * node->P);
    if (node->u > 0.)
        u += node->A * pow(node->A / node->P, 4./3.)
            / (node->friction_coefficient[0] * node->friction_coefficient[0]
              * node->u * node->dx);
    return u / node->dx;
}
```

4.25.2.6 void node_discharge_centre_diffusive_Manning (Node * node)

Function to calculate the diffusive discharge with the Manning model using centred derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 51 of file [model_diffusive.c](#).

```
{
    double dz;
    dz = (node - 1)->zS - (node + 1)->zS;
    if (dz <= 0.) node->Q = 0.; else
        node->Q = sqrt(dz / ((node - 1)->ix + node->ix)) * node->A
            * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.25.2.7 void node_discharge_left_diffusive_Manning (Node * node)

Function to calculate the diffusive discharge with the Manning model using left derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 83 of file [model_diffusive.c](#).

```
{
    double dz;
    dz = (node - 1)->zS - node->zS;
    if (dz <= 0.) node->Q = 0.; else
        node->Q = sqrt(dz / (node - 1)->ix) * node->A
            * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.25.2.8 void node_discharge_right_diffusive_Manning (Node * node)

Function to calculate the diffusive discharge with the Manning model using right derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 67 of file [model_diffusive.c](#).

```
{
    double dz;
    dz = node->zs - (node + 1)->zs;
    if (dz <= 0.) node->Q = 0.; else
        node->Q = sqrt(dz / node->ix) * node->A * pow(node->A / node->P, 2./3.)
        / node->friction_coefficient[0];
}
```

4.25.2.9 void node_flows_diffusive (Node * node1)

Function to calculate the flux differences in a node with the diffusive model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 231 of file [model_diffusive.c](#).

```
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dT = node2->T - node1->T;
}
```

4.26 model_diffusive.c

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018 OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021 EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023 INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 #include <stdio.h>
00033 #include <math.h>
```

```

00037 #include "config.h"
00038 #include "channel.h"
00039 #include "node.h"
00040 #include "mesh.h"
00041 #include "model.h"
00042 #include "model_diffusive.h"
00043
00051 void node_discharge_centre_diffusive_Manning
00052 (Node *node)
00053 {
00054     double dz;
00055     dz = (node - 1)->zs - (node + 1)->zs;
00056     if (dz <= 0.) node->Q = 0.; else
00057         node->Q = sqrt(dz / ((node - 1)->ix + node->ix)) * node->A
00058             * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
00059 }
00067 void node_discharge_right_diffusive_Manning
00068 (Node *node)
00069 {
00070     double dz;
00071     dz = node->zs - (node + 1)->zs;
00072     if (dz <= 0.) node->Q = 0.; else
00073         node->Q = sqrt(dz / node->ix) * node->A * pow(node->A / node->P, 2./3.)
00074             / node->friction_coefficient[0];
00075 }
00083 void node_discharge_left_diffusive_Manning
00084 (Node *node)
00085 {
00086     double dz;
00087     dz = (node - 1)->zs - node->zs;
00088     if (dz <= 0.) node->Q = 0.; else
00089         node->Q = sqrt(dz / (node - 1)->ix) * node->A
00090             * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
00091 }
00101 void model_node_parameters_centre_diffusive
00102 (Model *model, Node *node)
00103 {
00104     node_depth(node);
00105     node_width(node);
00106     node_perimeter(node);
00107     if (node->A <= 0.)
00108     {
00109         node->s = node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA
00110             = 0.;
00111     }
00112     else if (node->h < model->minimum_depth)
00113     {
00114         node->s = node->As / node->A;
00115         node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA = 0.;
00116     }
00117     else
00118     {
00119         node->s = node->As / node->A;
00120         model->node_discharge_centre(node);
00121         node->u = node->Q / node->A;
00122         node->T = node->Q * node->s;
00123         model->node_friction(node);
00124         model->node_diffusion(node);
00125         node->KxA = node->Kx * node->A;
00126     }
00127     node->zs = node->zb + node->h;
00128     model->node_infiltration(node);
00129     node->Pi = node->P * node->i;
00130 }
00140 void model_node_parameters_right_diffusive
00141 (Model *model, Node *node)
00142 {
00143     node_depth(node);
00144     node_width(node);
00145     node_perimeter(node);
00146     if (node->A <= 0.)
00147     {
00148         node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00149     }
00150     else if (node->h < model->minimum_depth)
00151     {
00152         node->s = node->As / node->A;
00153         node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00154     }
00155     else
00156     {
00157         node->s = node->As / node->A;
00158         model->node_discharge_right(node);

```

```

00158         node->u = node->Q / node->A;
00159         node->T = node->Q * node->s;
00160         model->node_diffusion(node);
00161         node->KxA = node->Kx * node->A;
00162     }
00163     node->zs = node->zb + node->h;
00164     model->node_infiltration(node);
00165     node->Pi = node->P * node->i;
00166 }
00167
00177 void model_node_parameters_left_diffusive (
    Model *model, Node *node)
00178 {
00179     node_depth(node);
00180     node_width(node);
00181     node_perimeter(node);
00182     if (node->A <= 0.)
00183     {
00184         node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00185     }
00186     else if (node->h < model->minimum_depth)
00187     {
00188         node->s = node->As / node->A;
00189         node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00190     }
00191     else
00192     {
00193         node->s = node->As / node->A;
00194         node->u = node->Q / node->A;
00195         node->T = node->Q * node->s;
00196         model->node_diffusion(node);
00197         node->KxA = node->Kx * node->A;
00198     }
00199     node->zs = node->zb + node->h;
00200     model->node_infiltration(node);
00201     node->Pi = node->P * node->i;
00202 }
00203
00212 double node_ldt_max_diffusive(Node *node)
00213 {
00214     double u;
00215     u = 5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
00216         / (node->B * node->P);
00217     if (node->u > 0.)
00218         u += node->A * pow(node->A / node->P, 4./3.)
00219             / (node->friction_coefficient[0] * node->friction_coefficient[0]
00220                * node->u * node->dx);
00221     return u / node->dx;
00222 }
00223
00231 void node_flows_diffusive(Node *model)
00232 {
00233     Node *node2 = model + 1;
00234     model->dQ = node2->Q - model->Q;
00235     model->dT = node2->T - model->T;
00236 }
00237
00246 double model_inlet_dtmax_diffusive(Model *model)
00247 {
00248     double A, Q, h, B, c;
00249     Node *node = model->mesh->node;
00250     Q = hydrogram_discharge(model->channel->water_inlet,
00251                             model->t);
00251     h = node_critical_depth(node, Q);
00252     A = h * (node->B0 + h * node->Z);
00253     B = node->B0 + 2 * h * node->Z;
00254     c = sqrt(G * A / B);
00255     return node->ix / c;
00256 }

```

4.27 model_diffusive.h File Reference

Header file to define the diffusive model.

Functions

- void [node_discharge_centre_diffusive_Manning](#) (Node *node)

Function to calculate the diffusive discharge with the Manning model using centred derivatives.

- void [node_discharge_right_diffusive_Manning](#) (Node *node)
Function to calculate the diffusive discharge with the Manning model using right derivatives.
- void [node_discharge_left_diffusive_Manning](#) (Node *node)
Function to calculate the diffusive discharge with the Manning model using left derivatives.
- void [model_node_parameters_centre_diffusive](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the diffusive model using centred derivatives.
- void [model_node_parameters_right_diffusive](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the diffusive model using right derivatives.
- void [model_node_parameters_left_diffusive](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the diffusive model using left derivatives.
- double [node_1dt_max_diffusive](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the diffusive model.
- void [node_flows_diffusive](#) (Node *node1)
Function to calculate the flux differences in a node with the diffusive model.
- double [model_inlet_dtmx_diffusive](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the diffusive model.

4.27.1 Detailed Description

Header file to define the diffusive model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_diffusive.h](#).

4.27.2 Function Documentation

4.27.2.1 double model_inlet_dtmx_diffusive (Model * model)

Function to calculate the allowed maximum time step size at the inlet with the diffusive model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 246 of file [model_diffusive.c](#).

```
{
    double A, Q, h, B, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    return node->ix / c;
}
```

4.27.2.2 void model_node_parameters_centre_diffusive (Model * model, Node * node)

Function to calculate the numerical parameters of a node with the diffusive model using centred derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 101 of file [model_diffusive.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA
            = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_centre(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_friction(node);
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.27.2.3 void model_node_parameters_left_diffusive (Model * model, Node * node)

Function to calculate the numerical parameters of a node with the diffusive model using left derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 177 of file [model_diffusive.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
}
```



```

    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}

```

4.27.2.4 void model_node_parameters_right_diffusive (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the diffusive model using right derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 140 of file [model_diffusive.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_right(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}

```

4.27.2.5 double node_1dt_max_diffusive (Node * *node*)

Function to calculate the allowed maximum time step size in a node with the diffusive model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 212 of file [model_diffusive.c](#).

```

{
    double u;
    u = 5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
        / (node->B * node->P);
    if (node->u > 0.)
        u += node->A * pow(node->A / node->P, 4./3.)
            / (node->friction_coefficient[0] * node->friction_coefficient[0]
                * node->u * node->dx);
    return u / node->dx;
}

```

4.27.2.6 void node_discharge_centre_diffusive_Manning (Node * node)

Function to calculate the diffusive discharge with the Manning model using centred derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 51 of file [model_diffusive.c](#).

```
{
    double dz;
    dz = (node - 1)->zs - (node + 1)->zs;
    if (dz <= 0.) node->Q = 0.; else
        node->Q = sqrt(dz / ((node - 1)->ix + node->ix)) * node->A
            * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.27.2.7 void node_discharge_left_diffusive_Manning (Node * node)

Function to calculate the diffusive discharge with the Manning model using left derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 83 of file [model_diffusive.c](#).

```
{
    double dz;
    dz = (node - 1)->zs - node->zs;
    if (dz <= 0.) node->Q = 0.; else
        node->Q = sqrt(dz / (node - 1)->ix) * node->A
            * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.27.2.8 void node_discharge_right_diffusive_Manning (Node * node)

Function to calculate the diffusive discharge with the Manning model using right derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 67 of file [model_diffusive.c](#).

```
{
    double dz;
    dz = node->zs - (node + 1)->zs;
    if (dz <= 0.) node->Q = 0.; else
        node->Q = sqrt(dz / node->ix) * node->A * pow(node->A / node->P, 2./3.)
            / node->friction_coefficient[0];
}
```

4.27.2.9 void node_flows_diffusive (Node * node1)

Function to calculate the flux differences in a node with the diffusive model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 231 of file [model_diffusive.c](#).

```
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dT = node2->T - node1->T;
}
```

4.28 model_diffusive.h

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 // in order to prevent multiple definitions
00033 #ifndef MODEL_DIFFUSIVE__H
00034 #define MODEL_DIFFUSIVE__H 1
00035
00036 // member functions
00037
00038 void node_discharge_centre_diffusive_Manning
00039     (Node *node);
00040 void node_discharge_right_diffusive_Manning
00041     (Node *node);
00042 void node_discharge_left_diffusive_Manning
00043     (Node *node);
00044 void model_node_parameters_centre_diffusive
00045     (Model *model, Node *node);
00046 void model_node_parameters_right_diffusive
00047     (Model *model, Node *node);
00048 void model_node_parameters_left_diffusive(
00049     Model *model, Node *node);
00049 double node_ldt_max_diffusive(Node *node);
00050 void node_flows_diffusive(Node *node);
00051 double model_inlet_dtmax_diffusive(Model *model);
00052 #endif
```

4.29 model_diffusive_upwind.c File Reference

Source file to define the upwind numerical model applied to the diffusive model.

```
#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_diffusive_upwind.h"
```

Functions

- void [model_surface_flow_diffusive_upwind](#) (Model *model)

Function to make the surface flow with the upwind numerical scheme.

4.29.1 Detailed Description

Source file to define the upwind numerical model applied to the diffusive model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_diffusive_upwind.c](#).

4.29.2 Function Documentation

4.29.2.1 void [model_surface_flow_diffusive_upwind](#) (Model * *model*)

Function to make the surface flow with the upwind numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 51 of file [model_diffusive_upwind.c](#).

```
{
    int i;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    double inlet_water_contribution, inlet_solute_contribution;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    for (i = 0; ++i < mesh->n;)
    {
        model->node_flows(node + i - 1);
        node[i].A -= model->dt * node[i - 1].dQ / node[i].dx;
        node[i].As -= model->dt * node[i - 1].dT / node[i].dx;
    }
    node[0].A -= inlet_water_contribution / node[0].dx;
    node[0].As -= inlet_solute_contribution / node[0].dx;
}
```

4.30 model_diffusive_upwind.c

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012
00013     2. Redistributions in binary form must reproduce the above copyright
00014         notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 #include <stdio.h>
00033 #include <math.h>
00034 #include "config.h"
00035 #include "channel.h"
00036 #include "node.h"
00037 #include "mesh.h"
00038 #include "model.h"
00039 #include "model_diffusive_upwind.h"
00040
00041 void model_surface_flow_diffusive_upwind(
00042     Model *model)
00043 {
00044     int i;
00045     Mesh *mesh = model->mesh;
00046     Node *node = mesh->node;
00047     double inlet_water_contribution, inlet_solute_contribution;
00048     inlet_water_contribution = model->dt * node[0].Q;
00049     inlet_solute_contribution = model->dt * node[0].T;
00050     for (i = 0; ++i < mesh->n;)
00051     {
00052         model->node_flows(node + i - 1);
00053         node[i].A -= model->dt * node[i - 1].dQ / node[i].dx;
00054         node[i].As -= model->dt * node[i - 1].dT / node[i].dx;
00055     }
00056     node[0].A -= inlet_water_contribution / node[0].dx;
00057     node[0].As -= inlet_solute_contribution / node[0].dx;
00058 }

```

4.31 model_diffusive_upwind.h File Reference

Header file to define the upwind numerical model applied to the diffusive model.

Functions

- void `model_surface_flow_diffusive_upwind` (Model *model)

Function to make the surface flow with the upwind numerical scheme.

4.31.1 Detailed Description

Header file to define the upwind numerical model applied to the diffusive model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_diffusive_upwind.h](#).

4.31.2 Function Documentation

4.31.2.1 void model_surface_flow_diffusive_upwind (Model * model)

Function to make the surface flow with the upwind numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 51 of file [model_diffusive_upwind.c](#).

```
{
    int i;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    double inlet_water_contribution, inlet_solute_contribution;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    for (i = 0; ++i < mesh->n;)
    {
        model->node_flows(node + i - 1);
        node[i].A -= model->dt * node[i - 1].dQ / node[i].dx;
        node[i].As -= model->dt * node[i - 1].dT / node[i].dx;
    }
    node[0].A -= inlet_water_contribution / node[0].dx;
    node[0].As -= inlet_solute_contribution / node[0].dx;
}
```

4.32 model_diffusive_upwind.h

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012
00013     2. Redistributions in binary form must reproduce the above copyright
00014         notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
```

```

00020 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
      INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00037 // in order to prevent multiple definitions
00038 #ifndef MODEL_DIFFUSIVE_UPWIND__H
00039 #define MODEL_DIFFUSIVE_UPWIND__H 1
00040
00041 // member functions
00042
00043 void model_surface_flow_diffusive_upwind(
      Model *model);
00044
00045 #endif

```

4.33 model_kinematic.c File Reference

Source file to define the kinematic model.

```

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_kinematic.h"

```

Functions

- void [node_discharge_centre_kinematic_Manning](#) (Node *node)
Function to calculate the kinematic discharge with the Manning model using centred derivatives.
- void [node_discharge_right_kinematic_Manning](#) (Node *node)
Function to calculate the kinematic discharge with the Manning model using right derivatives.
- void [node_discharge_left_kinematic_Manning](#) (Node *node)
Function to calculate the kinematic discharge with the Manning model using left derivatives.
- void [model_node_parameters_centre_kinematic](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the kinematic model using centred derivatives.
- void [model_node_parameters_right_kinematic](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the kinematic model using right derivatives.
- void [model_node_parameters_left_kinematic](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the kinematic model using left derivatives.
- double [node_1dt_max_kinematic](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the kinematic model.
- void [node_flows_kinematic](#) (Node *node1)
Function to calculate the flux differences in a node with the kinematic model.
- double [model_inlet_dtmx_kinematic](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the kinematic model.

4.33.1 Detailed Description

Source file to define the kinematic model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_kinematic.c](#).

4.33.2 Function Documentation**4.33.2.1 double model_inlet_dtxmax_kinematic (Model * *model*)**

Function to calculate the allowed maximum time step size at the inlet with the kinematic model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 233 of file [model_kinematic.c](#).

```
{
    double A, Q, h, B, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    return node->ix / c;
}
```

4.33.2.2 void model_node_parameters_centre_kinematic (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the kinematic model using centred derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 93 of file [model_kinematic.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA
            = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA = 0.;
    }
    else
```



```

{
    node->s = node->As / node->A;
    model->node_discharge_centre(node);
    node->u = node->Q / node->A;
    node->T = node->Q * node->s;
    model->node_friction(node);
    model->node_diffusion(node);
    node->KxA = node->Kx * node->A;
}
node->zs = node->zb + node->h;
model->node_infiltration(node);
node->Pi = node->P * node->i;
}

```

4.33.2.3 void model_node_parameters_left_kinematic (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the kinematic model using left derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 169 of file [model_kinematic.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_left(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}

```

4.33.2.4 void model_node_parameters_right_kinematic (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the kinematic model using right derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 132 of file [model_kinematic.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
}

```

```

    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_right(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}

```

4.33.2.5 double node_1dt_max_kinematic (Node * node)

Function to calculate the allowed maximum time step size in a node with the kinematic model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 205 of file [model_kinematic.c](#).

```

{
    return (5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
        / (node->B * node->P)) / node->dx;
}

```

4.33.2.6 void node_discharge_centre_kinematic_Manning (Node * node)

Function to calculate the kinematic discharge with the Manning model using centred derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 51 of file [model_kinematic.c](#).

```

{
    node->Q = sqrt(((node - 1)->zb - (node + 1)->zb)
        / ((node - 1)->ix + node->ix)) * node->A * pow(node->A / node->P, 2./
        3.)
        / node->friction_coefficient[0];
}

```

4.33.2.7 void node_discharge_left_kinematic_Manning (Node * node)

Function to calculate the kinematic discharge with the Manning model using left derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 78 of file [model_kinematic.c](#).

```
{
    node->Q = sqrt(((node - 1)->zb - node->zb) / (node - 1)->ix) * node->A
        * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.33.2.8 void node_discharge_right_kinematic_Manning (Node * node)

Function to calculate the kinematic discharge with the Manning model using right derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 65 of file [model_kinematic.c](#).

```
{
    node->Q = sqrt((node->zb - (node + 1)->zb) / node->ix) * node->A
        * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.33.2.9 void node_flows_kinematic (Node * node1)

Function to calculate the flux differences in a node with the kinematic model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 218 of file [model_kinematic.c](#).

```
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dT = node2->T - node1->T;
}
```

4.34 model_kinematic.c

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012
00013     2. Redistributions in binary form must reproduce the above copyright
00014         notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018     OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021     EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023     INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
```

```

00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00035 #include <stdio.h>
00036 #include <math.h>
00037 #include "config.h"
00038 #include "channel.h"
00039 #include "node.h"
00040 #include "mesh.h"
00041 #include "model.h"
00042 #include "model_kinematic.h"
00043
00051 void node_discharge_centre_kinematic_Manning
00052 (Node *node)
00053 {
00054     node->Q = sqrt(((node - 1)->zb - (node + 1)->zb)
00055         / ((node - 1)->ix + node->ix)) * node->A * pow(node->A / node->P, 2./3.
00056     )
00057     / node->friction_coefficient[0];
00058 }
00059
00065 void node_discharge_right_kinematic_Manning
00066 (Node *node)
00067 {
00068     node->Q = sqrt((node->zb - (node + 1)->zb) / node->ix) * node->A
00069     * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
00070 }
00071
00078 void node_discharge_left_kinematic_Manning
00079 (Node *node)
00080 {
00081     node->Q = sqrt(((node - 1)->zb - node->zb) / (node - 1)->ix) * node->A
00082     * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
00083 }
00084
00093 void model_node_parameters_centre_kinematic
00094 (Model *model, Node *node)
00095 {
00096     node_depth(node);
00097     node_width(node);
00098     node_perimeter(node);
00099     if (node->A <= 0.)
00100     {
00101         node->s = node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA
00102         = 0.;
00103     }
00104     else if (node->h < model->minimum_depth)
00105     {
00106         node->s = node->As / node->A;
00107         node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA = 0.;
00108     }
00109     else
00110     {
00111         node->s = node->As / node->A;
00112         model->node_discharge_centre(node);
00113         node->u = node->Q / node->A;
00114         node->T = node->Q * node->s;
00115         model->node_friction(node);
00116         model->node_diffusion(node);
00117         node->KxA = node->Kx * node->A;
00118     }
00119     node->zs = node->zb + node->h;
00120     model->node_infiltration(node);
00121     node->Pi = node->P * node->i;
00122 }
00123
00132 void model_node_parameters_right_kinematic
00133 (Model *model, Node *node)
00134 {
00135     node_depth(node);
00136     node_width(node);
00137     node_perimeter(node);
00138     if (node->A <= 0.)
00139     {
00140         node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00141     }
00142     else if (node->h < model->minimum_depth)
00143     {
00144         node->s = node->As / node->A;
00145         node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00146     }
00147     else
00148     {

```

```

00148         node->s = node->As / node->A;
00149         model->node_discharge_right(node);
00150         node->u = node->Q / node->A;
00151         node->T = node->Q * node->s;
00152         model->node_diffusion(node);
00153         node->KxA = node->Kx * node->A;
00154     }
00155     node->zs = node->zb + node->h;
00156     model->node_infiltration(node);
00157     node->Pi = node->P * node->i;
00158 }
00159
00169 void model_node_parameters_left_kinematic(
    Model *model, Node *node)
00170 {
00171     node_depth(node);
00172     node_width(node);
00173     node_perimeter(node);
00174     if (node->A <= 0.)
00175     {
00176         node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00177     }
00178     else if (node->h < model->minimum_depth)
00179     {
00180         node->s = node->As / node->A;
00181         node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
00182     }
00183     else
00184     {
00185         node->s = node->As / node->A;
00186         model->node_discharge_left(node);
00187         node->u = node->Q / node->A;
00188         node->T = node->Q * node->s;
00189         model->node_diffusion(node);
00190         node->KxA = node->Kx * node->A;
00191     }
00192     node->zs = node->zb + node->h;
00193     model->node_infiltration(node);
00194     node->Pi = node->P * node->i;
00195 }
00196
00205 double node_ldt_max_kinematic(Node *node)
00206 {
00207     return (5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
00208         / (node->B * node->P)) / node->dx;
00209 }
00210
00218 void node_flows_kinematic(Node *model)
00219 {
00220     Node *node2 = model + 1;
00221     model->dQ = node2->Q - model->Q;
00222     model->dT = node2->T - model->T;
00223 }
00224
00233 double model_inlet_dtmax_kinematic(Model *model)
00234 {
00235     double A, Q, h, B, c;
00236     Node *node = model->mesh->node;
00237     Q = hydrogram_discharge(model->channel->water_inlet,
    model->t);
00238     h = node_critical_depth(node, Q);
00239     A = h * (node->B0 + h * node->Z);
00240     B = node->B0 + 2 * h * node->Z;
00241     c = sqrt(G * A / B);
00242     return node->ix / c;
00243 }

```

4.35 model_kinematic.h File Reference

Header file to define the kinematic model.

Functions

- void [node_discharge_centre_kinematic_Manning](#) (Node *node)
Function to calculate the kinematic discharge with the Manning model using centred derivatives.
- void [node_discharge_right_kinematic_Manning](#) (Node *node)
Function to calculate the kinematic discharge with the Manning model using right derivatives.

- void [node_discharge_left_kinematic_Manning](#) (Node *node)
Function to calculate the kinematic discharge with the Manning model using left derivatives.
- void [model_node_parameters_centre_kinematic](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the kinematic model using centred derivatives.
- void [model_node_parameters_right_kinematic](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the kinematic model using right derivatives.
- void [model_node_parameters_left_kinematic](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the kinematic model using left derivatives.
- double [node_1dt_max_kinematic](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the kinematic model.
- void [node_flows_kinematic](#) (Node *node1)
Function to calculate the flux differences in a node with the kinematic model.
- double [model_inlet_dtmmax_kinematic](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the kinematic model.

4.35.1 Detailed Description

Header file to define the kinematic model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_kinematic.h](#).

4.35.2 Function Documentation

4.35.2.1 double [model_inlet_dtmmax_kinematic](#) (Model * *model*)

Function to calculate the allowed maximum time step size at the inlet with the kinematic model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 233 of file [model_kinematic.c](#).

```
{
    double A, Q, h, B, c;
    Node *node = model->mesh->node;
    Q = hydrogram\_discharge(model->channel->water_inlet,
        model->t);
    h = node\_critical\_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    return node->ix / c;
}
```

4.35.2.2 void model_node_parameters_centre_kinematic (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the kinematic model using centred derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 93 of file [model_kinematic.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA
            = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Sf = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_centre(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_friction(node);
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}
```

4.35.2.3 void model_node_parameters_left_kinematic (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the kinematic model using left derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 169 of file [model_kinematic.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_left(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
}
```

```

    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}

```

4.35.2.4 void model_node_parameters_right_kinematic (Model * *model*, Node * *node*)

Function to calculate the numerical parameters of a node with the kinematic model using right derivatives.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 132 of file [model_kinematic.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        model->node_discharge_right(node);
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
}

```

4.35.2.5 double node_1dt_max_kinematic (Node * *node*)

Function to calculate the allowed maximum time step size in a node with the kinematic model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 205 of file [model_kinematic.c](#).

```

{
    return (5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
            / (node->B * node->P)) / node->dx;
}

```

4.35.2.6 void node_discharge_centre_kinematic_Manning (Node * *node*)

Function to calculate the kinematic discharge with the Manning model using centred derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 51 of file [model_kinematic.c](#).

```
{
    node->Q = sqrt(((node - 1)->zb - (node + 1)->zb)
        / ((node - 1)->ix + node->ix) * node->A * pow(node->A / node->P, 2./
        3.))
        / node->friction_coefficient[0];
}
```

4.35.2.7 void node_discharge_left_kinematic_Manning (Node * *node*)

Function to calculate the kinematic discharge with the Manning model using left derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 78 of file [model_kinematic.c](#).

```
{
    node->Q = sqrt(((node - 1)->zb - node->zb) / (node - 1)->ix) * node->A
        * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.35.2.8 void node_discharge_right_kinematic_Manning (Node * *node*)

Function to calculate the kinematic discharge with the Manning model using right derivatives.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 65 of file [model_kinematic.c](#).

```
{
    node->Q = sqrt((node->zb - (node + 1)->zb) / node->ix) * node->A
        * pow(node->A / node->P, 2./3.) / node->friction_coefficient[0];
}
```

4.35.2.9 void node_flows_kinematic (Node * *node1*)

Function to calculate the flux differences in a node with the kinematic model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 218 of file [model_kinematic.c](#).

```
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dT = node2->T - node1->T;
}
```

4.36 model_kinematic.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018 OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021 EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023 INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 // in order to prevent multiple definitions
00033 #ifndef MODEL_KINEMATIC__H
00034 #define MODEL_KINEMATIC__H 1
00035
00036 // member functions
00037
00038 void node_discharge_centre_kinematic_Manning
00039     (Node *node);
00040 void node_discharge_right_kinematic_Manning
00041     (Node *node);
00042 void node_discharge_left_kinematic_Manning
00043     (Node *node);
00044 void model_node_parameters_centre_kinematic
00045     (Model *model, Node *node);
00046 void model_node_parameters_right_kinematic
00047     (Model *model, Node *node);
00048 void model_node_parameters_left_kinematic(
00049     Model *model, Node *node);
00050 double node_ldt_max_kinematic(Node *node);
00051 void node_flows_kinematic(Node *node);
00052 double model_inlet_dtmax_kinematic(Model *model);
00053 #endif

```

4.37 model_kinematic_upwind.c File Reference

Source file to define the upwind numerical model applied to the kinematic model.

```

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_kinematic_upwind.h"

```

Functions

- void [model_surface_flow_kinematic_upwind](#) (Model *model)
Function to make the surface flow with the upwind numerical scheme.

4.37.1 Detailed Description

Source file to define the upwind numerical model applied to the kinematic model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_kinematic_upwind.c](#).

4.37.2 Function Documentation

4.37.2.1 void model_surface_flow_kinematic_upwind (Model * model)

Function to make the surface flow with the upwind numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 51 of file [model_kinematic_upwind.c](#).

```
{
    int i;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    double inlet_water_contribution, inlet_solute_contribution;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    for (i = 0; ++i < mesh->n;)
    {
        model->node_flows(node + i - 1);
        node[i].A -= model->dt * node[i - 1].dQ / node[i].dx;
        node[i].As -= model->dt * node[i - 1].dT / node[i].dx;
    }
    node[0].A -= inlet_water_contribution / node[0].dx;
    node[0].As -= inlet_solute_contribution / node[0].dx;
}
```

4.38 model_kinematic_upwind.c

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012
00012     2. Redistributions in binary form must reproduce the above copyright
```

```

        notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS`` AND ANY EXPRESS
        OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
        EVENT
00020 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
        INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00036 #include <stdio.h>
00037 #include <math.h>
00038 #include "config.h"
00039 #include "channel.h"
00040 #include "node.h"
00041 #include "mesh.h"
00042 #include "model.h"
00043 #include "model_kinematic_upwind.h"
00044
00051 void model_surface_flow_kinematic_upwind(
        Model *model)
00052 {
00053     int i;
00054     Mesh *mesh = model->mesh;
00055     Node *node = mesh->node;
00056     double inlet_water_contribution, inlet_solute_contribution;
00057     inlet_water_contribution = model->dt * node[0].Q;
00058     inlet_solute_contribution = model->dt * node[0].T;
00059     for (i = 0; ++i < mesh->n;)
00060     {
00061         model->node_flows(node + i - 1);
00062         node[i].A -= model->dt * node[i - 1].dQ / node[i].dx;
00063         node[i].As -= model->dt * node[i - 1].dT / node[i].dx;
00064     }
00065     node[0].A -= inlet_water_contribution / node[0].dx;
00066     node[0].As -= inlet_solute_contribution / node[0].dx;
00067 }

```

4.39 model_kinematic_upwind.h File Reference

Header file to define the upwind numerical model applied to the kinematic model.

Functions

- void [model_surface_flow_kinematic_upwind](#) (Model *model)
Function to make the surface flow with the upwind numerical scheme.

4.39.1 Detailed Description

Header file to define the upwind numerical model applied to the kinematic model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_kinematic_upwind.h](#).

4.39.2 Function Documentation

4.39.2.1 void model_surface_flow_kinematic_upwind (Model * model)

Function to make the surface flow with the upwind numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 51 of file [model_kinematic_upwind.c](#).

```
{
    int i;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    double inlet_water_contribution, inlet_solute_contribution;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    for (i = 0; ++i < mesh->n;)
    {
        model->node_flows(node + i - 1);
        node[i].A -= model->dt * node[i - 1].dQ / node[i].dx;
        node[i].As -= model->dt * node[i - 1].dT / node[i].dx;
    }
    node[0].A -= inlet_water_contribution / node[0].dx;
    node[0].As -= inlet_solute_contribution / node[0].dx;
}
```

4.40 model_kinematic_upwind.h

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
00015         notice,
00016         this list of conditions and the following disclaimer in the
00017         documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00020 OR
00021 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00023 EVENT
00024 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00025 INDIRECT,
00026 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00027 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00030 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00031 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033 // in order to prevent multiple definitions
00034 #ifndef MODEL_KINEMATIC_UPWIND__H
00035 #define MODEL_KINEMATIC_UPWIND__H 1
00036
00037 // member functions
00038
00039 void model_surface_flow_kinematic_upwind(
00040     Model *model);
00041
00042 #endif
```

4.41 model_zero_inertia.c File Reference

Source file to define the zero inertia model.

```
#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_zero_inertia.h"
```

Functions

- void [model_node_parameters_zero_inertia](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the zero-inertia model.
- double [node_1dt_max_zero_inertia](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the zero-inertia model.
- void [node_flows_zero_inertia](#) (Node *node1)
Function to calculate the flux differences in a node with the zero-inertia model.
- double [model_inlet_dtmax_zero_inertia](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the zero-inertia model.

4.41.1 Detailed Description

Source file to define the zero inertia model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_zero_inertia.c](#).

4.41.2 Function Documentation

4.41.2.1 double model_inlet_dtmax_zero_inertia (Model * model)

Function to calculate the allowed maximum time step size at the inlet with the zero-inertia model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line 123 of file [model_zero_inertia.c](#).

```

{
    double A, Q, h, B, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    return node->ix / c;
}

```

4.41.2.2 void model_node_parameters_zero_inertia (Model * model, Node * node)

Function to calculate the numerical parameters of a node with the zero-inertia model.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line 53 of file [model_zero_inertia.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    node_critical_velocity(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->Sf = node->T = node->Kx = node->KxA
            = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->Sf = node->T = node->Kx = node->KxA = 0.;
    }
    else
    {
        node->s = node->As / node->A;
        node->u = node->Q / node->A;
        node->T = node->Q * node->s;
        model->node_friction(node);
        model->node_diffusion(node);
        node->KxA = node->Kx * node->A;
    }
    node->zs = node->zb + node->h;
    node->l1 = fmax(node->c, fabs(node->u));
    model->node_infiltration(node);
    node->Pi = node->P * node->i;
    if (isnan(node->P)) printf("P=%lg\n", node->P);
    if (isnan(node->i)) printf("i=%lg\n", node->i);
}

```

4.41.2.3 double node_1dt_max_zero_inertia (Node * node)

Function to calculate the allowed maximum time step size in a node with the zero-inertia model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 94 of file [model_zero_inertia.c](#).

```

{

```

```

    return node->l1 / node->dx;
}

```

4.41.2.4 void node_flows_zero_inertia (Node * node1)

Function to calculate the flux differences in a node with the zero-inertia model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 106 of file [model_zero_inertia.c](#).

```

{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dF = G * 0.5 * (node2->A + node1->A)
        * (node2->zS - node1->zS + 0.5 * (node2->Sf + node1->Sf) * node1->ix);
    node1->dT = node2->T - node1->T;
}

```

4.42 model_zero_inertia.c

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
00015        notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00020     OR
00021 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00023     EVENT
00024 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00025     INDIRECT,
00026 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00027     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00030     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00031     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033 #include <stdio.h>
00034 #include <math.h>
00035 #include "config.h"
00036 #include "channel.h"
00037 #include "node.h"
00038 #include "mesh.h"
00039 #include "model.h"
00040 #include "model_zero_inertia.h"
00041
00042 void model_node_parameters_zero_inertia(Model
00043     *model, Node *node)
00044 {
00045     node_depth(node);
00046     node_width(node);
00047     node_perimeter(node);
00048     node_critical_velocity(node);
00049     if (node->A <= 0.)
00050     {
00051         node->s = node->Q = node->u = node->Sf = node->T = node->Kx = node->KxA

```



```

00062         = 0.;
00063     }
00064     else if (node->h < model->minimum_depth)
00065     {
00066         node->s = node->As / node->A;
00067         node->Q = node->u = node->Sf = node->T = node->Kx = node->KxA = 0.;
00068     }
00069     else
00070     {
00071         node->s = node->As / node->A;
00072         node->u = node->Q / node->A;
00073         node->T = node->Q * node->s;
00074         model->node_friction(node);
00075         model->node_diffusion(node);
00076         node->KxA = node->Kx * node->A;
00077     }
00078     node->zs = node->zb + node->h;
00079     node->l1 = fmax(node->c, fabs(node->u));
00080     model->node_infiltration(node);
00081     node->Pi = node->P * node->i;
00082     if (isnan(node->P)) printf("P=%lg\n", node->P);
00083     if (isnan(node->i)) printf("i=%lg\n", node->i);
00084 }
00085
00094 double node_1dt_max_zero_inertia(Node *node)
00095 {
00096     return node->l1 / node->dx;
00097 }
00098
00106 void node_flows_zero_inertia(Node *node1)
00107 {
00108     Node *node2 = node1 + 1;
00109     node1->dQ = node2->Q - node1->Q;
00110     node1->dF = G * 0.5 * (node2->A + node1->A)
00111         * (node2->zs - node1->zs + 0.5 * (node2->Sf + node1->Sf) * node1->ix);
00112     node1->dT = node2->T - node1->T;
00113 }
00114
00123 double model_inlet_dtmax_zero_inertia(Model *
model)
00124 {
00125     double A, Q, h, B, c;
00126     Node *node = model->mesh->node;
00127     Q = hydrogram\_discharge(model->channel->water_inlet,
model->t);
00128     h = node\_critical\_depth(node, Q);
00129     A = h * (node->B0 + h * node->Z);
00130     B = node->B0 + 2 * h * node->Z;
00131     c = sqrt(G * A / B);
00132     return node->ix / c;
00133 }

```

4.43 model_zero_inertia.h File Reference

Header file to define the zero inertia model.

Functions

- void [model_node_parameters_zero_inertia](#) (Model *model, Node *node)
Function to calculate the numerical parameters of a node with the zero-inertia model.
- double [node_1dt_max_zero_inertia](#) (Node *node)
Function to calculate the allowed maximum time step size in a node with the zero-inertia model.
- void [node_flows_zero_inertia](#) (Node *node1)
Function to calculate the flux differences in a node with the zero-inertia model.
- double [model_inlet_dtmax_zero_inertia](#) (Model *model)
Function to calculate the allowed maximum time step size at the inlet with the zero-inertia model.

4.43.1 Detailed Description

Header file to define the zero inertia model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_zero_inertia.h](#).

4.43.2 Function Documentation**4.43.2.1 double model_inlet_dtmx_zero_inertia (Model * model)**

Function to calculate the allowed maximum time step size at the inlet with the zero-inertia model.

Parameters

<i>model</i>	model struct.
--------------	---------------

Returns

allowed maximum time step size.

Definition at line [123](#) of file [model_zero_inertia.c](#).

```
{
    double A, Q, h, B, c;
    Node *node = model->mesh->node;
    Q = hydrogram_discharge(model->channel->water_inlet,
        model->t);
    h = node_critical_depth(node, Q);
    A = h * (node->B0 + h * node->Z);
    B = node->B0 + 2 * h * node->Z;
    c = sqrt(G * A / B);
    return node->ix / c;
}
```

4.43.2.2 void model_node_parameters_zero_inertia (Model * model, Node * node)

Function to calculate the numerical parameters of a node with the zero-inertia model.

Parameters

<i>model</i>	model struct.
<i>node</i>	node struct.

Definition at line [53](#) of file [model_zero_inertia.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_perimeter(node);
    node_critical_velocity(node);
    if (node->A <= 0.)
    {
        node->s = node->Q = node->u = node->Sf = node->T = node->Kx = node->KxA
            = 0.;
    }
    else if (node->h < model->minimum_depth)
    {
        node->s = node->As / node->A;
        node->Q = node->u = node->Sf = node->T = node->Kx = node->KxA = 0.;
    }
}
```

```

else
{
    node->s = node->As / node->A;
    node->u = node->Q / node->A;
    node->T = node->Q * node->s;
    model->node_friction(node);
    model->node_diffusion(node);
    node->KxA = node->Kx * node->A;
}
node->zs = node->zb + node->h;
node->l1 = fmax(node->c, fabs(node->u));
model->node_infiltration(node);
node->Pi = node->P * node->i;
if (isnan(node->P)) printf("P=%lg\n", node->P);
if (isnan(node->i)) printf("i=%lg\n", node->i);
}

```

4.43.2.3 double node_1dt_max_zero_inertia (Node * node)

Function to calculate the allowed maximum time step size in a node with the zero-inertia model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Returns

inverse of the allowed maximum time step size.

Definition at line 94 of file [model_zero_inertia.c](#).

```

{
    return node->l1 / node->dx;
}

```

4.43.2.4 void node_flows_zero_inertia (Node * node1)

Function to calculate the flux differences in a node with the zero-inertia model.

Parameters

<i>node1</i>	node struct.
--------------	--------------

Definition at line 106 of file [model_zero_inertia.c](#).

```

{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dF = G * 0.5 * (node2->A + node1->A)
        * (node2->zs - node1->zs + 0.5 * (node2->Sf + node1->Sf) * node1->ix);
    node1->dT = node2->T - node1->T;
}

```

4.44 model_zero_inertia.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burquete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:

```

```

00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012
00013     2. Redistributions in binary form must reproduce the above copyright
        notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
        OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
        EVENT
00020 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
        INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00036 // in order to prevent multiple definitions
00037 #ifndef MODEL_ZERO_INERTIA__H
00038 #define MODEL_ZERO_INERTIA__H 1
00039
00040 // member functions
00041
00042 void model_node_parameters_zero_inertia(Model
        *model, Node *node);
00043 double node_ldt_max_zero_inertia(Node *node);
00044 void node_flows_zero_inertia(Node *node1);
00045 double model_inlet_dtmax_zero_inertia(Model *
        model);
00046
00047 #endif

```

4.45 model_zero_inertia_LaxFriedrichs.c File Reference

Source file to define the Lax-Friedrichs numerical model applied to the zero-inertia model.

```

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
#include "mesh.h"
#include "model.h"
#include "model_zero_inertia_LaxFriedrichs.h"

```

Functions

- void `model_surface_flow_zero_inertia_LaxFriedrichs` (Model *model)
Function to make the surface flow with the Lax-Friedrichs numerical scheme.

4.45.1 Detailed Description

Source file to define the Lax-Friedrichs numerical model applied to the zero-inertia model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_zero_inertia_LaxFriedrichs.c](#).

4.45.2 Function Documentation

4.45.2.1 void model_surface_flow_zero_inertia_LaxFriedrichs (Model * model)

Function to make the surface flow with the Lax-Friedrichs numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 52 of file [model_zero_inertia_LaxFriedrichs.c](#).

```
{
    int i, n1;
    double k1, k2, inlet_water_contribution, inlet_solute_contribution;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        model->node_flows(node + i);
        node[i].dQr = node[i].dFr = node[i].dTr = node[i].dQl = node[i].
dFl
        = node[i].dTl = 0;
        if (node[i].h <= model->minimum_depth &&
            node[i + 1].h <= model->minimum_depth)
            continue;

        // wave decomposition

        node[i].dQr = node[i].dQl = 0.5 * node[i].dQ;
        node[i].dFr = node[i].dFl = 0.5 * node[i].dF;
        node[i].dTr = node[i].dTl = 0.5 * node[i].dT;

        // artificial viscosity

        k1 = 0.5 * fmax(node[i + 1].l1, node[i].l1);
        k2 = k1 * (node[i + 1].A - node[i].A);
        node[i].dQl += k2;
        node[i].dQr -= k2;
        k2 = k1 * node[i].dQ;
        node[i].dFl += k2;
        node[i].dFr -= k2;
        k2 = k1 * (node[i + 1].As - node[i].As);
        node[i].dTl += k2;
        node[i].dTr -= k2;
    }

    // variables actualization
    for (i = 0; i < n1; ++i)
    {
        node[i].A -= model->dt * node[i].dQr / node[i].dx;
        node[i].Q -= model->dt * node[i].dFr / node[i].dx;
        node[i].As -= model->dt * node[i].dTr / node[i].dx;
        node[i + 1].A -= model->dt * node[i].dQl / node[i + 1].dx;
        node[i + 1].Q -= model->dt * node[i].dFl / node[i + 1].dx;
        node[i + 1].As -= model->dt * node[i].dTl / node[i + 1].dx;
    }
    node[0].A -= inlet_water_contribution / node[0].dx;
    node[0].As -= inlet_solute_contribution / node[0].dx;
}
```

4.46 model_zero_inertia_LaxFriedrichs.c

00001 /*

```

00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009     are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
00015         notice,
00016         this list of conditions and the following disclaimer in the
00017         documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00020     OR
00021     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022     MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00023     EVENT
00024     SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00025     INDIRECT,
00026     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00027     LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028     PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00030     OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00031     ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #include <stdio.h>
00035 #include <math.h>
00036 #include "config.h"
00037 #include "channel.h"
00038 #include "node.h"
00039 #include "mesh.h"
00040 #include "model.h"
00041 #include "model_zero_inertia_LaxFriedrichs.h"
00042
00043 void model_surface_flow_zero_inertia_LaxFriedrichs
00044     (Model *model)
00045 {
00046     int i, nl;
00047     double k1, k2, inlet_water_contribution, inlet_solute_contribution;
00048     Mesh *mesh = model->mesh;
00049     Node *node = mesh->node;
00050     inlet_water_contribution = model->dt * node[0].Q;
00051     inlet_solute_contribution = model->dt * node[0].T;
00052     nl = mesh->n - 1;
00053     for (i = 0; i < nl; ++i)
00054     {
00055         model->node_flows(node + i);
00056         node[i].dQr = node[i].dFr = node[i].dTr = node[i].dQl = node[i].dFl
00057             = node[i].dTl = 0;
00058         if (node[i].h <= model->minimum_depth &&
00059             node[i + 1].h <= model->minimum_depth)
00060             continue;
00061
00062         // wave decomposition
00063
00064         node[i].dQr = node[i].dQl = 0.5 * node[i].dQ;
00065         node[i].dFr = node[i].dFl = 0.5 * node[i].dF;
00066         node[i].dTr = node[i].dTl = 0.5 * node[i].dT;
00067
00068         // artificial viscosity
00069
00070         k1 = 0.5 * fmax(node[i + 1].ll, node[i].ll);
00071         k2 = k1 * (node[i + 1].A - node[i].A);
00072         node[i].dQl += k2;
00073         node[i].dQr -= k2;
00074         k2 = k1 * node[i].dQ;
00075         node[i].dFl += k2;
00076         node[i].dFr -= k2;
00077         k2 = k1 * (node[i + 1].As - node[i].As);
00078         node[i].dTl += k2;
00079         node[i].dTr -= k2;
00080     }
00081
00082     // variables actualization
00083
00084     for (i = 0; i < nl; ++i)
00085     {
00086         node[i].A -= model->dt * node[i].dQr / node[i].dx;
00087         node[i].Q -= model->dt * node[i].dFr / node[i].dx;
00088         node[i].As -= model->dt * node[i].dTr / node[i].dx;
00089     }
00090 }

```

```

00097         node[i + 1].A -= model->dt * node[i].dQl / node[i + 1].dx;
00098         node[i + 1].Q -= model->dt * node[i].dFl / node[i + 1].dx;
00099         node[i + 1].As -= model->dt * node[i].dTl / node[i + 1].dx;
00100     }
00101     node[0].A -= inlet_water_contribution / node[0].dx;
00102     node[0].As -= inlet_solute_contribution / node[0].dx;
00103 }

```

4.47 model_zero_inertia_LaxFriedrichs.h File Reference

Header file to define the Lax-Friedrichs numerical model applied to the zero-inertia model.

Functions

- void [model_surface_flow_zero_inertia_LaxFriedrichs](#) (Model *model)
Function to make the surface flow with the Lax-Friedrichs numerical scheme.

4.47.1 Detailed Description

Header file to define the Lax-Friedrichs numerical model applied to the zero-inertia model.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [model_zero_inertia_LaxFriedrichs.h](#).

4.47.2 Function Documentation

4.47.2.1 void model_surface_flow_zero_inertia_LaxFriedrichs (Model * model)

Function to make the surface flow with the Lax-Friedrichs numerical scheme.

Parameters

<i>model</i>	model struct.
--------------	---------------

Definition at line 52 of file [model_zero_inertia_LaxFriedrichs.c](#).

```

{
    int i, n1;
    double k1, k2, inlet_water_contribution, inlet_solute_contribution;
    Mesh *mesh = model->mesh;
    Node *node = mesh->node;
    inlet_water_contribution = model->dt * node[0].Q;
    inlet_solute_contribution = model->dt * node[0].T;
    n1 = mesh->n - 1;
    for (i = 0; i < n1; ++i)
    {
        model->node_flows(node + i);
        node[i].dQr = node[i].dFr = node[i].dTr = node[i].dQl = node[i].
        dFl
            = node[i].dTl = 0;
        if (node[i].h <= model->minimum_depth &&
            node[i + 1].h <= model->minimum_depth)
            continue;

        // wave decomposition

```

```

    node[i].dQr = node[i].dQl = 0.5 * node[i].dQ;
    node[i].dFr = node[i].dFl = 0.5 * node[i].dF;
    node[i].dTr = node[i].dTl = 0.5 * node[i].dT;

    // artificial viscosity

    k1 = 0.5 * fmax(node[i + 1].l1, node[i].l1);
    k2 = k1 * (node[i + 1].A - node[i].A);
    node[i].dQl += k2;
    node[i].dQr -= k2;
    k2 = k1 * node[i].dQ;
    node[i].dFl += k2;
    node[i].dFr -= k2;
    k2 = k1 * (node[i + 1].As - node[i].As);
    node[i].dTl += k2;
    node[i].dTr -= k2;
}

// variables actualization
for (i = 0; i < nl; ++i)
{
    node[i].A -= model->dt * node[i].dQr / node[i].dx;
    node[i].Q -= model->dt * node[i].dFr / node[i].dx;
    node[i].As -= model->dt * node[i].dTr / node[i].dx;
    node[i + 1].A -= model->dt * node[i].dQl / node[i + 1].dx;
    node[i + 1].Q -= model->dt * node[i].dFl / node[i + 1].dx;
    node[i + 1].As -= model->dt * node[i].dTl / node[i + 1].dx;
}
node[0].A -= inlet_water_contribution / node[0].dx;
node[0].As -= inlet_solute_contribution / node[0].dx;
}

```

4.48 model_zero_inertia_LaxFriedrichs.h

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
00015        notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00020 OR
00021 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00023 EVENT
00024 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00025 INDIRECT,
00026 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00027 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00030 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00031 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033 // in order to prevent multiple definitions
00034 #ifndef MODEL_ZERO_INERTIA_LAXFRIEDRICHS__H
00035 #define MODEL_ZERO_INERTIA_LAXFRIEDRICHS__H 1
00036
00037 // member functions
00038 void model_surface_flow_zero_inertia_LaxFriedrichs
00039     (Model *model);
00040
00041 #endif

```


4.49 node.c File Reference

Source file to define a mesh node.

```
#include <stdio.h>
#include <math.h>
#include "config.h"
#include "channel.h"
#include "node.h"
```

Functions

- void [node_depth](#) (Node *node)
Function to calculate the depth in a mesh node.
- void [node_width](#) (Node *node)
Function to calculate the width in a mesh node.
- void [node_perimeter](#) (Node *node)
Function to calculate the wetted perimeter in a mesh node.
- void [node_critical_velocity](#) (Node *node)
—
- void [node_subcritical_discharge](#) (Node *node)
Function to force a subcritical discharge in a mesh node.
- double [node_critical_depth](#) (Node *node, double Q)
Function to calculate the critical depth in a mesh node.
- void [node_friction_Manning](#) (Node *node)
Function to calculate the friction slope with the Manning model.
- void [node_infiltration_KostiakovLewis](#) (Node *node)
Function to calculate the infiltration with the Kostiakov-Lewis model.
- void [node_diffusion_Rutherford](#) (Node *node)
Function to calculate the diffusion coefficient with the Rutherford model.
- void [node_inlet](#) (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2)
Function to calculate the inlet boundary condition.
- void [node_outlet_closed](#) (Node *node)
Function to calculate a closed outlet boundary condition.
- void [node_outlet_open](#) (Node *node)
Function to calculate an open outlet boundary condition.

4.49.1 Detailed Description

Source file to define a mesh node.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [node.c](#).

4.49.2 Function Documentation

4.49.2.1 double node_critical_depth (Node * node, double Q)

Function to calculate the critical depth in a mesh node.

Parameters

<i>node</i>	node struct.
<i>Q</i>	discharge.

Returns

critical depth.

Definition at line 114 of file [node.c](#).

```
{
    double h[3], A[3], B[3], u[3], c[3];
    h[0] = 1.;
    do
    {
        h[0] *= 2;
        A[0] = h[0] * (node->B0 + h[0] * node->Z);
        B[0] = node->B0 + 2 * h[0] * node->Z;
        c[0] = G * A[0] / B[0];
        u[0] = Q / A[0];
        u[0] = u[0] * u[0];
    }
    while (u[0] > c[0]);
    h[1] = h[0];
    do
    {
        h[1] *= 0.5;
        A[1] = h[1] * (node->B0 + h[1] * node->Z);
        B[1] = node->B0 + 2 * h[1] * node->Z;
        c[1] = G * A[1] / B[1];
        u[1] = Q / A[1];
        u[1] = u[1] * u[1];
    }
    while (u[1] < c[1]);
    do
    {
        h[2] = 0.5 * (h[0] + h[1]);
        A[2] = h[2] * (node->B0 + h[2] * node->Z);
        B[2] = node->B0 + 2 * h[2] * node->Z;
        c[2] = G * A[2] / B[2];
        u[2] = Q / A[2];
        u[2] = u[2] * u[2];
        if (u[2] < c[1]) h[0] = h[2]; else h[1] = h[2];
    }
    while (h[0]-h[1] > critical_depth_tolerance);
    return 0.5 * (h[0] + h[1]);
}
```

4.49.2.2 void node_critical_velocity (Node * node)

.

Function to calculate the critical velocity in a mesh node

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 86 of file [node.c](#).

```
{
```

```

    if (node->B > 0.) node->c = sqrt(G * node->A / node->B); else node->c = 0.
    ;
}

```

4.49.2.3 void node_depth (Node * node)

Function to calculate the depth in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 47 of file [node.c](#).

```

{
    if (node->Z == 0.)
        node->h = node->A / node->B0;
    else
        node->h = (sqrt(node->B0 * node->B0 + 4. * node->A * node->Z)
            - node->B0) / (2 * node->Z);
    if (isnan(node->h)) printf("A=%lg B0=%lg Z=%lg\n", node->A, node->B0, node->Z);
}

```

4.49.2.4 void node_diffusion_Rutherford (Node * node)

Function to calculate the diffusion coefficient with the Rutherford model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 191 of file [node.c](#).

```

{
    node->Kx = node->diffusion_coefficient[0]
        * sqrt(G * node->P * node->A * fabs(node->Sf));
}

```

4.49.2.5 void node_friction_Manning (Node * node)

Function to calculate the friction slope with the Manning model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 160 of file [node.c](#).

```

{
    node->Sf = node->friction_coefficient[0] * node->friction_coefficient[0]
        * node->u * fabs(node->u) * pow(node->P / node->A, 4./3.);
}

```

4.49.2.6 void node_infiltration_KostiakovLewis (Node * node)

Function to calculate the infiltration with the Kostiakov-Lewis model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 173 of file [node.c](#).

```
{
    node->i = node->infiltration_coefficient[2];
    if (node->infiltration_coefficient[0] == 0.) return;
    node->i += node->infiltration_coefficient[0] *
        node->infiltration_coefficient[1]
        * pow(node->Ai / (node->infiltration_coefficient[0]
        * node->infiltration_coefficient[3]),
        1. - 1. / node->infiltration_coefficient[1]);
}
```

4.49.2.7 void node_inlet (Node * *node*, Hydrogram * *water*, Hydrogram * *solute*, double *t*, double *t2*)

Function to calculate the inlet boundary condition.

Parameters

<i>node</i>	node struct.
<i>water</i>	water inlet hydrogram.
<i>solute</i>	solute inlet hydrogram.
<i>t</i>	actual time.
<i>t2</i>	next time

Definition at line 212 of file [node.c](#).

```
{
    node->A += hydrogram_integrate(water, t, t2) / node->dx;
    node->As += hydrogram_integrate(solute, t, t2) / node->
        dx;
    node_subcritical_discharge(node);
}
```

4.49.2.8 void node_outlet_closed (Node * *node*)

Function to calculate a closed outlet boundary condition.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 225 of file [node.c](#).

```
{
    node->Q = 0.;
}
```

4.49.2.9 void node_outlet_open (Node * *node*)

Function to calculate an open outlet boundary condition.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 236 of file [node.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_critical_velocity(node);
    node->Q = fmax(node->Q, 1.01 * node->A * node->c);
}

```

4.49.2.10 void node_perimeter (Node * node)

Function to calculate the wetted perimeter in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 74 of file [node.c](#).

```

{
    node->P = node->B0 + 2 * sqrt(1 + node->Z * node->Z) * node->h;
}

```

4.49.2.11 void node_subcritical_discharge (Node * node)

Function to force a subcritical discharge in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 97 of file [node.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_critical_velocity(node);
    node->Q = fmin(node->Q, 0.99 * node->A * node->c);
}

```

4.49.2.12 void node_width (Node * node)

Function to calculate the width in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 63 of file [node.c](#).

```

{
    node->B = node->B0 + 2 * node->Z * node->h;
}

```

4.50 node.c

```

00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.

```

```

00006
00007 Redistribution and use in source and binary forms, with or without
00008 modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011         this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013         notice,
00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
00018 OR
00019 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
00021 EVENT
00022 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
00023 INDIRECT,
00024 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00025 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00028 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00029 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 */
00031
00032 #include <stdio.h>
00033 #include <math.h>
00034 #include "config.h"
00035 #include "channel.h"
00036 #include "node.h"
00037
00038 void node_depth(Node *node)
00039 {
00040     if (node->Z == 0.)
00041         node->h = node->A / node->B0;
00042     else
00043         node->h = (sqrt(node->B0 * node->B0 + 4. * node->A * node->Z)
00044                 - node->B0) / (2 * node->Z);
00045     if (isnan(node->h)) printf("A=%lg B0=%lg Z=%lg\n", node->A, node->B0, node->Z);
00046 }
00047
00048 void node_width(Node *node)
00049 {
00050     node->B = node->B0 + 2 * node->Z * node->h;
00051 }
00052
00053 void node_perimeter(Node *node)
00054 {
00055     node->P = node->B0 + 2 * sqrt(1 + node->Z * node->Z) * node->h;
00056 }
00057
00058 void node_critical_velocity(Node *node)
00059 {
00060     if (node->B > 0.) node->c = sqrt(G * node->A / node->B); else node->c = 0.
00061 }
00062
00063 void node_subcritical_discharge(Node *node)
00064 {
00065     node_depth(node);
00066     node_width(node);
00067     node_critical_velocity(node);
00068     node->Q = fmin(node->Q, 0.99 * node->A * node->c);
00069 }
00070
00071 double node_critical_depth(Node *node, double Q)
00072 {
00073     double h[3], A[3], B[3], u[3], c[3];
00074     h[0] = 1.;
00075     do
00076     {
00077         h[0] *= 2;
00078         A[0] = h[0] * (node->B0 + h[0] * node->Z);
00079         B[0] = node->B0 + 2 * h[0] * node->Z;
00080         c[0] = G * A[0] / B[0];
00081         u[0] = Q / A[0];
00082         u[0] = u[0] * u[0];
00083     }
00084     while (u[0] > c[0]);
00085     h[1] = h[0];
00086     do
00087     {
00088         h[1] *= 0.5;
00089         A[1] = h[1] * (node->B0 + h[1] * node->Z);
00090     }
00091     while (u[0] > c[0]);
00092     return h[1];
00093 }

```

```

00133         B[1] = node->B0 + 2 * h[1] * node->Z;
00134         c[1] = G * A[1] / B[1];
00135         u[1] = Q / A[1];
00136         u[1] = u[1] * u[1];
00137     }
00138     while (u[1] < c[1]);
00139     do
00140     {
00141         h[2] = 0.5 * (h[0] + h[1]);
00142         A[2] = h[2] * (node->B0 + h[2] * node->Z);
00143         B[2] = node->B0 + 2 * h[2] * node->Z;
00144         c[2] = G * A[2] / B[2];
00145         u[2] = Q / A[2];
00146         u[2] = u[2] * u[2];
00147         if (u[2] < c[1]) h[0] = h[2]; else h[1] = h[2];
00148     }
00149     while (h[0]-h[1] > critical_depth_tolerance);
00151     return 0.5 * (h[0] + h[1]);
00152 }
00153
00160 void node_friction_Manning(Node *node)
00161 {
00162     node->Sf = node->friction_coefficient[0] * node->friction_coefficient[0]
00163         * node->u * fabs(node->u) * pow(node->P / node->A, 4./3.);
00164 }
00165
00173 void node_infiltration_KostiakovLewis(Node *
node)
00174 {
00175     node->i = node->infiltration_coefficient[2];
00176     if (node->infiltration_coefficient[0] == 0.) return;
00177     node->i += node->infiltration_coefficient[0] *
node->infiltration_coefficient[1]
00178         * pow(node->Ai / (node->infiltration_coefficient[0]
* node->infiltration_coefficient[3]),
00179             1. - 1. / node->infiltration_coefficient[1]);
00182 }
00183
00191 void node_diffusion_Rutherford(Node *node)
00192 {
00193     node->Kx = node->diffusion_coefficient[0]
00194         * sqrt(G * node->P * node->A * fabs(node->Sf));
00195 }
00196
00211 void node_inlet
00212 (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2)
00213 {
00214     node->A += hydrogram_integrate(water, t, t2) / node->dx;
00215     node->As += hydrogram_integrate(solute, t, t2) / node->
dx;
00216     node_subcritical_discharge(node);
00217 }
00218
00225 void node_outlet_closed(Node *node)
00226 {
00227     node->Q = 0.;
00228 }
00229
00236 void node_outlet_open(Node *node)
00237 {
00238     node_depth(node);
00239     node_width(node);
00240     node_critical_velocity(node);
00241     node->Q = fmax(node->Q, 1.01 * node->A * node->c);
00242 }
00243

```

4.51 node.h File Reference

Header file to define a mesh node.

Data Structures

- struct [Node](#)

Struct to define a mesh node.

Functions

- void [node_depth](#) (Node *node)
Function to calculate the depth in a mesh node.
- void [node_width](#) (Node *node)
Function to calculate the width in a mesh node.
- void [node_perimeter](#) (Node *node)
Function to calculate the wetted perimeter in a mesh node.
- void [node_critical_velocity](#) (Node *node)
–
- void [node_subcritical_discharge](#) (Node *node)
Function to force a subcritical discharge in a mesh node.
- double [node_critical_depth](#) (Node *node, double Q)
Function to calculate the critical depth in a mesh node.
- void [node_friction_Manning](#) (Node *node)
Function to calculate the friction slope with the Manning model.
- void [node_infiltration_KostiakovLewis](#) (Node *node)
Function to calculate the infiltration with the Kostiakov-Lewis model.
- void [node_diffusion_Rutherford](#) (Node *node)
Function to calculate the diffusion coefficient with the Rutherford model.
- void [node_inlet](#) (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2)
Function to calculate the inlet boundary condition.
- void [node_outlet_closed](#) (Node *node)
Function to calculate a closed outlet boundary condition.
- void [node_outlet_open](#) (Node *node)
Function to calculate an open outlet boundary condition.

Variables

- double [critical_depth_tolerance](#)
Accuracy calculating the critical depth.

4.51.1 Detailed Description

Header file to define a mesh node.

Author

Javier Burguete Tolosa.

Copyright

Copyright 2011, Javier Burguete Tolosa.

Definition in file [node.h](#).

4.51.2 Function Documentation

4.51.2.1 double node_critical_depth (Node * node, double Q)

Function to calculate the critical depth in a mesh node.

Parameters

<i>node</i>	node struct.
<i>Q</i>	discharge.

Returns

critical depth.

Definition at line 114 of file [node.c](#).

```
{
    double h[3], A[3], B[3], u[3], c[3];
    h[0] = 1.;
    do
    {
        h[0] *= 2;
        A[0] = h[0] * (node->B0 + h[0] * node->Z);
        B[0] = node->B0 + 2 * h[0] * node->Z;
        c[0] = G * A[0] / B[0];
        u[0] = Q / A[0];
        u[0] = u[0] * u[0];
    }
    while (u[0] > c[0]);
    h[1] = h[0];
    do
    {
        h[1] *= 0.5;
        A[1] = h[1] * (node->B0 + h[1] * node->Z);
        B[1] = node->B0 + 2 * h[1] * node->Z;
        c[1] = G * A[1] / B[1];
        u[1] = Q / A[1];
        u[1] = u[1] * u[1];
    }
    while (u[1] < c[1]);
    do
    {
        h[2] = 0.5 * (h[0] + h[1]);
        A[2] = h[2] * (node->B0 + h[2] * node->Z);
        B[2] = node->B0 + 2 * h[2] * node->Z;
        c[2] = G * A[2] / B[2];
        u[2] = Q / A[2];
        u[2] = u[2] * u[2];
        if (u[2] < c[1]) h[0] = h[2]; else h[1] = h[2];
    }
    while (h[0]-h[1] > critical_depth_tolerance);
    return 0.5 * (h[0] + h[1]);
}
```

4.51.2.2 void node_critical_velocity (Node * node)

•

Function to calculate the critical velocity in a mesh node

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 86 of file [node.c](#).

```
{
    if (node->B > 0.) node->c = sqrt(G * node->A / node->B); else node->c = 0.
    ;
}
```

4.51.2.3 void node_depth (Node * node)

Function to calculate the depth in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 47 of file [node.c](#).

```
{
    if (node->Z == 0.)
        node->h = node->A / node->B0;
    else
        node->h = (sqrt(node->B0 * node->B0 + 4. * node->A * node->Z)
                  - node->B0) / (2 * node->Z);
    if (isnan(node->h)) printf("A=%lg B0=%lg Z=%lg\n", node->A, node->B0, node->Z);
}
```

4.51.2.4 void node_diffusion_Rutherford (Node * *node*)

Function to calculate the diffusion coefficient with the Rutherford model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 191 of file [node.c](#).

```
{
    node->Kx = node->diffusion_coefficient[0]
              * sqrt(G * node->P * node->A * fabs(node->Sf));
}
```

4.51.2.5 void node_friction_Manning (Node * *node*)

Function to calculate the friction slope with the Manning model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 160 of file [node.c](#).

```
{
    node->Sf = node->friction_coefficient[0] * node->friction_coefficient[0]
              * node->u * fabs(node->u) * pow(node->P / node->A, 4./3.);
}
```

4.51.2.6 void node_infiltration_KostiakovLewis (Node * *node*)

Function to calculate the infiltration with the Kostiakov-Lewis model.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 173 of file [node.c](#).

```
{
    node->i = node->infiltration_coefficient[2];
    if (node->infiltration_coefficient[0] == 0.) return;
    node->i += node->infiltration_coefficient[0] *
              node->infiltration_coefficient[1]
              * pow(node->Ai / (node->infiltration_coefficient[0]
```

```

    * node->infiltration_coefficient[3]),
    1. - 1. / node->infiltration_coefficient[1]);
}

```

4.51.2.7 void node_inlet (Node * *node*, Hydrogram * *water*, Hydrogram * *solute*, double *t*, double *t2*)

Function to calculate the inlet boundary condition.

Parameters

<i>node</i>	node struct.
<i>water</i>	water inlet hydrogram.
<i>solute</i>	solute inlet hydrogram.
<i>t</i>	actual time.
<i>t2</i>	next time

Definition at line 212 of file [node.c](#).

```

{
    node->A += hydrogram_integrate(water, t, t2) / node->dx;
    node->As += hydrogram_integrate(solute, t, t2) / node->
        dx;
    node_subcritical_discharge(node);
}

```

4.51.2.8 void node_outlet_closed (Node * *node*)

Function to calculate a closed outlet boundary condition.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 225 of file [node.c](#).

```

{
    node->Q = 0.;
}

```

4.51.2.9 void node_outlet_open (Node * *node*)

Function to calculate an open outlet boundary condition.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 236 of file [node.c](#).

```

{
    node_depth(node);
    node_width(node);
    node_critical_velocity(node);
    node->Q = fmax(node->Q, 1.01 * node->A * node->c);
}

```

4.51.2.10 void node_perimeter (Node * *node*)

Function to calculate the wetted perimeter in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 74 of file [node.c](#).

```
{
    node->P = node->B0 + 2 * sqrt(1 + node->Z * node->Z) * node->h;
}
```

4.51.2.11 void node_subcritical_discharge (Node * *node*)

Function to force a subcritical discharge in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 97 of file [node.c](#).

```
{
    node_depth(node);
    node_width(node);
    node_critical_velocity(node);
    node->Q = fmin(node->Q, 0.99 * node->A * node->c);
}
```

4.51.2.12 void node_width (Node * *node*)

Function to calculate the width in a mesh node.

Parameters

<i>node</i>	node struct.
-------------	--------------

Definition at line 63 of file [node.c](#).

```
{
    node->B = node->B0 + 2 * node->Z * node->h;
}
```

4.52 node.h

```
00001 /*
00002 SWOCS: a software to check the numerical performance of different models in
00003     channel or furrow flows
00004
00005 Copyright 2011, Javier Burguete Tolosa.
00006
00007 Redistribution and use in source and binary forms, with or without
00008     modification,
00009 are permitted provided that the following conditions are met:
00010     1. Redistributions of source code must retain the above copyright notice,
00011       this list of conditions and the following disclaimer.
00012     2. Redistributions in binary form must reproduce the above copyright
00013       notice,
```

```

00014         this list of conditions and the following disclaimer in the
00015         documentation and/or other materials provided with the distribution.
00016
00017 THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
    OR
00018 IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00019 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
    EVENT
00020 SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
    INDIRECT,
00021 INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
00022 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00023 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00024 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
00025 OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
00026 ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00027 */
00028
00036 // in order to prevent multiple definitions
00037 #ifndef NODE__H
00038 #define NODE__H 1
00039
00044 struct _Node
00045 {
00134     double friction_coefficient[3],
    infiltration_coefficient[4],
00135     diffusion_coefficient[1], x, dx, ix, A, Ai
    , Q, s, si, As, Asi, h, Sf,
00136     zb, zs, P, B, u, c, l1, l2, i, Pi, Z, B0, F, T, Kx
    , KxA, Kxi, KxiA,
00137     dQ, dF, dT, dQl, dFl, dTl, dQr, dFr, dTr, nu;
00138 };
00139
00143 typedef struct _Node Node;
00144
00145 // global variables
00146
00147 extern double critical_depth_tolerance;
00148
00149 // member functions
00150
00151 void node_depth(Node *node);
00152 void node_width(Node *node);
00153 void node_perimeter(Node *node);
00154 void node_critical_velocity(Node *node);
00155 void node_subcritical_discharge(Node *node);
00156 double node_critical_depth(Node *node, double Q);
00157 void node_friction_Manning(Node *node);
00158 void node_infiltration_KostiakovLewis(Node *
    node);
00159 void node_diffusion_Rutherford(Node *node);
00160 void node_inlet
    (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2);
00162 void node_outlet_closed(Node *node);
00163 void node_outlet_open(Node *node);
00164
00165 #endif

```

Index

- Channel, [5](#)
- channel.c, [13](#)
 - channel_diffusion_read_Rutherford, [14](#)
 - channel_friction_read_Manning, [14](#)
 - channel_infiltration_read_KostiakovLewis, [14](#)
 - channel_read, [15](#)
 - hydrogram_discharge, [17](#)
 - hydrogram_integrate, [17](#)
 - hydrogram_read, [18](#)
 - interpolate, [18](#)
- channel.h, [22](#)
 - channel_diffusion_read_Rutherford, [23](#)
 - channel_friction_read_Manning, [24](#)
 - channel_infiltration_read_KostiakovLewis, [24](#)
 - channel_read, [25](#)
 - hydrogram_discharge, [26](#)
 - hydrogram_integrate, [27](#)
 - hydrogram_read, [27](#)
 - interpolate, [28](#)
- channel_diffusion_read_Rutherford
 - channel.c, [14](#)
 - channel.h, [23](#)
- channel_friction_read_Manning
 - channel.c, [14](#)
 - channel.h, [24](#)
- channel_infiltration_read_KostiakovLewis
 - channel.c, [14](#)
 - channel.h, [24](#)
- channel_read
 - channel.c, [15](#)
 - channel.h, [25](#)
- config.h, [30](#)
- Hydrogram, [6](#)
- hydrogram_discharge
 - channel.c, [17](#)
 - channel.h, [26](#)
- hydrogram_integrate
 - channel.c, [17](#)
 - channel.h, [27](#)
- hydrogram_read
 - channel.c, [18](#)
 - channel.h, [27](#)
- interpolate
 - channel.c, [18](#)
 - channel.h, [28](#)
- main.c, [31](#)
- Mesh, [6](#)
- mesh.c, [35](#)
 - mesh_initial_conditions_dry, [36](#)
 - mesh_initial_conditions_profile, [36](#)
 - mesh_open, [37](#)
 - mesh_read, [38](#)
 - mesh_solute_mass, [39](#)
 - mesh_water_mass, [39](#)
 - mesh_write_flows, [40](#)
 - mesh_write_variables, [40](#)
- mesh.h, [44](#)
 - mesh_open, [44](#)
 - mesh_read, [45](#)
 - mesh_solute_mass, [46](#)
 - mesh_water_mass, [46](#)
 - mesh_write_flows, [47](#)
 - mesh_write_variables, [47](#)
- mesh_initial_conditions_dry
 - mesh.c, [36](#)
- mesh_initial_conditions_profile
 - mesh.c, [36](#)
- mesh_open
 - mesh.c, [37](#)
 - mesh.h, [44](#)
- mesh_read
 - mesh.c, [38](#)
 - mesh.h, [45](#)
- mesh_solute_mass
 - mesh.c, [39](#)
 - mesh.h, [46](#)
- mesh_water_mass
 - mesh.c, [39](#)
 - mesh.h, [46](#)
- mesh_write_flows
 - mesh.c, [40](#)
 - mesh.h, [47](#)
- mesh_write_variables
 - mesh.c, [40](#)
 - mesh.h, [47](#)
- Model, [7](#)
- model.c, [48](#)
 - model_diffusion_explicit, [50](#)
 - model_diffusion_implicit, [50](#)
 - model_infiltration, [50](#)
 - model_node_diffusion_1dt_max, [51](#)
 - model_parameters, [51](#)
 - model_print, [52](#)
 - model_probes_read, [52](#)
 - model_read, [53](#)
 - model_step, [54](#)

- model_write_advance, 54
 - model_write_probes, 55
- model.h, 59
 - model_diffusion_explicit, 60
 - model_diffusion_implicit, 60
 - model_infiltration, 61
 - model_node_diffusion_1dt_max, 61
 - model_parameters, 62
 - model_print, 62
 - model_probes_read, 62
 - model_read, 63
 - model_step, 64
 - model_write_advance, 65
 - model_write_probes, 65
- model_complete.c, 67
 - model_inlet_dtmax_complete, 67
 - model_node_parameters_complete, 68
 - node_1dt_max_complete, 68
 - node_flows_complete, 69
- model_complete.h, 70
 - model_inlet_dtmax_complete, 71
 - model_node_parameters_complete, 71
 - node_1dt_max_complete, 72
 - node_flows_complete, 72
- model_complete_LaxFriedrichs.c, 73
 - model_surface_flow_complete_LaxFriedrichs, 74
- model_complete_LaxFriedrichs.h, 76
 - model_surface_flow_complete_LaxFriedrichs, 76
- model_diffusion_explicit
 - model.c, 50
 - model.h, 60
- model_diffusion_implicit
 - model.c, 50
 - model.h, 60
- model_diffusive.c, 78
 - model_inlet_dtmax_diffusive, 79
 - model_node_parameters_centre_diffusive, 79
 - model_node_parameters_left_diffusive, 79
 - model_node_parameters_right_diffusive, 80
 - node_1dt_max_diffusive, 81
 - node_discharge_centre_diffusive_Manning, 81
 - node_discharge_left_diffusive_Manning, 81
 - node_discharge_right_diffusive_Manning, 81
 - node_flows_diffusive, 82
- model_diffusive.h, 84
 - model_inlet_dtmax_diffusive, 85
 - model_node_parameters_centre_diffusive, 85
 - model_node_parameters_left_diffusive, 86
 - model_node_parameters_right_diffusive, 87
 - node_1dt_max_diffusive, 87
 - node_discharge_centre_diffusive_Manning, 87
 - node_discharge_left_diffusive_Manning, 88
 - node_discharge_right_diffusive_Manning, 88
 - node_flows_diffusive, 88
- model_diffusive_upwind.c, 89
 - model_surface_flow_diffusive_upwind, 90
- model_diffusive_upwind.h, 91
 - model_surface_flow_diffusive_upwind, 92
- model_infiltration
 - model.c, 50
 - model.h, 61
- model_inlet_dtmax_complete
 - model_complete.c, 67
 - model_complete.h, 71
- model_inlet_dtmax_diffusive
 - model_diffusive.c, 79
 - model_diffusive.h, 85
- model_inlet_dtmax_kinematic
 - model_kinematic.c, 94
 - model_kinematic.h, 100
- model_inlet_dtmax_zero_inertia
 - model_zero_inertia.c, 108
 - model_zero_inertia.h, 112
- model_kinematic.c, 93
 - model_inlet_dtmax_kinematic, 94
 - model_node_parameters_centre_kinematic, 94
 - model_node_parameters_left_kinematic, 95
 - model_node_parameters_right_kinematic, 95
 - node_1dt_max_kinematic, 96
 - node_discharge_centre_kinematic_Manning, 96
 - node_discharge_left_kinematic_Manning, 96
 - node_discharge_right_kinematic_Manning, 97
 - node_flows_kinematic, 97
- model_kinematic.h, 99
 - model_inlet_dtmax_kinematic, 100
 - model_node_parameters_centre_kinematic, 100
 - model_node_parameters_left_kinematic, 101
 - model_node_parameters_right_kinematic, 102
 - node_1dt_max_kinematic, 102
 - node_discharge_centre_kinematic_Manning, 102
 - node_discharge_left_kinematic_Manning, 103
 - node_discharge_right_kinematic_Manning, 103
 - node_flows_kinematic, 103
- model_kinematic_upwind.c, 104
 - model_surface_flow_kinematic_upwind, 105
- model_kinematic_upwind.h, 106
 - model_surface_flow_kinematic_upwind, 107
- model_node_diffusion_1dt_max
 - model.c, 51
 - model.h, 61
- model_node_parameters_centre_diffusive
 - model_diffusive.c, 79
 - model_diffusive.h, 85
- model_node_parameters_centre_kinematic
 - model_kinematic.c, 94
 - model_kinematic.h, 100
- model_node_parameters_complete
 - model_complete.c, 68
 - model_complete.h, 71
- model_node_parameters_left_diffusive
 - model_diffusive.c, 79
 - model_diffusive.h, 86
- model_node_parameters_left_kinematic
 - model_kinematic.c, 95
 - model_kinematic.h, 101
- model_node_parameters_right_diffusive

- model_diffusive.c, [80](#)
 - model_diffusive.h, [87](#)
 - model_node_parameters_right_kinematic
 - model_kinematic.c, [95](#)
 - model_kinematic.h, [102](#)
 - model_node_parameters_zero_inertia
 - model_zero_inertia.c, [109](#)
 - model_zero_inertia.h, [112](#)
 - model_parameters
 - model.c, [51](#)
 - model.h, [62](#)
 - model_print
 - model.c, [52](#)
 - model.h, [62](#)
 - model_probes_read
 - model.c, [52](#)
 - model.h, [62](#)
 - model_read
 - model.c, [53](#)
 - model.h, [63](#)
 - model_step
 - model.c, [54](#)
 - model.h, [64](#)
 - model_surface_flow_complete_LaxFriedrichs
 - model_complete_LaxFriedrichs.c, [74](#)
 - model_complete_LaxFriedrichs.h, [76](#)
 - model_surface_flow_diffusive_upwind
 - model_diffusive_upwind.c, [90](#)
 - model_diffusive_upwind.h, [92](#)
 - model_surface_flow_kinematic_upwind
 - model_kinematic_upwind.c, [105](#)
 - model_kinematic_upwind.h, [107](#)
 - model_surface_flow_zero_inertia_LaxFriedrichs
 - model_zero_inertia_LaxFriedrichs.c, [115](#)
 - model_zero_inertia_LaxFriedrichs.h, [117](#)
 - model_write_advance
 - model.c, [54](#)
 - model.h, [65](#)
 - model_write_probes
 - model.c, [55](#)
 - model.h, [65](#)
 - model_zero_inertia.c, [108](#)
 - model_inlet_dtmx_zero_inertia, [108](#)
 - model_node_parameters_zero_inertia, [109](#)
 - node_1dt_max_zero_inertia, [109](#)
 - node_flows_zero_inertia, [110](#)
 - model_zero_inertia.h, [111](#)
 - model_inlet_dtmx_zero_inertia, [112](#)
 - model_node_parameters_zero_inertia, [112](#)
 - node_1dt_max_zero_inertia, [113](#)
 - node_flows_zero_inertia, [113](#)
 - model_zero_inertia_LaxFriedrichs.c, [114](#)
 - model_zero_inertia_LaxFriedrichs.h, [117](#)
- Node, [8](#)
- node.c, [119](#)
 - node_critical_depth, [120](#)
 - node_critical_velocity, [120](#)
 - node_depth, [121](#)
 - node_diffusion_Rutherford, [121](#)
 - node_friction_Manning, [121](#)
 - node_infiltration_KostiakovLewis, [121](#)
 - node_inlet, [122](#)
 - node_outlet_closed, [122](#)
 - node_outlet_open, [122](#)
 - node_perimeter, [123](#)
 - node_subcritical_discharge, [123](#)
 - node_width, [123](#)
 - node.h, [125](#)
 - node_critical_depth, [126](#)
 - node_critical_velocity, [127](#)
 - node_depth, [127](#)
 - node_diffusion_Rutherford, [128](#)
 - node_friction_Manning, [128](#)
 - node_infiltration_KostiakovLewis, [128](#)
 - node_inlet, [129](#)
 - node_outlet_closed, [129](#)
 - node_outlet_open, [129](#)
 - node_perimeter, [129](#)
 - node_subcritical_discharge, [130](#)
 - node_width, [130](#)
 - node_1dt_max_complete
 - model_complete.c, [68](#)
 - model_complete.h, [72](#)
 - node_1dt_max_diffusive
 - model_diffusive.c, [81](#)
 - model_diffusive.h, [87](#)
 - node_1dt_max_kinematic
 - model_kinematic.c, [96](#)
 - model_kinematic.h, [102](#)
 - node_1dt_max_zero_inertia
 - model_zero_inertia.c, [109](#)
 - model_zero_inertia.h, [113](#)
 - node_critical_depth
 - node.c, [120](#)
 - node.h, [126](#)
 - node_critical_velocity
 - node.c, [120](#)
 - node.h, [127](#)
 - node_depth
 - node.c, [121](#)
 - node.h, [127](#)
 - node_diffusion_Rutherford
 - node.c, [121](#)
 - node.h, [128](#)
 - node_discharge_centre_diffusive_Manning
 - model_diffusive.c, [81](#)
 - model_diffusive.h, [87](#)
 - node_discharge_centre_kinematic_Manning
 - model_kinematic.c, [96](#)
 - model_kinematic.h, [102](#)
 - node_discharge_left_diffusive_Manning
 - model_diffusive.c, [81](#)
 - model_diffusive.h, [88](#)
 - node_discharge_left_kinematic_Manning
 - model_kinematic.c, [96](#)
 - model_kinematic.h, [103](#)

- node_discharge_right_diffusive_Manning
 - model_diffusive.c, [81](#)
 - model_diffusive.h, [88](#)
- node_discharge_right_kinematic_Manning
 - model_kinematic.c, [97](#)
 - model_kinematic.h, [103](#)
- node_flows_complete
 - model_complete.c, [69](#)
 - model_complete.h, [72](#)
- node_flows_diffusive
 - model_diffusive.c, [82](#)
 - model_diffusive.h, [88](#)
- node_flows_kinematic
 - model_kinematic.c, [97](#)
 - model_kinematic.h, [103](#)
- node_flows_zero_inertia
 - model_zero_inertia.c, [110](#)
 - model_zero_inertia.h, [113](#)
- node_friction_Manning
 - node.c, [121](#)
 - node.h, [128](#)
- node_infiltration_KostiakovLewis
 - node.c, [121](#)
 - node.h, [128](#)
- node_inlet
 - node.c, [122](#)
 - node.h, [129](#)
- node_outlet_closed
 - node.c, [122](#)
 - node.h, [129](#)
- node_outlet_open
 - node.c, [122](#)
 - node.h, [129](#)
- node_perimeter
 - node.c, [123](#)
 - node.h, [129](#)
- node_subcritical_discharge
 - node.c, [123](#)
 - node.h, [130](#)
- node_width
 - node.c, [123](#)
 - node.h, [130](#)

Probes, [10](#)