DSM2-STM Reference Manual

Generated by Doxygen 1.5.8

Sat Aug 22 23:32:21 2009

Contents

Chapter 1

STM Code Documentation and Recommended Work Plan

1.1 Introduction

This module of DSM2 is a transport code modeling advection, diffusion, sediment processes and reactions.

1.2 Recommended Work Plan

1. Get doxygen working. The documentation is automatically built by the MS Visual Studio solution as long as doxygen is on path. See below on Visual Studio.

2. Code a function-test pair and add it to test_transport_driver(). I suggest you either

   - adjust your diffusion code stylistically and write a matching unit test (see below on unit tests) or
   - complete the extrapolate() function in the advect module and write a test for that. See below for how to write good unit tests.

3. Move the advection.replace_boundary_flux() subroutine out of the advection model since it it is driver/application specific. This step will teach you about interfaces and callbacks. Discuss the design with Nicky and Kevin and Eli.

4. After that, start looking at which functions in the advection module are not really implemented. Work in function-test pairs (they often evolve together). As much as you can, put "asserts" in your testing function that are needed regardless of

whether you can pass them. The Tests, and in some ways the documentation, are our way of communicating expected behavior.

5. You will be at the same point you are in Matlab when you code the uniform flow advection test. However, we will not want you to code this or turn it in until the other testing (extrapolate(), conservative update) are well tested.

6. Do not significantly change the granularity or form of these functions without a design meeting. It is recommended that you discuss the API (names and arguments) of routines and the basic flow as a skeleton before you code along with the tests. If you pass the tests and have a functional form we can use, you have succeeded at the task by definition. In the meantime:

7. I recommend you work in this order, but you may rearrange the diffusion part as you wish:

    - code and test existing skeleton routines in the transport library
    - code test for uniform flow advection no source or diffusion
        - achieve comfort with convergence testing
    - code test for non-uniform flow advection (handle flow/area consistency issues)
    - add linear source and test convergence
    - code test for diffusion
        - explicit operator
        - implicit solver
            * convergence test
    - code test for advective-diffusion
    - meet on channel network challenges
    - modify for sediment
    - complete channel network stuff

1.3 Visual Studio

Hopefully we will just give you a complete project that has most of the settings right. To get this to compile you need to make sure you have:

1. Doxygen installed and on path. To get the doc project to generate, you need to have doxygen installed. Make sure it is on path (test by typing doxygen at the command line). You can add to the path on your computer or set it up in Visual Studio's Tools > Options > Projects and Solutions > VC++ Di drectories (for executables) Note the doc project has a custom build step (cd's to /doc and does documentation.bat) output for the custom build step is /doc/html/index.html You can view /doc/html/index.html in Visual Studio by right clicking it and choosing "View in Browser"

2. Set up the dependencies (hopefully we will do this for you)
    - test_transport on transport and fruit
    - sandbox_application on transport (you can alter as desired)e
3. Set the fortran "additional include directories" in sandbox_application and test_transport_driver to pick up the fruit and transport module files. Hopefully we will do this for you.
4. The sandbox is like scratch paper – it is an executable project to do whatever you want.

1.4 Module Divisions: Library versus Application

You will notice that the main computational code is in a static library called /transport. In the future we may have other static library modules called /sediment /nonconservative and to deal with I/O.

By and large, the drivers (test code and application) know about the libraries and not vice versa. The drivers do the setup and request the transport library to solve individual time steps. There are a couple nuanced points about this:

- The state_variables module properly allocates and houses state variables. It is like a utility for drivers tests and applications. You should use the state_variables module and then pass the variables into the solver as arguments.
- The advection.advect() function takes a callback for a source term that must be provided by the driver/application. The reason I did this is that we want to be able to stipulate different source terms for different tests. We will probably want to implement advection.replace_boundary_flux() the same way.

When you use callbacks, you should do so at the level of the whole grid. In other words compute_source is perfectly efficient as long as its job is to compute source over the grid, not a point.

- Other common jobs done by the driver do not affect the interior of advect(). For instance, the initial condition, and hydro_data(). These have to be user supplied. I have tried to include interface blocks for these routines in the library, but not an implementation.

1.5 Why All The _lo and _hi Data Structures??? Isn't this Wasteful?

You may notice that I have included a lot of data structures that look like flux_lo(icell) and flux_hi(icell) and in some cases (but not all) you may feel that there is a redundancy

because it seems like flux_hi(icell) == flux_hi(icell+1). This, in fact will be true for some data structures for some data. However, when we get to channel networks it will not be true. At that point we will be stuck with a choice between:

- face-centered arrays whose mappings are non-obvious, ie: flux(hi_face(ncell)).
- flux_hi(ncell), which is still perfectly well-defined still.

1.6 Tests

I put the Fruit testing framework here directly as a static library to reduce complexity of the project files. We can move it later.

The test_transport project is the first battery of tests. The tests in test_gradient.f90 are fairly complete for that module. You will find a test driver called test_transport_-driver.f90 The driver program there is where you would add new tests. Try to have one for every medium sized routine. For algorithms, the best integration tests are either accuracy compared to a known solution or (better) convergence.

Remember, unit tests are silent. They should also be complete and catch "corner" cases. Look at the flux limiter test. I spot checked one "ordinary" case in the middle of the array in a gentle area where the limited flux is just the centered difference. More importantly, I tested both ends of the array and I looked at reversals of sign in both directions and places where the limiter is in use due to big jumps on the hi and lo side.

This is usually a sobering experience.

1.7 Todo Tags

In the code you should use !todo: for any notes to yourself about things to do later. You will already see a few in there. Do not use any other format for this job.

1.8 Style Notes:

Please try to adhere to these guidelines as well as trying to get a sense of the code already provided and matching its style. Although we encourage you to "not worry about it" when you turn in code every other week, it would be nice if you took 1-2 hours to go over this as a checklist.

- Try to follow the subroutine style. The gradient module is probably the best example for routines and state_variables module shows how to encapsulate variables and document them.

- All routines are implicit none. Write this out, do not use compiler options.
- Do not pass data to subroutines by "use" statements. Put data in the signature. Imagine that you want to test the routine later out of context. Long signatures are fine.
- Routine names are lower_underscore. Try to name them with one well chosen word or two, avoiding redundancy. Do not abbreviate unless:
  - the abbreviation is very easy to interpret and the original is taxing. For instance, conc is fine for concentration.
  - the abbreviation is standard English, in which case you should always use it: e.g. abbr for abbreviation.
  - the abbreviation is well established in the code. For instance I use lo and hi to show that a quantity is on face on the lo (n-1) or hi (n+1) side of the cell.
- Variables are also lower_underscore.
- Some constants associated with precision like LARGEREAL and STM_REAL are all caps.
  - OK, this is a bit consistent but the typesafe scalar values for numbers (one, two, half, etc) in the stm_precision module are lower because... well it just looks way better
- Use the typesafe scalar values for numbers that have them: x = two$*$y. This avoids having to keep track of the "d" for double precision and prevents real-double conversion.
- Prefer subroutines to functions.
- Argument lists should not be long. Use continutation lines (&) when there is more than one line full. See the examples.
- Section off the arguments to all routines. See gradient.f90 for examples
- Section off routines from one another using //////. See gradient.f90
- Declare the intent for every variable.
- Use real(STM_REAL) for floating point. Avoid hardwiring.
- Do not add things like "last modified" that have to be maintained by hand. Let version control do this.
- Put the licence at the top.
- Initialize variables to LARGEREAL or put LARGEREAL in indexes that won't get touched. For instance, if the derivative and value arrays are the same side and you do a "lo-side" difference, there is nothing to put in the first index. Set that to LARGEREAL. The reason for this is that if it gets initialized to something reasonable (the compiler often choses zero, at least in debug mode) it can lead to bugs that look deceptively reasonable.

- In some cases you may see a_ prepended to an argument name. It means "argument". This was done when a global or module variable clashes with the argument name. It isn't required (local names take precedence) but it prevents ambiguity.
- Indent with spaces, and consistently within a file. Use F90 free-form syntax. Again, see gradient.f90.
- Try not to check in code with commented out sections.

Avoid comments when clear code names will suffice. Comments should outline the intent of little blocks of code when that intent is not obvious.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Modules Index

3.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 4

Data Type Index

4.1 Data Types List

Here are the data types with brief descriptions:

Chapter 5

Module Documentation

5.1 test_transport

Packages

- package example_hydro_data

  Sample flow fields for tests.

- package example_initial_conditions

  Sample initial conditions for tests!>.

- package example_sources

  Simple source term subroutines for testing.

- package test_diffusion

  todo: write tests for explicit operator and implicit diffusion

- package test_extrapolate

  Testing of the extrapolate step.

- package test_gradient

  Test module for gradients and limiters.

- package test_transport_driver

  Main program unit for testing advection.

- package test_uniform_flow

  Test of transport in uniform flow.

5.2 transport

Packages

- package advection

  Module orchestrating the advection scheme. The main routine in the module is advection().

- package diffusion

  Explicit and implicit diffusion operators go here.

- package gradient

  Module containing routines for calculating differences and limiters.

- package hydro_data_if

  Hydrodynamics interface to be fulfilled by driver or application.

- package source_if

  Source interface to be fulfilled by driver or application.

- package state_variables

  Defines state variables for the transport problem as well as functions to allocate them.

- package stm_precision

  Definition of model precision, precision-related constants and special values.

Chapter 6

Module Documentation

6.1 Package advection

Module orchestrating the advection scheme. The main routine in the module is advection().

Functions/Subroutines

- subroutine advect (mass, mass_prev, flow_lo, flow_hi, area, area_prev, area_lo, area_hi, ncell, nvar, time, dt, dx, compute_source)

  Advection plus sources for a time step. The final argument to advect is a callback for computing the source term, which should conform to the source_if interface The algoritm looks like this:
  - *Convert to primitive variables*
  - *Extrapolate to faces*
    - *difference()*
    - *limiter()*
    - *extrapolate()*
  - *upwind()*
  - *compute_flux()*
  - *replace_boundary_flux()*
  - *Compute conservative divergence*
  - *Apply divergence in conservative_update.*

- subroutine extrapolate (conc_lo, conc_hi, conc, flow, area, ncell, nvar, time, dt, dx)

  Extrapolate primitive data from cell center at the old time to cell edges at the half time. The extrapolation is done by a Taylor series in time and space in which an explicit discretization of the PDE is used to represent the time part.

- subroutine compute_flux (flux_lo, flux_hi, conc_lo, conc_hi, flow_lo, flow_hi, ncell, nvar)

  Compute the fluxes naively – no boundary considerations yet.

- subroutine replace_boundary_flux (flux_lo, flux_hi, conc_lo, conc_hi, flow_lo, flow_hi, ncell, nvar, time, dt, dx)

  Replace original calculated flux at boundary locations todo: figure out if the arguments are right and move this routine to the application – it should just be an interface like sources and hydro_data Also, eventually have to think out channel network.

- subroutine upwind (flux_lo, flux_hi, conc_lo, conc_hi, flow_lo, flow_hi, ncell, nvar)

  Given flows and concentrations on either side of a cell calculate the upwind mass flux.

6.1.1 Detailed Description

Module orchestrating the advection scheme. The main routine in the module is advection().

6.1.2 Function/Subroutine Documentation

6.1.2.1 subroutine advection.advect (real(STM_REAL),dimension(ncell,nvar),intent(out) *mass*, real(STM_REAL),dimension(ncell,nvar),intent(in) *mass_prev*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *area*, real(STM_REAL),dimension(ncell,nvar),intent(in) *area_prev*, real(STM_REAL),dimension(ncell,nvar),intent(in) *area_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *area_hi*, integer,intent(in) *ncell*, integer,intent(in) *nvar*, real(STM_REAL),intent(in) *time*, real(STM_REAL),intent(in) *dt*, real(STM_REAL),intent(in) *dx*, compute_source)

Advection plus sources for a time step. The final argument to advect is a callback for computing the source term, which should conform to the source_if interface The algoritm looks like this:

- Convert to primitive variables
- Extrapolate to faces

  - difference()
  - limiter()
  - extrapolate()
- upwind()
- compute_flux()
- replace_boundary_flux()
- Compute conservative divergence
- Apply divergence in conservative_update.

Parameters:

mass mass concentration at new time

mass_prev mass concentration at new time

flow_lo flow on lo side of cells centered in time

flow_hi flow on hi side of cells centered in time

area cell-centered area at new time

area_prev cell-centered area at old time??

area_lo lo side area centered in time

area_hi hi side area centered in time

ncell Number of cells

nvar Number of variables

time current time

dt current time step

dx spatial step

6.1.2.2 subroutine advection.compute_flux (real(STM_REAL),dimension(ncell,nvar),intent(out) *flux_lo*, real(STM_REAL),dimension(ncell,nvar),intent(out) *flux_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_hi*, integer,intent(in) *ncell*, integer,intent(in) *nvar*)

Compute the fluxes naively – no boundary considerations yet.

Parameters:

flux_lo Flux on lo face at half time

flux_hi Flux on hi face at half time

conc_lo upwinded concentration at half time at lo face

conc_hi upwinded concentration at half time at hi face

flow_lo time-centered flow at lo face

flow_hi time-centered flow at hi face

ncell Number of cells

nvar Number of variables

6.1.2.3 subroutine advection.extrapolate (real(STM_REAL),dimension(ncell,nvar),intent(out) *conc_lo*, real(STM_REAL),dimension(ncell,nvar),intent(out) *conc_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow*, real(STM_REAL),dimension(ncell,nvar),intent(in) *area*, integer,intent(in) *ncell*, integer,intent(in) *nvar*, real(STM_REAL),intent(in) *time*, real(STM_REAL),intent(in) *dt*, real(STM_REAL),intent(in) *dx*)

Extrapolate primitive data from cell center at the old time to cell edges at the half time. The extrapolation is done by a Taylor series in time and space in which an explicit discretization of the PDE is used to represent the time part.

Parameters:

conc_lo estimate from this cell extrapolated to lo face at half time

conc_hi estimate from this cell extrapolated to hi face at half time

conc cell centered conc at old time

flow cell-centered flow at old time

area cell-centered area at old time

ncell Number of cells

nvar Number of variables

time time

dt length of current time step being advanced

dx spatial step

6.1.2.4 subroutine advection.replace_boundary_flux (real(STM_REAL),dimension(ncell,nvar),intent(inout) *flux_lo*, real(STM_REAL),dimension(ncell,nvar),intent(inout) *flux_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_hi*, integer,intent(in) *ncell*, integer,intent(in) *nvar*, real(STM_REAL),intent(in) *time*, real(STM_REAL),intent(in) *dt*, real(STM_REAL),intent(in) *dx*)

Replace original calculated flux at boundary locations todo: figure out if the arguments are right and move this routine to the application – it should just be an interface like sources and hydro_data Also, eventually have to think out channel network.

Parameters:

flux_lo Flux on lo face at half time

flux_hi Flux on hi face at half time

conc_lo upwinded concentration at half time at lo face

conc_hi upwinded concentration at half time at hi face

flow_lo time-centered flow at lo face

flow_hi time-centered flow at hi face

ncell Number of cells

nvar Number of variables

time time

dt length of current time step

dx spatial step

6.1.2.5 subroutine advection.upwind (real(STM_REAL),dimension(ncell,nvar),intent(out) *flux_lo*, real(STM_REAL),dimension(ncell,nvar),intent(out) *flux_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *flow_hi*, integer,intent(in) *ncell*, integer,intent(in) *nvar*)

Given flows and concentrations on either side of a cell calculate the upwind mass flux.

Parameters:

flux_lo flux on lo side of cells centered in time

flux_hi flux on hi side of cells centered in time

conc_lo conc on lo side of cells centered in time

conc_hi conc on hi side of cells centered in time

flow_lo flow on lo side of cells centered in time

flow_hi flow on hi side of cells centered in time

ncell Number of cells

nvar Number of variables

6.2 Package diffusion

Explicit and implicit diffusion operators go here.

6.2.1 Detailed Description

Explicit and implicit diffusion operators go here.

6.3 Package example_hydro_data

Sample flow fields for tests.

Functions/Subroutines

- subroutine constant_uniform (flow, flow_lo, flow_hi, ncell, time, q_const)

  Constant uniform flow todo: needs to satisfy the hydro_data_if interface, has to change.

6.3.1 Detailed Description

Sample flow fields for tests.

6.3.2 Function/Subroutine Documentation

6.3.2.1 subroutine example_hydro_data.constant_uniform (real(STM_REAL),dimension(ncell),intent(out) *flow*, real(STM_REAL),dimension(ncell),intent(out) *flow_lo*, real(STM_REAL),dimension(ncell),intent(out) *flow_hi*, integer,intent(in) *ncell*, real(STM_REAL),intent(in) *time*, real(STM_REAL),intent(in) *q_const*)

Constant uniform flow todo: needs to satisfy the hydro_data_if interface, has to change.

Parameters:

flow cell-centered flow at time

flow_lo flow on lo side of cells at time

flow_hi flow on hi side at time

ncell Number of cells

time time

q_const constant flow to be used

6.4 Package example_initial_conditions

Sample initial conditions for tests!>.

Functions/Subroutines

- subroutine gaussian_ic

  Initialize the concentration fields with gaussian plume todo: abstract the part that does the shape somewhere so that it is easy to write a corresponding routine to verify simple advection and clarify this in the documentation.

- subroutine step_ic

  Initialize the concentration fields with a step function.

6.4.1 Detailed Description

Sample initial conditions for tests!>.

6.5 Package example_sources

Simple source term subroutines for testing.

Functions/Subroutines

- subroutine no_source (source, conc, area, flow, a_ncell, a_nvar)

  Empty source implementation.

6.5.1 Detailed Description

Simple source term subroutines for testing.

6.5.2 Function/Subroutine Documentation

6.5.2.1 subroutine example_sources.no_source (real(STM_REAL),dimension(a_ncell,a_nvar),intent(out) *source*, real(STM_REAL),dimension(a_ncell,a_nvar),intent(in) *conc*, real(STM_REAL),dimension(a_ncell,a_nvar),intent(in) *area*, real(STM_REAL),dimension(a_ncell,a_nvar),intent(in) *flow*, integer,intent(in) *a_ncell*, integer,intent(in) *a_nvar*)

Empty source implementation.

Parameters:

source cell centered source

conc Concentration

area area at source

flow flow at source location

a_ncell Number of cells

a_nvar Number of variables

6.6 Package gradient

Module containing routines for calculating differences and limiters.

Functions/Subroutines

- subroutine difference (grad_lo, grad_hi, grad_center, vals, ncell, nvar)

  Calculate the undivided lo, hi and centered differences.

- subroutine limiter (grad_lim, grad_lo, grad_hi, grad_center, ncell, nvar)

  Apply a flux limiter (van Leer) given one-sided and centered gradients.

6.6.1 Detailed Description

Module containing routines for calculating differences and limiters.

6.6.2 Function/Subroutine Documentation

6.6.2.1 subroutine gradient.difference (real(STM_REAL),dimension(ncell,nvar),intent(out) *grad_lo*, real(STM_REAL),dimension(ncell,nvar),intent(out) *grad_hi*, real(STM_REAL),dimension(ncell,nvar),intent(out) *grad_center*, real(STM_REAL),dimension(ncell,nvar),intent(in) *vals*, integer,intent(in) *ncell*, integer,intent(in) *nvar*)

Calculate the undivided lo, hi and centered differences.

Parameters:

grad_lo gradient on lo side, LARGEREAL in first index

grad_hi hi side (n+1) minus (n) LARGEREAL for last index

grad_center centered diff, LARGEREAL for undefined boundary cells

vals data to be differenced

ncell Number of cells

nvar Number of variables

6.6.2.2 subroutine gradient.limiter (real(STM_-REAL),dimension(ncell,nvar),intent(out) *grad_lim*, real(STM_REAL),dimension(ncell,nvar),intent(in) *grad_lo*, real(STM_REAL),dimension(ncell,nvar),intent(in) *grad_hi*, real(STM_REAL),dimension(ncell,nvar),intent(in) *grad_center*, integer,intent(in) *ncell*, integer,intent(in) *nvar*)

Apply a flux limiter (van Leer) given one-sided and centered gradients.

Parameters:

grad_lim limited gradient

grad_lo gradient on lo side, LARGEREAL in first index

grad_hi hi side (n+1) minus (n) LARGEREAL for last index

grad_center centered diff, LARGEREAL for undefined boundary cells

ncell Number of cells

nvar Number of variables

6.7 Package hydro_data_if

Hydrodynamics interface to be fulfilled by driver or application.

Data Types

- interface hydro_data

  Generic interface for fetching hydrodynamic data.

6.7.1 Detailed Description

Hydrodynamics interface to be fulfilled by driver or application.

6.8 Package source_if

Source interface to be fulfilled by driver or application.

Data Types

- interface compute_source

  Calculate source.

6.8.1 Detailed Description

Source interface to be fulfilled by driver or application.

6.9 Package state_variables

Defines state variables for the transport problem as well as functions to allocate them.

Functions/Subroutines

- subroutine allocate_state (a_ncell, a_nvar)

  Allocate the state variables consistently including concentration and hydrodynamics. Initial value is LARGEREAL.

- subroutine deallocate_state

  Deallocate the state variables including concentration and hydrodynamics and reset ncell and nvar to zero.

Variables

- integer ncell

  number of computation cells

- integer nvar

  number of variables

- real(STM_REAL), dimension(:,:), allocatable, save conc

  Concentration in the current/new time step, dimensions (ncell, nvar).

- real(STM_REAL), dimension(:,:), allocatable, save conc_prev

  Concentration in the previous time step, dimensions (ncell, nvar).

- real(STM_REAL), dimension(:), allocatable, save area

  Cell-centered area dimensions (ncell).

- real(STM_REAL), dimension(:), allocatable, save area_lo

  Face area on lo side of cell (so this is cell-indexed), dimensions (ncell) todo: we should talk about this design a bit do not change this without talking to Eli.

- real(STM_REAL), dimension(:), allocatable, save area_hi

  Face-centered area on hi side of cell (so this is cell-indexed), dimensions (ncell).

- real(STM_REAL), dimension(:), allocatable, save flow_lo

  face-centered flow on lo side of cell (so this is cell-indexed), dimensions (ncell) todo: we should talk about this design a bit

- real(STM_REAL), dimension(:), allocatable, save flow_hi

  face-centered flow on hi side of cell (so this is cell-indexed), dimensions (ncell)

6.9.1 Detailed Description

Defines state variables for the transport problem as well as functions to allocate them.

6.9.2 Function/Subroutine Documentation

6.9.2.1 subroutine state_variables.allocate_state (integer,intent(in) *a_ncell*, integer,intent(in) *a_nvar*)

Allocate the state variables consistently including concentration and hydrodynamics. Initial value is LARGEREAL.

Parameters:

a_ncell Number of requested cells

a_nvar Number of constituents

6.10 Package stm_precision

Definition of model precision, precision-related constants and special values.

Variables

- integer, parameter STM_REAL = 8

  Precision of REAL.

- real(STM_REAL) minus = -1.D0

  Real constant -1. properly typed.

- real(STM_REAL) zero = 0.D0

  Real constant 0. properly typed.

- real(STM_REAL) one = 1.D0

  Real constant 1. properly typed.

- real(STM_REAL) two = 2.D0

  Real constant 2. properly typed.

- real(STM_REAL) half = 5.D-1

  Real constant 0.5 properly typed.

- real(STM_REAL) fourth = 2.5D-1

  Real constant 0.25 properly typed.

- real(STM_REAL) LARGEREAL = 1.23456789D8

  Absurd high value, for initialization and for marking undefined data in calculations. This makes bugs easier to spot.

6.10.1 Detailed Description

Definition of model precision, precision-related constants and special values.

6.11 Package test_diffusion

todo: write tests for explicit operator and implicit diffusion

6.11.1 Detailed Description

todo: write tests for explicit operator and implicit diffusion

6.12 Package test_extrapolate

Testing of the extrapolate step.

Functions/Subroutines

- subroutine **test_extrapolation**

6.12.1 Detailed Description

Testing of the extrapolate step.

6.13 Package test_gradient

Test module for gradients and limiters.

Functions/Subroutines

- subroutine test_gradient_calc

  Tesitng routine for undivided differences.

- subroutine test_limiter

  Tesitng routine for van Leer limiter.

6.13.1 Detailed Description

Test module for gradients and limiters.

6.14 Package test_transport_driver

Main program unit for testing advection.

6.14.1 Detailed Description

Main program unit for testing advection.

6.15 Package test_uniform_flow

Test of transport in uniform flow.

Functions/Subroutines

- subroutine test_uniform_flow_advection ()

  Subroutine that runs a small advective simulation.

6.15.1 Detailed Description

Test of transport in uniform flow.

Chapter 7

Data Type Documentation

7.1 source_if.compute_source Interface Reference

Calculate source.

Public Member Functions

- subroutine compute_source (source, conc, area, flow, ncell, nvar, time)

  Generic interface for calculating source that should be fulfilled by client programs.

7.1.1 Detailed Description

Calculate source.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 subroutine source_if.compute_source.compute_source (real(STM_REAL),dimension(ncell,nvar),intent(out) *source*, real(STM_REAL),dimension(ncell,nvar),intent(in) *conc*, real(STM_REAL),dimension(ncell),intent(in) *area*, real(STM_REAL),dimension(ncell),intent(in) *flow*, integer,intent(in) *ncell*, integer,intent(in) *nvar*, real(STM_REAL),intent(in) *time*)

Generic interface for calculating source that should be fulfilled by client programs.

Parameters:

source cell centered source

conc Concentration

area area at source

flow flow at source location

ncell Number of cells

nvar Number of variables

time flow at source location

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

- D:/Delta/models/dsm2_stm/src/transport/source_if.f90

7.2 hydro_data_if.hydro_data Interface Reference

Generic interface for fetching hydrodynamic data.

Public Member Functions

- subroutine hydro_data8 (flow, flow_lo, flow_hi, area, area_lo, area_hi, ncell, time, dt)

  Fill in hydrodynamic data. This data might be calculated from a function or provided by another module Note that continuity must be satisfied between time steps. The implementation must be provided by the driver or application.

7.2.1 Detailed Description

Generic interface for fetching hydrodynamic data.

7.2.2 Member Function/Subroutine Documentation

7.2.2.1 subroutine hydro_data_if.hydro_data.hydro_data8 (real(STM_REAL),dimension(ncell),intent(out) *flow*, real(STM_REAL),dimension(ncell),intent(out) *flow_lo*, real(STM_REAL),dimension(ncell),intent(out) *flow_hi*, real(STM_REAL),dimension(ncell),intent(out) *area*, real(STM_REAL),dimension(ncell),intent(out) *area_lo*, real(STM_REAL),dimension(ncell),intent(out) *area_hi*, integer,intent(in) *ncell*, real(STM_REAL),intent(in) *time*, real(STM_REAL),intent(in) *dt*)

Fill in hydrodynamic data. This data might be calculated from a function or provided by another module Note that continuity must be satisfied between time steps. The implementation must be provided by the driver or application.

Parameters:

flow cell and time centered flow

flow_lo lo face flow, time centered

flow_hi hi face flow, time centered

area cell center area, old time

area_lo area lo face, time centered

area_hi area hi face, time centered

ncell number of cells

time time of request "old time"

dt time step for

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

- D:/Delta/models/dsm2_stm/src/transport/hydro_data_if.f90