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_dtmax)(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

channel	channel struct.
file	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

channel	channel struct.
file	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

channel	channel struct.
file	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

channel	channel struct.
file	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%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;
    }
#if 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

hydrogram	hydrogram struct.
t	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

hydrogram	hydrogram struct.
t1	initial time.
t2	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

hydrogram	hydrogram struct.
file	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
       modification,
00008 are permitted provided that the following conditions are met:
00009 
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 
00035 #include <stdio.h>
00036 #include <stdlib.h>
00037 #include "config.h"
00038 #include "channel.h"
00039 
00055 double interpolate(double x, double x1, double x2, double y1, double
       y2)
00056 {
00057     return y1 + (x - x1) * (y2 - y1) / (x2 - x1);
00058 }
00059 
00069 int hydrogram_read(Hydrogram *hydrogram, FILE *file)
00070 {
00071     int i;
00072     char *msg;
00073     if (fscanf(file, "%d", &hydrogram->n) != 1 || hydrogram->n < 1)
00074     {
        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;
}

double hydrogram_discharge(Hydrogram *hydrogram, double t)
{
    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]);
}

double hydrogram_integrate(Hydrogram *hydrogram, double t1,
double t2)
{
    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]);
}

```

```c
int channel_friction_read_Manning(Channel *channel
, FILE *file)
{
    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;
}

int channel_infiltration_read_KostiakovLewis
(Channel *channel, FILE *file)
{
    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;
}

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

int channel_read(Channel *channel, FILE *file)
{
    char *msg;
    if (fscanf(file, "%lf%lf%lf%lf%lf%d%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;
    }
#if 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 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

channel	channel struct.
file	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

channel	channel struct.
file	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

channel	channel struct.
file	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;
}
```

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

function to read a channel.

Parameters

channel	channel struct.
file	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%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;
    }
#if 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

hydrogram	hydrogram struct.
t	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

hydrogram	hydrogram struct.
t1	initial time.
t2	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

hydrogram	hydrogram struct.
file	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

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);
}
```

4.4 channel.h

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef CHANNEL__H
#define CHANNEL__H 1

struct _Hydrogram
{
    double *t, *Q;
    int n;
};

typedef struct _Hydrogram Hydrogram;

struct _Channel
{
    Hydrogram water_inlet[1], solute_inlet[1];
    double friction_coefficient[3], infiltration_coefficient[4],
        diffusion_coefficient[1], slope, length, bottom_width, wall_slope,
        height;
    int type_outlet, friction_model, infiltration_model, diffusion_model;
};

typedef struct _Channel Channel;

// member functions

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

int hydrogram_read(Hydrogram *hydrogram, FILE *file);
double hydrogram_discharge(Hydrogram *hydrogram, double t);
double hydrogram_integrate(Hydrogram *hydrogram, double t1, double t2);

int channel_friction_read_Manning(Channel *channel, FILE *file);
int channel_infiltration_read_KostiakovLewis(Channel *channel, FILE *file);
int channel_diffusion_read_Rutherford(Channel *channel, FILE *file);
int channel_read(Channel *channel,FILE *file);

```

```
#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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```

// in order to prevent multiple definitions
#ifndef CONFIG__H
#define CONFIG__H 1

#define G 9.81

// debug defines

#define DEBUG_MODEL_READ    0

#define DEBUG_MESH_OPEN     0

#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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"
//#include "model_diffusive_LaxFriedrichs.h"
//#include "model_kinematic_LaxFriedrichs.h"

double critical_depth_tolerance = 0.001;

int main(int argn, char **argc)
{
    int i;
    FILE *file, *file_advance, *file_probes;
    Model model[1];
    if (argn < 3 || argn == 6 || argn > 7)
    {
        printf("the sintaxis is:\n./SWOCS input_file "
            "output_variables_file "
            "[output_flows_file] [output_advance_file]"
            "[input_probes_file output_probes_file]\n");
        return 1;
    }

    if (!model_read(model, argc[1])) return 2;

    switch (model->type_model)
    {
    case 1:
        model->model_node_parameters_centre
            = model->model_node_parameters_right
            = model->model_node_parameters_left
            = model_node_parameters_complete;
        model->node_1dt_max = node_1dt_max_complete;
        model->node_flows = node_flows_complete;
        model->model_inlet_dtmax = model_inlet_dtmax_complete
;
        goto complete;
    case 2:
        model->model_node_parameters_centre
            = model->model_node_parameters_right
            = model->model_node_parameters_left
            = model_node_parameters_zero_inertia
;
        model->node_1dt_max = node_1dt_max_zero_inertia
;
        model->node_flows = node_flows_zero_inertia;
        model->model_inlet_dtmax = model_inlet_dtmax_zero_inertia
;
        goto zero_inertia;
    case 3:
        switch (model->channel->friction_model)
        {
        case 1:
            model->node_discharge_centre
                = node_discharge_centre_diffusive_Manning
;
            model->node_discharge_right
                = node_discharge_right_diffusive_Manning
;
            model->node_discharge_left
                = node_discharge_left_diffusive_Manning
;
            model->model_node_parameters_centre
                = model_node_parameters_centre_diffusive
;
            model->model_node_parameters_right
                = model_node_parameters_right_diffusive
;
            model->model_node_parameters_left
                = model_node_parameters_left_diffusive
;
        }
        model->node_1dt_max = node_1dt_max_diffusive;
        model->node_flows = node_flows_diffusive;
        model->model_inlet_dtmax = model_inlet_dtmax_diffusive
;
        goto diffusive;
    case 4:
        switch (model->channel->friction_model)
        {
        case 1:
            model->node_discharge_centre
                = node_discharge_centre_kinematic_Manning
;
            model->node_discharge_right
                = node_discharge_right_kinematic_Manning
;
            model->node_discharge_left
                = node_discharge_left_kinematic_Manning
;
            model->model_node_parameters_centre
                = model_node_parameters_centre_kinematic
;
            model->model_node_parameters_right
                = model_node_parameters_right_kinematic
;
            model->model_node_parameters_left
                = model_node_parameters_left_kinematic
;
        }
        model->node_1dt_max = node_1dt_max_kinematic;
        model->node_flows = node_flows_kinematic;
        model->model_inlet_dtmax = model_inlet_dtmax_kinematic
;
        goto kinematic;
    default:
        printf("model: bad type\n");
        return 2;
    }

complete:
    switch (model->type_surface_flow)
    {
    case 1:
        model->model_surface_flow = model_surface_flow_complete_upwind;
        goto calculate;
    case 2:
        model->model_surface_flow = model_surface_flow_complete_LaxFriedrichs
;
        goto calculate;
    default:
        printf("model: bad surface flow type\n");
        return 2;
    }

zero_inertia:
    switch (model->type_surface_flow)
    {
    case 1:
        model->model_surface_flow = model_surface_flow_zero_inertia_upwind;
        goto calculate;
    case 2:
        model->model_surface_flow =
            model_surface_flow_zero_inertia_LaxFriedrichs
;
        goto calculate;
    default:
        printf("model: bad surface flow type\n");
        return 2;
    }

diffusive:
    switch (model->type_surface_flow)
    {
    case 1:
        model->model_surface_flow = model_surface_flow_diffusive_upwind
;
        break;
//  case 2:
//      model->model_surface_flow = model_surface_flow_diffusive_LaxFriedrichs;
//      break;
    default:
        printf("model: bad surface flow type\n");
        return 2;
    }

kinematic:
    switch (model->type_surface_flow)
    {
    case 1:
        model->model_surface_flow = model_surface_flow_kinematic_upwind
;
        break;
//  case 2:
//      model->model_surface_flow = model_surface_flow_kinematic_LaxFriedrichs;
//      break;
    default:
        printf("model: bad surface flow type\n");
        return 2;
    }

calculate:
    switch (model->type_diffusion)
    {
    case 1:
        model->model_diffusion = model_diffusion_explicit
;
        break;
    case 2:
        model->model_diffusion = model_diffusion_implicit
;
        break;
    default:
        printf("model: bad diffusion type\n");
        return 2;
    }

    if (argn > 4)
    {
        // opening the advance file
        file_advance = fopen(argc[4], "w");

        if (argn > 6)
        {
            // opening the probes files
            if (!model_probes_read(model, argc[5])) return 2;
            file_probes = fopen(argc[6], "w");
            if (!file_probes)
            {
                printf("model: unable to open the probes output file\n");
                return 2;
            }
        }
    }

    // init model parameters
    model_parameters(model);

    // main calculation bucle
    for (model->t = 0, i = 0; model->t < model->tfinal; ++i)
    {
        if (argn > 4)
        {
            // writing the advance
            model_write_advance(model, file_advance);

            // writing the probes
            if (argn > 6) model_write_probes(model,
file_probes);
        }

        // model step
        model_step(model);
//      model_print(model, i);
    }
    model_print(model, i);

    // writing result variables
    file = fopen(argc[2], "w");
    mesh_write_variables(model->mesh, file);
    fclose(file);

    // writing result flows
    if (argn > 3)
    {
        file = fopen(argc[3], "w");
        mesh_write_flows(model->mesh, file);
        fclose(file);

        if (argn > 4)
        {
            // closing the advance
            model_write_advance(model, file_advance);
            fclose(file_advance);

            if (argn > 6)
            {
                // closing the probes
                model_write_probes(model, file_probes);
                fclose(file_probes);
            }
        }
    }

    return 0;
}
```

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

mesh	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

mesh	mesh struct.
file	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

mesh	mesh struct.
channel	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

mesh	mesh struct.
channel	channel struct.
file	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;
    }
#if 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

mesh	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

mesh	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

mesh	mesh struct.
file	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

mesh	mesh struct.
file	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

```c
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

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

int mesh_open(Mesh *mesh, Channel *channel)
{
    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;
```

```
}

void mesh_initial_conditions_dry(Mesh *mesh)
{
    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.;
}

int mesh_initial_conditions_profile(Mesh *mesh,
FILE *file)
{
    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;
}

int mesh_read(Mesh *mesh, Channel *channel, FILE *file)
{
    char *msg;
    if (fscanf(file, "%d%d", &mesh->n, &mesh->type) != 2)
```

```c
    {
        msg = "mesh: bad defined\n";
        goto bad;
    }
    if (mesh->n < 3)
    {
        msg = "mesh: bad nodes number\n";
        goto bad;
    }
#if 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;
}

void mesh_write_variables(Mesh *mesh, FILE *file)
{
    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);
    }
}

void mesh_write_flows(Mesh *mesh, FILE *file)
{
    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));
    }
}

double mesh_water_mass(Mesh *mesh)
{
    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;
}

double mesh_solute_mass(Mesh *mesh)
{
    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 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

mesh	mesh struct.
channel	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

mesh	mesh struct.
channel	channel struct.
file	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;
    }
#if 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

mesh	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

mesh	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

mesh	mesh struct.
file	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

mesh	mesh struct.
file	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);
    }
}
```

4.12 mesh.h

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MESH__H
#define MESH__H 1

struct _Mesh
{
    Node *node;
    int n, type;
};

typedef struct _Mesh Mesh;

// member functions

int mesh_open(Mesh *mesh, Channel *channel);
int mesh_read(Mesh *mesh, Channel *channel, FILE *file);
void mesh_write_variables(Mesh *mesh, FILE *file);
void mesh_write_flows(Mesh *mesh, FILE *file);
double mesh_water_mass(Mesh *mesh);
double mesh_solute_mass(Mesh *mesh);

#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

model	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

model	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

model	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;
    }
}
```

4.13.2.4 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

node	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

model	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

model	model struct.
nsteps	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

model	model struct.
name	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

model	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%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;
    }
#if 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

model	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_1dt_max(node + i));
    if (model->type_diffusion == 1)
    {
        for (i = 0, dtmax = 0; i < mesh->n; ++i)
            dtmax = fmax(dtmax, model_node_diffusion_1dt_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

model	model struct.
file	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

model	model struct.
file	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

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

void model_parameters(Model *model)
{
    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);
}

void model_infiltration(Model *model)
{
    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;
    }
}

void model_diffusion_explicit(Model *model)
{
    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;
    }
}

void model_diffusion_implicit(Model *model)
{
    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;
    }
}

double model_node_diffusion_1dt_max(Node *node)
{
    return (2 * node->Kx + fabs(node->u) * node->dx) / (node->dx * node->dx);
}

void model_step(Model *model)
{
    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_1dt_max(node + i));
    if (model->type_diffusion == 1)
    {
        for (i = 0, dtmax = 0; i < mesh->n; ++i)
            dtmax = fmax(dtmax, model_node_diffusion_1dt_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;
}

int model_read(Model *model, char *file_name)
{
    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%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;
    }
#if 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;
}

void model_print(Model *model, int nsteps)
{
    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));
}

void model_write_advance(Model *model, FILE *file)
{
    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);
}

int model_probes_read(Model *model, char *name)
{
    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;
}

void model_write_probes(Model *model, FILE *file)
{
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");
}
```

4.15 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

model	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

model	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

model	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

node	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

model	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

model	model struct.
nsteps	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

model	model struct.
name	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

model	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%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;
    }
#if 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

model	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_1dt_max(node + i));
    if (model->type_diffusion == 1)
    {
        for (i = 0, dtmax = 0; i < mesh->n; ++i)
            dtmax = fmax(dtmax, model_node_diffusion_1dt_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

model	model struct.
file	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

model	model struct.
file	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL__H
#define MODEL__H 1

struct _Probes
{
    double *x;
    int *node, n;
};

typedef struct _Probes Probes;

struct _Model
{
    Mesh mesh[1];
    Channel channel[1];
    Probes probes[1];
    double t, t2, dt, tfinal, cfl, interval, minimum_depth;
    void (*model_node_parameters_centre)(struct _Model *model, Node *node);
    void (*model_node_parameters_right)(struct _Model *model, Node *node);
    void (*model_node_parameters_left)(struct _Model *model, Node *node);
    double (*node_1dt_max)(Node *node);
    double (*model_inlet_dtmax)(struct _Model *model);
    void (*node_flows)(Node *node1);
    void (*node_discharge_centre)(Node *node);
    void (*node_discharge_right)(Node *node);
    void (*node_discharge_left)(Node *node);
    void (*node_friction)(Node *node);
    void (*node_infiltration)(Node *node);
    void (*node_diffusion)(Node *node);
    void (*node_inlet)
        (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2);
    void (*node_outlet)(Node *node);
    void (*model_surface_flow)(struct _Model *model);
    void (*model_diffusion)(struct _Model *model);
    int type_surface_flow, type_diffusion, type_model;
};

typedef struct _Model Model;

// member functions

void model_parameters(Model *model);
void model_infiltration(Model *model);
void model_diffusion_explicit(Model *model);
void model_diffusion_implicit(Model *model);
double model_node_diffusion_1dt_max(Node *node);
void model_step(Model *model);
int model_read(Model *model, char *file_name);
void model_print(Model *model, int nsteps);
void model_write_advance(Model *model, FILE *file);
int model_probes_read(Model *model, char *name);
void model_write_probes(Model *model, FILE *file);

#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_dtmax_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_dtmax_complete (Model ∗ *model*)

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

Parameters

model	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*)

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

Parameters

model	model struct.
node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void model_node_parameters_complete(Model *model,
 Node *node)
{
    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;
}

double node_1dt_max_complete(Node *node)
{
    return (node->c + fabs(node->u)) / node->dx;
}

void node_flows_complete(Node *node1)
{
    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;
}

double model_inlet_dtmax_complete(Model *model)
{
    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 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

model	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*)

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

Parameters

model	model struct.
node	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

node	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

node1	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;
}
```

4.20 model_complete.h

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_COMPLETE__H
#define MODEL_COMPLETE__H 1

// member functions

void model_node_parameters_complete(Model *model,
 Node *node);
double node_1dt_max_complete(Node *node);
void node_flows_complete(Node *node1);
double model_inlet_dtmax_complete(Model *model);

#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

model	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].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;
}
```

4.22 model_complete_LaxFriedrichs.c

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void model_surface_flow_complete_LaxFriedrichs
 (Model *model)
{
    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;
}
```

4.23 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

model	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].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;
}
```

4.24 model_complete_LaxFriedrichs.h

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
```

```
#ifndef MODEL_COMPLETE_LAXFRIEDRICHS__H
#define MODEL_COMPLETE_LAXFRIEDRICHS__H 1

// member functions

void model_surface_flow_complete_LaxFriedrichs
 (Model *model);

#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_dtmax_diffusive (Model ∗ *model*)

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

Parameters

model	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

model	model struct.
node	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

model	model struct.
node	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

model	model struct.
node	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

node	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

node	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

node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void node_discharge_centre_diffusive_Manning
 (Node *node)
{
    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];
}

void node_discharge_right_diffusive_Manning
 (Node *node)
{
    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];
}

void node_discharge_left_diffusive_Manning
 (Node *node)
{
    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];
}

void model_node_parameters_centre_diffusive
 (Model *model, Node *node)
{
    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;
}

void model_node_parameters_right_diffusive
 (Model *model, Node *node)
{
    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;
}

void model_node_parameters_left_diffusive(
Model *model, Node *node)
{
    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;
}

double node_1dt_max_diffusive(Node *node)
{
    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;
}

void node_flows_diffusive(Node *node1)
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dT = node2->T - node1->T;
}

double model_inlet_dtmax_diffusive(Model *model)
{
    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 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_dtmax_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_dtmax_diffusive (Model ∗ *model*)

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

Parameters

model	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

model	model struct.
node	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

model	model struct.
node	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

model	model struct.
node	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

node	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

node	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

node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_DIFFUSIVE__H
#define MODEL_DIFFUSIVE__H 1

// member functions

void node_discharge_centre_diffusive_Manning(Node *node);
void node_discharge_right_diffusive_Manning(Node *node);
void node_discharge_left_diffusive_Manning(Node *node);
void model_node_parameters_centre_diffusive(Model *model, Node *node);
void model_node_parameters_right_diffusive(Model *model, Node *node);
void model_node_parameters_left_diffusive(Model *model, Node *node);
double node_1dt_max_diffusive(Node *node);
void node_flows_diffusive(Node *node1);
double model_inlet_dtmax_diffusive(Model *model);

#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

model	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void model_surface_flow_diffusive_upwind(Model *model)
{
    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;
}
```

4.31 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

model	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
```

```
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_DIFFUSIVE_UPWIND__H
#define MODEL_DIFFUSIVE_UPWIND__H 1

// member functions

void model_surface_flow_diffusive_upwind(
Model *model);

#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_dtmax_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_dtmax_kinematic (Model ∗ *model*)

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

Parameters

model	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

model	model struct.
node	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

model	model struct.
node	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

model	model struct.
node	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

node	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

node	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

node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
```

```c
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void node_discharge_centre_kinematic_Manning(Node *node)
{
    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];
}

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

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

void model_node_parameters_centre_kinematic(Model *model, Node *node)
{
    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;
}

void model_node_parameters_right_kinematic(Model *model, Node *node)
{
    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;
}

void model_node_parameters_left_kinematic(
Model *model, Node *node)
{
    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;
}

double node_1dt_max_kinematic(Node *node)
{
    return (5./3. * node->u - 4./3. * node->Q * sqrt(1 + node->Z * node->Z)
        / (node->B * node->P)) / node->dx;
}

void node_flows_kinematic(Node *node1)
{
    Node *node2 = node1 + 1;
    node1->dQ = node2->Q - node1->Q;
    node1->dT = node2->T - node1->T;
}

double model_inlet_dtmax_kinematic(Model *model)
{
    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 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_dtmax_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_dtmax_kinematic (Model ∗ *model*)

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

Parameters

model	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

model	model struct.
node	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

model	model struct.
node	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

model	model struct.
node	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

node	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

node	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

node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_KINEMATIC__H
#define MODEL_KINEMATIC__H 1

// member functions

void node_discharge_centre_kinematic_Manning(Node *node);
void node_discharge_right_kinematic_Manning(Node *node);
void node_discharge_left_kinematic_Manning(Node *node);
void model_node_parameters_centre_kinematic(Model *model, Node *node);
void model_node_parameters_right_kinematic(Model *model, Node *node);
void model_node_parameters_left_kinematic(Model *model, Node *node);
double node_1dt_max_kinematic(Node *node);
void node_flows_kinematic(Node *node1);
double model_inlet_dtmax_kinematic(Model *model);

#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

model	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
```

```
notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void model_surface_flow_kinematic_upwind(Model *model)
{
    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;
}
```

4.39 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

model	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_KINEMATIC_UPWIND__H
#define MODEL_KINEMATIC_UPWIND__H 1

// member functions

void model_surface_flow_kinematic_upwind(
Model *model);

#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

model	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

model	model struct.
node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void model_node_parameters_zero_inertia(Model *model, Node *node)
{
    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);
}

double node_1dt_max_zero_inertia(Node *node)
{
    return node->l1 / node->dx;
}

void node_flows_zero_inertia(Node *node1)
{
    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;
}

double model_inlet_dtmax_zero_inertia(Model *
model)
{
    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 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_dtmax_zero_inertia (Model ∗ *model*)

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

Parameters

model	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

model	model struct.
node	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

node	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

node1	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

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_ZERO_INERTIA__H
#define MODEL_ZERO_INERTIA__H 1

// member functions

void model_node_parameters_zero_inertia(Model *model, Node *node);
double node_1dt_max_zero_inertia(Node *node);
void node_flows_zero_inertia(Node *node1);
double model_inlet_dtmax_zero_inertia(Model *model);

#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

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

model	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 /*
```

```
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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"

void model_surface_flow_zero_inertia_LaxFriedrichs
 (Model *model)
{
    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;
}
```

4.47 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

model	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;
}
```

4.48 model_zero_inertia_LaxFriedrichs.h

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS
 OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
 EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef MODEL_ZERO_INERTIA_LAXFRIEDRICHS__H
#define MODEL_ZERO_INERTIA_LAXFRIEDRICHS__H 1

// member functions

void model_surface_flow_zero_inertia_LaxFriedrichs
 (Model *model);

#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

node	node struct.
Q	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

node	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

node	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

node	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

node	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

node	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

node	node struct.
water	water inlet hydrogram.
solute	solute inlet hydrogram.
t	actual time.
t2	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

node	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

node	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

node	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

node	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

node	node struct.

Definition at line 63 of file node.c.

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

4.50 node.c

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.
```

```

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

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

void node_depth(Node *node)
{
    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);
}

void node_width(Node *node)
{
    node->B = node->B0 + 2 * node->Z * node->h;
}

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

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

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

double node_critical_depth(Node *node, double Q)
{
    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]);
}

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

void node_infiltration_KostiakovLewis(Node *
node)
{
    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]);
}

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

void node_inlet
    (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2)
{
    node->A += hydrogram_integrate(water, t, t2) / node->dx;
    node->As += hydrogram_integrate(solute, t, t2) / node->
dx;
    node_subcritical_discharge(node);
}

void node_outlet_closed(Node *node)
{
    node->Q = 0.;
}

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

```

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

node	node struct.
Q	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

node	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

node	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

node	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

node	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

node	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

node	node struct.
water	water inlet hydrogram.
solute	solute inlet hydrogram.
t	actual time.
t2	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

node	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

node	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

node	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

node	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

node	node struct.

Definition at line 63 of file node.c.

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

4.52 node.h

```
/*
SWOCS: a software to check the numerical performance of different models in
    channel or furrow flows

Copyright 2011, Javier Burguete Tolosa.

Redistribution and use in source and binary forms, with or without
 modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright
 notice,
      this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Javier Burguete Tolosa ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL Javier Burguete Tolosa OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// in order to prevent multiple definitions
#ifndef NODE__H
#define NODE__H 1

struct _Node
{
    double friction_coefficient[3], infiltration_coefficient[4],
        diffusion_coefficient[1], x, dx, ix, A, Ai, Q, s, si, As, Asi, h, Sf,
        zb, zs, P, B, u, c, l1, l2, i, Pi, Z, B0, F, T, Kx, KxA, Kxi, KxiA,
        dQ, dF, dT, dQl, dFl, dTl, dQr, dFr, dTr, nu;
};

typedef struct _Node Node;

// global variables

extern double critical_depth_tolerance;

// member functions

void node_depth(Node *node);
void node_width(Node *node);
void node_perimeter(Node *node);
void node_critical_velocity(Node *node);
void node_subcritical_discharge(Node *node);
double node_critical_depth(Node *node, double Q);
void node_friction_Manning(Node *node);
void node_infiltration_KostiakovLewis(Node *node);
void node_diffusion_Rutherford(Node *node);
void node_inlet
    (Node *node, Hydrogram *water, Hydrogram *solute, double t, double t2);
void node_outlet_closed(Node *node);
void node_outlet_open(Node *node);

#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_dtmax_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_dtmax_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