G^3M - Global Gradient Groundwater Model Documentation

Release 1.0

Robert Reinecke

Apr 24, 2018

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- search

CHAPTER

TWO

QUICKSTART: THE MODEL FRAMEWORK

The global gradient-based groundwater model framework G^{3}M-f is an extesible model framework. Its main purpose is to be used as a main bilding block for the global groundwater mode G^{3}M. G^{3}M is a newly developed gradient-based groundwater model which adapts MODFLOW [@harbaugh2005modflow] principles for the globalscale. It is written in C++ and intended to be coupled to the global hydraulic model WaterGAP (http://watergap.de), but can also be used for regional groundwater models and coupling to other hydraulic models. While it is intended to be used as a in memory coupled model it is also capable of running a standard standalone groundwater model.

2.1 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

2.1.1 Prerequisites

```
clang >= 3.8 with openMP (currently gcc is not supported)
libboost >= 1.56
libGMP
libGtest
```

2.1.2 Build

```
mkdir build
cd build
cmake ../
make
```

2.1.3 How to use

Center building stone for the framework is the GW_interface connecting any model with the groundwater code. Implement this interface if you want to couple your model to G^{3}M-f or build a custom standalone application.

```
class GW_Interface {
    public:
        virtual ~GW_Interface() {}

        virtual void
        loadSettings() = 0;

        virtual void
```

(continues on next page)

(continued from previous page)

```
        setupSimulation() = 0;

        virtual void
        writeData() = 0;

        virtual void
        simulate() = 0;
};
```

2.2 Write out data

Writeout of data is specified by a JSON file called out.json. If you want to add custom fields you can do so in src/DataProcessing/DataOutput.

```
{
  "output": {
    "StaticResult": [
      {
        "name": "wtd",
        "type": "csv",
        "field": "DepthToWaterTable",
        "ID": "false",
        "position": "true"
      }
    ],
    "InnerIteration": {
    },
    "OuterIteration": {
    }
  }
}
```

2.3 Config model

In order to configure the model variables you can simply change the .json file. Allowing you to change the convergence criteria and the location for your input files.

```
{
  "config": {
    "model_config": {
      "nodes": "grid_simple.csv",
      "row_cols": "true",
      "stadystate": "true",
      "numberofnodes": 100,
      "threads": 1,
      "layers": 2,
      "confinement": [
        "false",
        "true"
      ],
      "cache": "false",
      "adaptivestepsize": "false",
      "boundarycondition": "SeaLevel",
      "sensitivity": "false"
    },
    "numerics": {
```

(continues on next page)

(continued from previous page)

```
      "solver": "PCG",
      "iterations": 500,
      "inner_itter": 10,
      "closingcrit": 1e-8,
      "headchange": 0.0001,
      "damping": "false",
      "min_damp": 0.01,
      "max_damp": 0.5,
      "stepsize": "daily"
    },
  "input": {
    "data_config": {
      "k_from_lith": "true",
      "k_ocean_from_file": "false",
      "specificstorage_from_file": "false",
      "specificyield_from_file": "false",
      "k_river_from_file": "true",
      "aquifer_depth_from_file": "false",
      "initial_head_from_file": "true",
      "data_as_array": "false"
    },
    "default_data": {
      "initial_head": 5,
      "K": 0.008,
      "oceanK": 800,
      "aquifer_thickness": [
        10,
        10
      ],
      "anisotropy": 10,
      "specificyield": 0.15,
      "specificstorage": 0.000015
    },
    "data": {
      "recharge": "recharge_simple.csv",
      "elevation": "elevation_simple.csv",
      "rivers": "rivers_simple.csv",
      "lithologie": "lithology_simple.csv",
      "river_conductance": "rivers_simple.csv",
      "initial_head": "heads_simple.csv"
    }
  }
  }
}
```

2.4 Building a simple model

The following shows the code for a simple model loop running a steady-state model with daily timesteps.

```
void StandaloneRunner::simulate() {
    Simulation::Stepper stepper = Simulation::Stepper(_eq, Simulation::DAY, 1);
    for (Simulation::step step : stepper) {
        LOG(userinfo) << "Running a steady state step";
        step.first->toogleSteadyState();
        step.first->solve();
        sim.printMassBalances();
    }
    DataProcessing::DataOutput::OutputManager("data/out_simple.json", sim).write();
```

(continues on next page)

(continued from previous page)

```
    //sim.save();
}
```

2.5 Deployment in other models

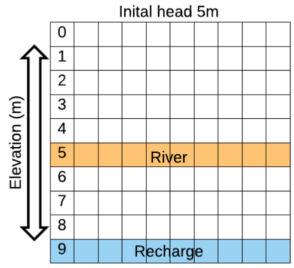
Just implement the GW_interface and provide a DataReader.

2.6 Running the tests

Automated tests consits of gunit test which are compiled automatically with the attached cmake file. You can run them by executing the test executable.

```
runUnitTests
```

2.6.1 Running a simple model



The following picture shows the conceptual example model:

After compilation run:

```
simple_model
```

It will yield a depth to water table CSV file called wtd.csv for a simple model.

2.7 Built With

- Eigen3 - Doing the math magic
- GTest - Test framework
- libboost - C++ magic
- OpenMP - Accelerator und Multi-Core support
- GMP - Large numbers

2.8 Contributing

Please read CONTRIBUTING.md for details on our code of conduct, and the process for submitting pull requests to us.

2.9 Versioning

We use SemVer for versioning. For the versions available, see the tags on this repository.

2.10 Authors and Contributors

- **Robert Reinecke** - *Initial work*

2.11 License

This project is licensed under the GNU General Public License - see the LICENSE file for details. Please note that the code contains a modified version of the Eigen3 library which is published under the MPL 2.0.

2.12 Acknowledgments

- Modflow 2005 for their great documentation
- Eigen3 for their awesome framework

CHAPTER

THREE

G^3M FRAMEWORK

The following describes the classes and modules that G^3M provides for building a groundwater model. The framework is separated into 6 packages:

- DataProcessing
- Logging
- Misc
- Model
- Simulation
- Solver

In order to implement any model the following interface has to be implemented:

class GlobalFlow::**GW_Interface**

Main interface to the groundwater model.

Interface to the groundwater simulation Implement me!

Subclassed by *GlobalFlow::GlobalStandaloneRunner*, GlobalFlow::StandaloneRunner

Public Functions

virtual GlobalFlow::*GW_Interface*`~GW_Interface`()

virtual void GlobalFlow::*GW_Interface*`loadSettings`() = 0

Read general simulation settings e.g. Options

virtual void GlobalFlow::*GW_Interface*`setupSimulation`() = 0

Do additional work required for a running simulation

virtual void GlobalFlow::*GW_Interface*`writeData`() = 0

How should the data be written out

void GlobalFlow::*GW_Interface*`setupCallBack`(Callback *callback*)

Set up the calback for model coupling Could be inefficient

Parameters

- `callback`: a callback function

virtual void GlobalFlow::*GW_Interface*`simulate`() = 0

Simulate/Run the model

3.1 DataProcessing

Data processing is mainly concerned with providing utilities for reading in data (*DataReader*) or writing out data. New types of outputs can be implemented in the *OutputFactory*.

class GlobalFlow::**DataReader**

Interface that needs to be implemented for reading in required data for the model.

Subclassed by *GlobalFlow::DataProcessing::GlobalDataReader*, GlobalFlow::DataProcessing::SimpleDataReader

Public Functions

virtual GlobalFlow::*DataReader*~**DataReader**()

Virt destructor -> interface

void GlobalFlow::*DataReader***initNodes**(NodeVector *nodes*)

Initialize internal ref to node vetor.

Parameters

- `nodes`: The vector of nodes

virtual void GlobalFlow::*DataReader***readData**(Simulation::*Options* *op*) = 0

Entry point for reading simulation data.

Attention This method needs to be implemented!

Note *readData()* is called by simulation at startup

Parameters

- `op`: Options object

template <**class** Fun>
void GlobalFlow::*DataReader***loopFiles**(std::string *path*, std::vector<std::string> *files*, Fun *fun*)

Generic method for looping through files inside a directory and applying a generic function.

Parameters

- `path`: the directory
- `files`: a vector of files
- `fun`: a function that is applied e.g. reading the data

int GlobalFlow::*DataReader***check**(int *globid*)

Check weather id exists in the simulation.

Return i the position in the node vector

Parameters

- `globid`: Global identifier, can be different from position in node vector

template <**class** ProcessDataFunction>
void GlobalFlow::*DataReader***readTwoColumns**(std::string *path*, ProcessDataFunction *processData*)

Read data from a two-column csv file and apply function to data.

Parameters

- `path`: to the csv file

- `processData`: A processing function e.g. upscaling of data

void GlobalFlow::*DataReader***readZeroPointFiveToFiveMin**(std::string *path*)
Creates a mapping of 0.5° ArcIDs to a list of contained 5' GlobIDs.

Parameters

- `path`: to file

const std::unordered_map<int, std::vector<int>> &GlobalFlow::*DataReader***getArcIDMapping**()
provides acccess to mapping of different resolutions

Return <ARCID(0.5°), vector<GlobalID(5')>>

const std::unordered_map<int, int> &GlobalFlow::*DataReader***getGlobIDMapping**()
provides access to mapping of data ids to position in node vector

Return <GlobalID, ID>

std::string GlobalFlow::*DataReader***buildDir**(std::string *path*)
Builds a corect path from the base dir.

Return A path based on the base dir

Parameters

- `path`: The relative path from the config

3.1.1 DataOutput

FieldCollector

enum GlobalFlow::DataProcessing::DataOutput::**FieldType**
What kind of data is collected Internal data fields that can be written out.

Values:

GlobalFlow::DataProcessing::DataOutput**ID**
Internal position

GlobalFlow::DataProcessing::DataOutput**ARCID**
Data ID

GlobalFlow::DataProcessing::DataOutput**AREA**
Area of the node

GlobalFlow::DataProcessing::DataOutput**CONDUCT**
Hydraulic conductivity of the node

GlobalFlow::DataProcessing::DataOutput**ELEVATION**
Elevation of the node

GlobalFlow::DataProcessing::DataOutput**SLOPE**
Slope in the node

GlobalFlow::DataProcessing::DataOutput**X**

GlobalFlow::DataProcessing::DataOutput**Y**
Postion of the node in X and Y

GlobalFlow::DataProcessing::DataOutput**HEAD**
Hydraulic head

GlobalFlow::DataProcessing::DataOutput**EQ_HEAD**
The equilibrium head -> inital head

GlobalFlow::DataProcessing::DataOutput**IN**

GlobalFlow::DataProcessing::DataOutput**OUT**
All in and outflows

GlobalFlow::DataProcessing::DataOutput**EQ_FLOW**
Lateral flows based on the equilibrium head

GlobalFlow::DataProcessing::DataOutput**LATERAL_FLOW**
Sum of all lateral flows of node

GlobalFlow::DataProcessing::DataOutput**LATERAL_OUT_FLOW**
Only lateral out flows

GlobalFlow::DataProcessing::DataOutput**WETLANDS**
Is there a wetland?

GlobalFlow::DataProcessing::DataOutput**LAKES**
Is there a lake?

GlobalFlow::DataProcessing::DataOutput**FLOW_HEAD**
Surface water body head

GlobalFlow::DataProcessing::DataOutput**RECHARGE**
GW recharge rate

GlobalFlow::DataProcessing::DataOutput**DYN_RIVER**
Is there a dynamic river?

GlobalFlow::DataProcessing::DataOutput**NODE_VELOCITY**
Velocity of lateral gw flow

GlobalFlow::DataProcessing::DataOutput**RIVER_OUT**
Outflow to river

GlobalFlow::DataProcessing::DataOutput**RIVER_IN**
Inflow from river

GlobalFlow::DataProcessing::DataOutput**WTD**
Depth to groundwater table based on elevation

GlobalFlow::DataProcessing::DataOutput**RIVER_CONDUCT**
Conductance of riverbed

GlobalFlow::DataProcessing::DataOutput**DRAIN_CONDUCT**
Conductance of drainbed

GlobalFlow::DataProcessing::DataOutput**WETLAND_CONDUCT**
Conductance of wetland

GlobalFlow::DataProcessing::DataOutput**GL_WETLAND_CONDUCT**
Conductance of global wetland

GlobalFlow::DataProcessing::DataOutput**LAKE_CONDUCT**
Conductance of lake

GlobalFlow::DataProcessing::DataOutput**OCEAN_OUT**
Boundary condition outflow

GlobalFlow::DataProcessing::DataOutput**GL_WETLAND_OUT**
Global wetland outflow

GlobalFlow::DataProcessing::DataOutput**WETLAND_OUT**
Wetland outflow

GlobalFlow::DataProcessing::DataOutput**LAKE_OUT**
Lake outflow

GlobalFlow::DataProcessing::DataOutput**GL_WETLAND_IN**
Global wetland inflow

GlobalFlow::DataProcessing::DataOutput**WETLAND_IN**
Wetland inflow

GlobalFlow::DataProcessing::DataOutput**LAKE_IN**
Lake inflow

GlobalFlow::DataProcessing::DataOutput**NON_VALID**

class GlobalFlow::DataProcessing::DataOutput::**FieldCollector**
Iterates over internal fields and searches for data to be written out This is currently relatively inefficient

Public Types

using GlobalFlow::DataProcessing::DataOutput::*FieldCollector***pos_v** = std::vector<std::pair<double, double>>

Public Functions

GlobalFlow::DataProcessing::DataOutput::*FieldCollector***FieldCollector** (*FieldType enumField*)

The constructor

Parameters

- `enumField`: What data should be collected

GlobalFlow::DataProcessing::DataOutput::*FieldCollector***FieldCollector** ()

Note Should not be used

pos_v GlobalFlow::DataProcessing::DataOutput::*FieldCollector***getPositions** (Simulation::*Simulation* **const** &*simulation*)

get the positions of the nodes

Return A vector of positions

Parameters

- `simulation`: The simulation

std::vector<large_num> GlobalFlow::DataProcessing::DataOutput::*FieldCollector***getIds** (Simulation::*Simulation* **const** &*simulation*)

Get the data ids of the nodes.

Return A vector of IDs

Parameters

- `simulation`: The simulation

template **<typename** T>
data_vector<T> GlobalFlow::DataProcessing::DataOutput::*FieldCollector***get** (Simulation::*Simulation* **const** &*simulation*)

Collects the data from simulation nodes

Note Relatively inefficient currently

Return The collected data

Parameters

- `simulation`: The simulation

OutputFactory

template <typename T>
class GlobalFlow::DataProcessing::DataOutput::**OutputInterface**

Writes data to a file

Subclassed by *GlobalFlow::DataProcessing::DataOutput::CSVOutput< T >*, *GlobalFlow::DataProcessing::DataOutput::GFS_JSONOutput< T >*, *GlobalFlow::DataProcessing::DataOutput::NETCDFOutput< T >*

Public Functions

virtual GlobalFlow::DataProcessing::DataOutput::*OutputInterface*~**OutputInterface**()

virtual void GlobalFlow::DataProcessing::DataOutput::*OutputInterface***write**(path *filePath*, bool *printID*, bool *printXY*, std::vector<T> *data*, pos_v *p*, a_vector *ids*) = 0

Needs to be implemented

Parameters

- `filePath`:
- `printID`: Bool
- `printXY`: Bool
- `data`: Data vector
- `p`: *Position* vector

template <typename T>
class GlobalFlow::DataProcessing::DataOutput::**CSVOutput**

Writes data to a CSV file

Inherits from *GlobalFlow::DataProcessing::DataOutput::OutputInterface< T >*

Public Functions

void GlobalFlow::DataProcessing::DataOutput::*CSVOutput***write**(path *filePath*, bool *printID*, bool *printXY*, std::vector<std::pair<double, double>> *data*, pos_v *p*, a_vector *ids*)

void GlobalFlow::DataProcessing::DataOutput::*CSVOutput***write**(path *filePath*, bool *printID*, bool *printXY*, std::vector<bool> *data*, pos_v *p*, a_vector *ids*)

void GlobalFlow::DataProcessing::DataOutput::*CSVOutput***write**(path *filePath*, bool *printID*, bool *printXY*, std::vector<double> *data*, pos_v *p*, a_vector *ids*)

void GlobalFlow::DataProcessing::DataOutput::*CSVOutput***write**(path *filePath*, bool *printID*, bool *printXY*, std::vector<std::string> *data*, pos_v *p*, a_vector *ids*)

template <typename T>

class GlobalFlow::DataProcessing::DataOutput::**NETCDFOutput**
Writes data to a NETCDF file

Inherits from *GlobalFlow::DataProcessing::DataOutput::OutputInterface< T >*

Public Functions

void GlobalFlow::DataProcessing::DataOutput::*NETCDFOutput***write** (path *filePath*, bool *printID*, bool *printXY*, std::vector<T> *data*, pos_v *p*, a_vector *ids*)

Needs to be implemented

Parameters

- `filePath`:
- `printID`: Bool
- `printXY`: Bool
- `data`: Data vector
- `p`: *Position* vector

template <typename T>
class GlobalFlow::DataProcessing::DataOutput::**GFS_JSONOutput**
Inherits from *GlobalFlow::DataProcessing::DataOutput::OutputInterface< T >*

Public Functions

void GlobalFlow::DataProcessing::DataOutput::*GFS_JSONOutput***write** (path *filePath*, bool *printID*, bool *printXY*, std::vector<T> *data*, pos_v *p*, a_vector *ids*)

Needs to be implemented

Parameters

- `filePath`:
- `printID`: Bool
- `printXY`: Bool
- `data`: Data vector
- `p`: *Position* vector

template <typename T>
class GlobalFlow::DataProcessing::DataOutput::**OutputFactory**

Public Static Functions

static *OutputInterface*<T> *GlobalFlow::DataProcessing::DataOutput::*OutputFactory***getOutput** (OutputType *type*)

OutputManager

class GlobalFlow::DataProcessing::DataOutput::**OutputManager**

Public Functions

GlobalFlow::DataProcessing::DataOutput::*OutputManager***OutputManager**(path *output_spec_path*, Simulation::*Simulation* **const** &*sim*)

void GlobalFlow::DataProcessing::DataOutput::*OutputManager***write**()
Visits all registered output options and triggers write.

3.2 Logging

Provides readable logging facilities

class GlobalFlow::Logging::**Logger**
Inherits from GlobalFlow::Logging::LoggerInterface

Public Functions

GlobalFlow::Logging::*Logger***Logger**()

3.3 Misc

Contains some deprecated helpers for iterating different grid solutions not documented here.

Functions

void **NANChecker**(**const** double &*value*, std::string *message*)

double **roundValue**(double *valueToRound*)
template <**typename** T, typename… *Args*>
std::unique_ptr<T> **make_unique**(Args&&... *args*)
template <**class** T>
Is<T> **is**(T *d*)

class NANInSolutionException
Inherits from exception

Private Functions

virtual const char **NANInSolutionException***what**() **const**

class InfInSolutionException
Inherits from exception

Private Functions

virtual const char **InfInSolutionException***what**() **const**
template <**class** T>
struct Is
#include <Helpers.hpp> http://stackoverflow.com/questions/15181579

Public Functions

bool *Is*`in` (T *a*)

`template <class` Arg, class… *Args*>
bool *Is*`in` (Arg *a*, Args... *args*)

Public Members

T *Is*`d_`

`class Position`

Public Functions

Position`Position` (double *lat*, double *lon*)

Public Members

`const` double *Position*`lat` = {0}

`const` double *Position*`lon` = {0}

3.4 Model

ExternalFlow

`class` GlobalFlow::Model::`ExternalFlow`
TODO add flow equation here

Public Functions

GlobalFlow::Model::*ExternalFlow*`ExternalFlow` (int *id*, FlowType *type*, t_meter *flowHead*, t_s_meter_t *cond*, t_meter *bottom*)

GlobalFlow::Model::*ExternalFlow*`ExternalFlow` (int *id*, t_vol_t *recharge*, FlowType *type*)
Only for RECHARGE FAST_SURFACE_RUNOFF

GlobalFlow::Model::*ExternalFlow*`ExternalFlow` (int *id*, t_meter *flowHead*, t_meter *bottom*, t_vol_t *evapotrans*)
Constructor for Evapotranspiration.

Return

Parameters

- `id`:
- `flowHead`:
- `bottom`:
- `evapotrans`:

bool GlobalFlow::Model::*ExternalFlow*`flowIsHeadDependant` (t_meter *head*) `const`
Check if flow can be calculated on the right hand side

Return Bool

Parameters

- `head`: The current hydraulic head

t_s_meter_t GlobalFlow::Model::*ExternalFlow*`getP`(t_meter *head*, t_meter *eq_head*, t_vol_t *recharge*, t_dim *slope*, t_vol_t *eqFlow*) `const`

The head dependant part of the external flow equation

Return

Parameters

- `head`: The current hydraulic head
- `eq_head`: The equilibrium head
- `recharge`: The current recharge
- `slope`:
- `eqFlow`:

t_vol_t GlobalFlow::Model::*ExternalFlow*`getQ`(t_meter *head*, t_meter *eq_head*, t_vol_t *recharge*, t_dim *slope*, t_vol_t *eqFlow*) `const`

The head independant part of the external flow equation

Return

Parameters

- `head`:
- `eq_head`:
- `recharge`:
- `slope`:
- `eqFlow`:

FlowType GlobalFlow::Model::*ExternalFlow*`getType`() `const`

t_meter GlobalFlow::Model::*ExternalFlow*`getBottom`() `const`

t_vol_t GlobalFlow::Model::*ExternalFlow*`getRecharge`() `const`

t_meter GlobalFlow::Model::*ExternalFlow*`getFlowHead`() `const`

t_s_meter_t GlobalFlow::Model::*ExternalFlow*`getDyn`(t_vol_t *current_recharge*, t_meter *eq_head*, t_meter *head*, t_vol_t *eq_flow*) `const`

t_meter GlobalFlow::Model::*ExternalFlow*`getRiverDiff`(t_meter *eqHead*) `const`

t_s_meter_t GlobalFlow::Model::*ExternalFlow*`getConductance`() `const`

int GlobalFlow::Model::*ExternalFlow*`getID`() `const`

void GlobalFlow::Model::*ExternalFlow*`setMult`(double *mult*)

FluidMechanics

`class` GlobalFlow::Model::`FluidMechanics`

Provides helper functions for conductance calulcations

Public Functions

GlobalFlow::Model::*FluidMechanics***FluidMechanics**()

t_meter GlobalFlow::Model::*FluidMechanics***calcDeltaV**(t_meter *head*, t_meter *elevation*, t_meter *depth*)

Used to calculate if a cell is dry

quantity<MeterSquaredPerTime> GlobalFlow::Model::*FluidMechanics***calculateHarmonicMeanConductance**(Flo *flow*

Calculates the horizontal flow between two nodes.

Return A weighted conductance value for the flow between two nodes Calculates the harmonic mean conductance between two nodes. $C = 2 EdgeLenght_1 { (TR_1 TR_2)}{(TR_1 EdgeLenght_1 + TR_2 EdgeLenght_2)}$

Parameters

- `flow`: a touple of inputs about the aquifer

double GlobalFlow::Model::*FluidMechanics***smoothFunction__NWT**(t_meter *elevation*, t_meter *verticalSize*, t_meter *head*)

Simple smoother function to buffer iteration steps in NWT approach

Return smoothed head

Parameters

- `elevation`:
- `verticalSize`:
- `head`:

quantity<MeterSquaredPerTime> GlobalFlow::Model::*FluidMechanics***getHCOF**(bool *steadyState*, quantity<Dimensionless> *stepModifier*, t_s_meter *storageCapacity*, t_s_meter_t *P*)

Get the coeffiecients for storage and P components

Return HCOF

Parameters

- `steadyState`:
- `stepModifier`:
- `storageCapacity`:
- `P`:

quantity<MeterSquaredPerTime> GlobalFlow::Model::*FluidMechanics***calculateVerticalConductance**(FlowInpu *flow*)

Calculates the vertical flow between two nodes

Return the vertical conductance

Parameters

- `flow`: a touple of inputs about the aquifer double GlobalFlow::Model::*FluidMechanics*`getDerivate__NWT` (t_meter *elevation*, t_meter *verticalSize*, t_meter *head*)

Calculate derivates for NWT approach

Return

Parameters

- `elevation`:
- `verticalSize`:
- `head`:

NodeInterface

class GlobalFlow::Model::**NodeInterface**

Interface defining required fields for a node. A node is the central comutational and spatial unit. A simulated area is seperated into a discrete raster of cells or nodes (seperate computational units which stay in contact to ech other). *Is* equal to 'cell'.

Nodes can be of different physical property e.g. different size.

Subclassed by GlobalFlow::Model::StandardNode, GlobalFlow::Model::StaticHeadNode

Public Functions

GlobalFlow::Model::*NodeInterface*`NodeInterface` (NodeVector *nodes*, double *lat*, double *lon*, t_s_meter *area*, large_num *ArcID*, large_num *ID*, t_vel *K*, int *stepModifier*, double *aquiferDepth*, double *anisotropy*, double *specificYield*, double *specificStorage*, bool *confined*)

Constructor of abstract class *NodeInterface*.

Parameters

- `nodes`: Vector of all other existing nodes
- `lat`: The latitude
- `lon`: The Longitude
- `area`: Area in m^2
- `ArcID`: Unique ARC-ID specified by Kassel
- `ID`: Internal ID = *Position* in vector
- `K`: Hydraulic conductivity in meter/day (default)
- `stepModifier`: Modfies default step size of day (default=1)
- `aquiferDepth`: Vertical size of the cell
- `anisotropy`: Modifier for vertical conductivity based on horizontal
- `specificYield`: Yield of storage for dewatered conditions
- `specificStorage`: Specific storage - currently for confined and unconfined
- `confined`: *Is* node in a confined layer

virtual GlobalFlow::Model::*NodeInterface*`~NodeInterface` ()

large_num GlobalFlow::Model::*NodeInterface*`getID` ()

void GlobalFlow::Model::*NodeInterface*`setElevation` (t_meter *elevation*)

Set elevation on top layer and propagate to lower layers.

Parameters

- `elevation`: The top elevation (e.g. from DEM)

void GlobalFlow::Model::*NodeInterface***setSlope** (double *slope_percent*)
Set slope from data on all layers Slope input is in % but is required as absolut thus: slope = sloper_percent / 100.

Parameters

- `slope`:

void GlobalFlow::Model::*NodeInterface***setEfold** (double *efold*)
Set e-folding factor from data on all layers.

Parameters

- `e-fold`:

void GlobalFlow::Model::*NodeInterface***setEqHead** (t_meter *wtd*)
Calculated equilibrium groundwater-head from eq_wtd Assumes that if initialhead = false that the eq_head is also used as initial head.

Parameters

- `head`:

template <class HeadType>
FlowInputHor GlobalFlow::Model::*NodeInterface***createDataTuple** (map_itter *got*)

FlowInputVert GlobalFlow::Model::*NodeInterface***createDataTuple** (map_itter *got*)

template <class HeadType>
t_vol_t GlobalFlow::Model::*NodeInterface***calcLateralFlows** (bool *onlyOut*)
Calculate the lateral groundwater flow to the neighbouring nodes Generic function used for calulating equlibrium and current step flow

Return

t_vol_t GlobalFlow::Model::*NodeInterface***getEqFlow** ()
Calculate the equlibrium lateral flows

Return eq lateral flow

t_vol_t GlobalFlow::Model::*NodeInterface***getLateralFlows** ()
Get the current lateral flow

Return

t_vol_t GlobalFlow::Model::*NodeInterface***getLateralOutFlows** ()
Get the current lateral out flows

Return

bool GlobalFlow::Model::*NodeInterface***resetFloodingHead** ()
Cuts off all heads above surface elevation.

Warning Should only be used in spinn up phase!

Return Bool if node was reset

void GlobalFlow::Model::*NodeInterface***scaleRiverConduct** ()
Scales river conduct by 50%.

Warning Should only be used in spinn up phase

void GlobalFlow::Model::*NodeInterface*`updateHeadChange`()
Update the current head change (in comparison to last time step)

Note Should only be caled at end of timestep

void GlobalFlow::Model::*NodeInterface*`initHead_t0`()

void GlobalFlow::Model::*NodeInterface*`setHead_direct`(double *head*)

t_vel GlobalFlow::Model::*NodeInterface*`getK__pure`()

t_vel GlobalFlow::Model::*NodeInterface*`getK`()
Get hydraulic conductivity.

Return hydraulic conductivity (scaled by e-folding)

t_vel GlobalFlow::Model::*NodeInterface*`getK_vertical`()
Get hydraulic vertical conductivity.

Return hydraulic conductivity scaled by anisotropy (scaled by e-folding)

void GlobalFlow::Model::*NodeInterface*`setK`(t_vel *conduct*)
Modify hydraulic conductivity (applied to all layers below)

Parameters

- `New`: conductivity (if e-folding enabled scaled on layers)

void GlobalFlow::Model::*NodeInterface*`setK_direct`(t_vel *conduct*)
Modify hydraulic conductivity (no e-folding, no layers)

Parameters

- `New`: conductivity

t_c_meter GlobalFlow::Model::*NodeInterface*`getOUT`()
Get all outflow since simulation start.

t_c_meter GlobalFlow::Model::*NodeInterface*`getIN`()
Get all inflow since simulation start.

void GlobalFlow::Model::*NodeInterface*`toogleStadyState`(bool *onOFF*)
Toogle steady state simulation.

Parameters

- `onOFF`: true=on Turns all storage equations to zero with no timesteps

void GlobalFlow::Model::*NodeInterface*`updateStepSize`(double *mod*)

t_s_meter GlobalFlow::Model::*NodeInterface*`getStorageCapacity`()
Storage capacity based on yield or specific storage.

Return Potential flow budget when multiplied by head change Uses an 0.001m epsilon to determine if a water-table condition is present. If the layer is confined or not in water-table condition returns primary capacity.

ExternalFlow &GlobalFlow::Model::*NodeInterface*`getExternalFlowByName`(FlowType *type*)
Get and external flow by its FlowType.

Return Ref to external flow

Parameters

- `type`: The flow type

Exceptions

- `OutOfRangeException`:

t_vol_t GlobalFlow::Model::*NodeInterface*`getExternalFlowVolumeByName`(FlowType *type*)
Get and external flow volume by its FlowType.

Return Flow volume

Parameters

- `type`: The flow type

t_vol_t GlobalFlow::Model::*NodeInterface*`getTotalStorageFlow`()
Get flow budget based on head change.

Return Flow volume Note: Water entering storage is treated as an outflow (-), that is a loss of water from the flow system while water released from storage is treated as inflow (+), that is a source of water to the flow system

t_vol_t GlobalFlow::Model::*NodeInterface*`calculateExternalFlowVolume`(**const** *ExternalFlow* *&flow*)
Get flow budget of a specific external flows.

Return Flow volume Note: Water entering storage is treated as an outflow (-), that is a loss of water from the flow system while water released from storage is treated as inflow (+), that is a source of water to the flow system

Parameters

- `&flow`: A external flow

t_vol_t GlobalFlow::Model::*NodeInterface*`calculateDewateredFlow`()
Caluclate dewatered flow.

Return Flow volume per time If a cell is dewatered but below a saturated or partly saturated cell: this calculates the needed additional exchange volume

t_vol_t GlobalFlow::Model::*NodeInterface*`getCurrentIN`()
Get all current IN flow.

Return Flow volume

t_vol_t GlobalFlow::Model::*NodeInterface*`getCurrentOUT`()
Get all current OUT flow.

Return Flow volume

void GlobalFlow::Model::*NodeInterface*`saveMassBalance`()
Tell cell to save its flow budget.

void GlobalFlow::Model::*NodeInterface*`setNeighbour` (large_num *ID*, NeighbourPosition *neighbour*)

Add a neighbour.

Parameters

- `ID`: The internal ID and position in vector
- `neighbour`: The position relative to the cell

int GlobalFlow::Model::*NodeInterface*`getNumofNeighbours` ()

NodeInterface *GlobalFlow::Model::*NodeInterface*`getNeighbour` (NeighbourPosition *neighbour*)

Get a neighbour by position.

Return Pointer to cell object

Parameters

- `neighbour`: The position relative to the cell

int GlobalFlow::Model::*NodeInterface*`addExternalFlow` (FlowType *type*, t_meter *flowHead*, double *cond*, t_meter *bottom*)

At an external flow to the cell.

Return Number assigned by cell to flow

Parameters

- `type`: The flow type
- `flowHead`: The flow head
- `cond`: The conductance
- `bottom`: The bottom of the flow (e.g river bottom)

void GlobalFlow::Model::*NodeInterface*`removeExternalFlow` (FlowType *type*)

Remove an external flow to the cell by id.

Parameters

- `ID`: The flow id

bool GlobalFlow::Model::*NodeInterface*`hasTypeOfExternalFlow` (FlowType *type*)

Check for an external flow by type.

Return bool

Parameters

- `type`: The flow type

void GlobalFlow::Model::*NodeInterface*`updateUniqueFlow` (double *amount*, FlowType *flow* = RECHARGE)

Updates GW recharge Curently assumes only one recharge as external flow!

Parameters

- `amount`: The new flow amount

void GlobalFlow::Model::*NodeInterface*`scaleDynamicRivers` (double *mult*)

Scale dyn rivers for sensitivity

Parameters

- `mult`:

void GlobalFlow::Model::*NodeInterface***updateExternalFlowConduct** (double *amount*, FlowType *type*)

Update wetlands, lakes.

Parameters

- `amount`:
- `type`:

void GlobalFlow::Model::*NodeInterface***updateExternalFlowFlowHead** (double *amount*, FlowType *type*)

Multiplies flow head for Sensitivity An. wetlands, lakes, rivers.

Parameters

- `amount`:
- `type`:

void GlobalFlow::Model::*NodeInterface***updateLakeBottoms** (double *amount*)

Update lake bottoms Used for sensitivity.

Parameters

- `amount`:

bool GlobalFlow::Model::*NodeInterface***hasRiver** ()

Check for type river.

Return bool

bool GlobalFlow::Model::*NodeInterface***hasOcean** ()

Check for type ocean.

Return bool

t_vol_t GlobalFlow::Model::*NodeInterface***getQ** ()

Get Q part of flow equations.

Return volume over time

t_s_meter_t GlobalFlow::Model::*NodeInterface***getP** ()

Get P part of flow equations.

Return volume over time

t_vol_t GlobalFlow::Model::*NodeInterface***calculateNotHeadDependandFlows** ()

Get flow which is not groundwater head dependent.

Return volume over time Flow can be added to constant flows on right side of the equations If head is above river bottom for example

std::unordered_map<large_num, t_s_meter_t> GlobalFlow::Model::*NodeInterface***getJacobian** ()

The jacobian entry for the cell (NWT approach)

Return map <CellID,Conductance>

std::unordered_map<large_num, t_s_meter_t> GlobalFlow::Model::*NodeInterface***getConductance**()
The matrix entry for the cell.

Return map <CellID,Conductance> The left hand side of the equation

t_vol_t GlobalFlow::Model::*NodeInterface***getRHS**()
The right hand side of the equation.

Return volume per time

double GlobalFlow::Model::*NodeInterface***getRHS__NWT**()
The right hand side of the equation (NWT)

Return volume per time

void GlobalFlow::Model::*NodeInterface***setHead**(t_meter *head*)

t_meter GlobalFlow::Model::*NodeInterface***calcInitialHead**(t_meter *initialParam*)

bool GlobalFlow::Model::*NodeInterface***isStaticNode**()

PhysicalProperties &GlobalFlow::Model::*NodeInterface***getProperties**()

void GlobalFlow::Model::*NodeInterface***enableNWT**()

template <**typename** CompareFunction>
t_vol_t GlobalFlow::Model::*NodeInterface***getNonStorageFlow**(CompareFunction *compare*)
Caluclate non storage related in and out flow.

Return Flow volume

quantity<Velocity> GlobalFlow::Model::*NodeInterface***getVelocity**(map_itter *pos*)
Calculate the lateral flow velocity

Return

Parameters

- `pos`:

std::pair<double, double> GlobalFlow::Model::*NodeInterface***getVelocityVector**()
Calculate flow velocity for flow tracking Vx and Vy represent the flow velocity in x and y direction. A negative value represents a flow in the oposite direction.

Return Velocity vector (x,y)

Public Members

bool GlobalFlow::Model::*NodeInterface***cached** = {false}
Calculated equilibrium flow to neighbouring cells Static thus calculated only once.

Depends on: K in cell and eq_head in all 6 neighbours

t_vol_t GlobalFlow::Model::*NodeInterface***eq_flow** = {0 * si::cubic_meter / day}

class GlobalFlow::Model::*NodeInterface***NodeNotFoundException**
Inherits from exception

3.5 Simulation

Options

class GlobalFlow::Simulation::**Options**

Reads simulation options from a JSON file Defines getters and setters for options

Public Types

enum GlobalFlow::Simulation::*Options*`BoundaryCondition`

Values:

GlobalFlow::Simulation::*Options*`CONSTANT_HEAD_SEA_LEVEL`

GlobalFlow::Simulation::*Options*`CONSTANT_HEAD_NEIGHBOUR`

GlobalFlow::Simulation::*Options*`STATIC_HEAD_SEA_LEVEL`

Public Functions

void GlobalFlow::Simulation::*Options*`setClosingCrit`(double *crit*)

void GlobalFlow::Simulation::*Options*`setDamping`(bool *set*)

bool GlobalFlow::Simulation::*Options*`isDampingEnabled`()

double GlobalFlow::Simulation::*Options*`getMinDamp`()

double GlobalFlow::Simulation::*Options*`getMaxDamp`()

double GlobalFlow::Simulation::*Options*`getMaxHeadChange`()

bool GlobalFlow::Simulation::*Options*`isConfined`(int *layer*)

vector<bool> GlobalFlow::Simulation::*Options*`getConfinements`()

BoundaryCondition GlobalFlow::Simulation::*Options*`getBoundaryCondition`()

bool GlobalFlow::Simulation::*Options*`isSensitivity`()

bool GlobalFlow::Simulation::*Options*`isKFromLith`()

bool GlobalFlow::Simulation::*Options*`isKOceanFile`()

bool GlobalFlow::Simulation::*Options*`isSpecificStorageFile`()

bool GlobalFlow::Simulation::*Options*`isSpecificYieldFile`()

bool GlobalFlow::Simulation::*Options*`isKRiverFile`()

bool GlobalFlow::Simulation::*Options*`isAquiferDepthDile`()

string GlobalFlow::Simulation::*Options*`getKDir`()

string GlobalFlow::Simulation::*Options*`getKRiverDir`()

string GlobalFlow::Simulation::*Options*`getKOceanDir`()

string GlobalFlow::Simulation::*Options*`getSSDir`()

string GlobalFlow::Simulation::*Options*`getSYDir`()

string GlobalFlow::Simulation::*Options*`getAQDepthDir`()

bool GlobalFlow::Simulation::*Options*`isRowCol()`

int GlobalFlow::Simulation::*Options*`getInnerItter()`

long GlobalFlow::Simulation::*Options*`getNumberOfNodes()`

int GlobalFlow::Simulation::*Options*`getNumberOfLayers()`

int GlobalFlow::Simulation::*Options*`getMaxIterations()`

double GlobalFlow::Simulation::*Options*`getConverganceCriteria()`

string GlobalFlow::Simulation::*Options*`getSolverName()`

bool GlobalFlow::Simulation::*Options*`disableDryCells()`

string GlobalFlow::Simulation::*Options*`getNodesDir()`

string GlobalFlow::Simulation::*Options*`getElevation()`

string GlobalFlow::Simulation::*Options*`getEfolding()`

string GlobalFlow::Simulation::*Options*`getEqWTD()`

string GlobalFlow::Simulation::*Options*`getSlope()`

string GlobalFlow::Simulation::*Options*`getBlue()`

vector<string> GlobalFlow::Simulation::*Options*`getElevation_A()`

vector<string> GlobalFlow::Simulation::*Options*`getEfolding_a()`

vector<string> GlobalFlow::Simulation::*Options*`getEqWTD_a()`

vector<string> GlobalFlow::Simulation::*Options*`getSlope_a()`

vector<string> GlobalFlow::Simulation::*Options*`getBlue_a()`

string GlobalFlow::Simulation::*Options*`getRecharge()`

string GlobalFlow::Simulation::*Options*`getLithology()`

string GlobalFlow::Simulation::*Options*`getRivers()`

string GlobalFlow::Simulation::*Options*`getGlobalLakes()`

string GlobalFlow::Simulation::*Options*`getGlobalWetlands()`

string GlobalFlow::Simulation::*Options*`getLocalLakes()`

string GlobalFlow::Simulation::*Options*`getLocalWetlands()`

string GlobalFlow::Simulation::*Options*`getMapping()`

int GlobalFlow::Simulation::*Options*`getThreads()`

`const` bool GlobalFlow::Simulation::*Options*`adaptiveStepsizeEnabled()`

`const` int GlobalFlow::Simulation::*Options*`getStepsizeModifier()`

bool GlobalFlow::Simulation::*Options*`cacheEnabled()`

int GlobalFlow::Simulation::*Options*`getInitialHead()`

double GlobalFlow::Simulation::*Options*`getInitialK()`

double GlobalFlow::Simulation::*Options*`getOceanConduct()`

vector<int> GlobalFlow::Simulation::*Options***getAquiferDepth**()

double GlobalFlow::Simulation::*Options***getAnisotropy**()

double GlobalFlow::Simulation::*Options***getSpecificYield**()

double GlobalFlow::Simulation::*Options***getSpecificStorage**()

void GlobalFlow::Simulation::*Options***load**(**const** std::string &*filename*)

void GlobalFlow::Simulation::*Options***save**(**const** std::string &*filename*)

Simulation

class GlobalFlow::Simulation::**Simulation**

The simulation class which holds the equation, options and data instance Further contains methods for calulating the mass balance and sensitivity methods TODO: Clean me up!

Public Types

enum GlobalFlow::Simulation::*Simulation***Flows**

Values:

GlobalFlow::Simulation::*Simulation***RIVERS** = 1

GlobalFlow::Simulation::*Simulation***DRAINS**

GlobalFlow::Simulation::*Simulation***RIVER_MM**

GlobalFlow::Simulation::*Simulation***LAKES**

GlobalFlow::Simulation::*Simulation***WETLANDS**

GlobalFlow::Simulation::*Simulation***GLOBAL_WETLANDS**

GlobalFlow::Simulation::*Simulation***RECHARGE**

GlobalFlow::Simulation::*Simulation***FASTSURFACE**

GlobalFlow::Simulation::*Simulation***NAG**

GlobalFlow::Simulation::*Simulation***STORAGE**

GlobalFlow::Simulation::*Simulation***GENERAL_HEAD_BOUNDARY**

Public Functions

GlobalFlow::Simulation::*Simulation***Simulation**(*Options op*, *DataReader *reader*)

GlobalFlow::Simulation::*Simulation***Simulation**()

Solver::*Equation* *GlobalFlow::Simulation::*Simulation***getEquation**()

void GlobalFlow::Simulation::*Simulation***save**()

std::string GlobalFlow::Simulation::*Simulation***NodeInfosByID**(unsigned long *nodeID*)

Get basic node information by its id

Return A string of information

Parameters

- `nodeID`:

template <int *FieldNum*>

std::string GlobalFlow::Simulation::*Simulation*`getFlowSumByIDs` (std::array<int, FieldNum> *ids*)
Get budget per node

Return

Parameters

- `ids`:

std::string GlobalFlow::Simulation::*Simulation*`NodeFlowsByID` (unsigned long *nodeID*)
Return all external flows seperatly

`template <class` FunOut, `class` FunIn>
MassError GlobalFlow::Simulation::*Simulation*`getError` (FunOut *fun1*, FunIn *fun2*)
Calulate the mass error

Return

Parameters

- `fun1`: Function to get OutFlow
- `fun2`: Function to get InFlow

MassError GlobalFlow::Simulation::*Simulation*`getMassError` ()
Get the total mass balance

Return

MassError GlobalFlow::Simulation::*Simulation*`getCurrentMassError` ()
Get the mass balance for the current step

Return

double GlobalFlow::Simulation::*Simulation*`getLossToRivers` ()
Get the flow lost to external flows

Return

`template <class` Fun>
MassError GlobalFlow::Simulation::*Simulation*`getError` (Fun *fun*)
Decide if its an In or Outflow

Return

Parameters

- `fun`:

string GlobalFlow::Simulation::*Simulation*`getFlowByName` (*Flows* *flow*)
Helper function for printing the mass balance for each flow

Return

Parameters

- `flow`:

void GlobalFlow::Simulation::*Simulation*`printMassBalances` ()
Prints all mass balances

`const` double GlobalFlow::Simulation::*Simulation*`calcRecharge` (`const` double *recharge*, `const` double &*area*)

DataReader *GlobalFlow::Simulation::*Simulation*`getDataReader` ()

NodeVector &GlobalFlow::Simulation::*Simulation*`getNodes` ()

void GlobalFlow::Simulation::*Simulation*`scaleByIds` (vector<int> *ids*, string *field*, double *mult*)

template <class Fun, **class** ChangeFunction>
void GlobalFlow::Simulation::*Simulation***scaleByFunction** (Fun *fun*, ChangeFunction *apply*)

template <class Fun>
void GlobalFlow::Simulation::*Simulation***scaleByFunction** (Fun *fun*, string *field*, double *mult*)

Helper function for sensitivity

Parameters

- `fun`:
- `field`:
- `mult`:

std::vector<int> GlobalFlow::Simulation::*Simulation***readMultipliersPerID** (string *path*)

Expects a file with global_ID, parameter, multiplier Multiplier is log scaled

Return a vector of node-ids (used to write out heads in correct order)

Parameters

- `path`: to csv file

void GlobalFlow::Simulation::*Simulation***readMultipliers** (string *path*)

Scale values for sensitivity analysis

Parameters

- `path`:

void GlobalFlow::Simulation::*Simulation***writeResiduals** (string *path*)

Get the residuals of the current iteration

Parameters

- `path`:

template <typename DataArray>
void GlobalFlow::Simulation::*Simulation***updateGWRechargeFromWaterGAP** (DataArray *field*, short *month*, int *numberOfGridCells*)

Coupling function for WaterGAP

Note Under development

Parameters

- `field`:
- `month`:
- `numberOfGridCells`:

Public Members

NodeVector GlobalFlow::Simulation::*Simulation***nodes**

Stepper

class GlobalFlow::Simulation**::Stepper**

holding the simulation iterator

Inherits from GlobalFlow::Simulation::AbstractStepper

Public Functions

GlobalFlow::Simulation::*Stepper***Stepper**(Solver::*Equation* **eq*, **const** TimeFrame *time*, **const** size_t *steps*)

virtual Solver::*Equation* *GlobalFlow::Simulation::*Stepper***get**(int *col*) **const**

Iterator GlobalFlow::Simulation::*Stepper***begin**() **const**

Iterator GlobalFlow::Simulation::*Stepper***end**() **const**

const TimeFrame GlobalFlow::Simulation::*Stepper***getStepSize**()

3.6 Solver

Equation

class GlobalFlow::Solver::**Equation**
finite difference equation Should only be accessed through the stepper

Public Types

typedef Eigen::MatrixXd::Scalar GlobalFlow::Solver::*Equation***Scalar**

typedef Matrix<*Scalar*, Dynamic, 1> GlobalFlow::Solver::*Equation***VectorType**

Public Functions

GlobalFlow::Solver::*Equation***Equation**(large_num *numberOfNodes*, NodeVector *nodes*, Simulation::*Options* *options*)

GlobalFlow::Solver::*Equation***~Equation**()

void GlobalFlow::Solver::*Equation***solve**()
Solve the current iteration step

Solve *Equation*

int GlobalFlow::Solver::*Equation***getItter**()

Return The number of iterations

double GlobalFlow::Solver::*Equation***getError**()

Return The current residual error

GlobalFlow::Solver::*Equation***Equation**(**const** *Equation*&)

Equation &GlobalFlow::Solver::*Equation***operator=**(**const** *Equation*&)

VectorXd GlobalFlow::Solver::*Equation***getResults**()

bool GlobalFlow::Solver::*Equation***toogleSteadyState**()
Toogle the steady-state in all nodes

Return

void GlobalFlow::Solver::*Equation***updateStepSize**(size_t *mod*)
Set the correct stepsize (default is DAY)

Parameters

- `mod`:

VectorType GlobalFlow::Solver::*Equation***`getResiduals`**()

void GlobalFlow::Solver::*Equation***`updateClosingCrit`**(double *crit*)

Friends

std::ostream &**`operator<<`**(std::ostream &*os*, Equation &*eq*)
Helper to write out current residuals

Return

Parameters

- `os`:
- `eq`:

AdaptiveDamping .. doxygenclass:: GlobalFlow::Solver::AdaptiveDamping

CHAPTER

FOUR

G³M STEADY-STATE MODEL

The implementation of the steady state-model along with the input data e.g. yearly average recharge, consists of two classes:

1. The main class which implements the *Groundwater Interface*
2. A data reader implementing the *Data Reader Interface*, which specifies how to read in the data

Main

class GlobalFlow::**GlobalStandaloneRunner**
A standalone global steady-state groundwater model.

Inherits from *GlobalFlow::GW_Interface*

Public Functions

GlobalFlow::*GlobalStandaloneRunner***GlobalStandaloneRunner**()
Default constructor.

void GlobalFlow::*GlobalStandaloneRunner***loadSettings**()
Read general simulation settings e.g. Options

void GlobalFlow::*GlobalStandaloneRunner***setupSimulation**()
Do additional work required for a running simulation

void GlobalFlow::*GlobalStandaloneRunner***simulate**()
Simulate/Run the model

void GlobalFlow::*GlobalStandaloneRunner***writeData**()
How should the data be written out

Private Functions

set<int> GlobalFlow::*GlobalStandaloneRunner***getMapping**()
Helper function for arcID mappings.

Return a set of arcIDs

Private Members

Solver::*Equation* *GlobalFlow::*GlobalStandaloneRunner***_eq**

Simulation::*Options* GlobalFlow::*GlobalStandaloneRunner***op**

Simulation::*Simulation* GlobalFlow::*GlobalStandaloneRunner***sim**

DataProcessing::*GlobalDataReader* *GlobalFlow::*GlobalStandaloneRunner***reader**

Global Data Reader

class GlobalFlow::DataProcessing::**GlobalDataReader**

This class provides methods for loading large input data The paths are specified in the json file in in the data folder.

Inherits from *GlobalFlow::DataReader*

Public Types

using GlobalFlow::DataProcessing::*GlobalDataReader***Matrix** = std::vector<std::vector<T>>

Public Functions

GlobalFlow::DataProcessing::*GlobalDataReader***GlobalDataReader**(int *step*)

Constructor.

Parameters

- `step`: Daily, Monthly, ...

void GlobalFlow::DataProcessing::*GlobalDataReader***readData**(Simulation::*Options* *op*)

Entry point for reading simulation data.

Attention This method needs to be implemented!

Note *readData()* is called by simulation at startup

Parameters

- `op`: Options object

Matrix<int> GlobalFlow::DataProcessing::*GlobalDataReader***readGrid**(NodeVector *nodes*, std::string *path*, int *numberOfNodes*, double *defaultK*, double *aquiferDepth*, double *anisotropy*, double *specificYield*, double *specificStorage*, bool *confined*)

Method for already gridded defintions - that is structered in row and column.

Note Structured in row, col

Return A Matrix of computational nodes

Parameters

- `nodes`: Vector of nodes
- `path`: Path to read definitions from
- `numberOfNodes`: The number of expected computation nodes
- `defaultK`: The default conductivity
- `aquiferDepth`: The default depth per cell
- `anisotropy`: The default relation of vertical and horizontal conductivity
- `specificYield`: The default specific yield
- `specificStorage`: The default specific storage

- `confined`: If node is part of a confined layer?

int GlobalFlow::DataProcessing::*GlobalDataReader*`readLandMask`(NodeVector *nodes*, std::string *path*, int *numberOfNodes*, double *defaultK*, double *aquiferDepth*, double *anisotropy*, double *specificYield*, double *specificStorage*, bool *confined*)

Initial readin of node definitions - without col and row.

Note Without col and row Reads a csv file with x and y coordinates for predefined grid of cells

Return

Parameters

- `nodes`: Vector of nodes
- `path`: Path to read definitions from
- `numberOfNodes`: The number of expected computation nodes
- `defaultK`: The default conductivity
- `aquiferDepth`: The default depth per cell
- `anisotropy`: The default relation of vertical and horizontal conductivity
- `specificYield`: The default specific yield
- `specificStorage`: The default specific storage
- `confined`: If node is part of a confined layer?

void GlobalFlow::DataProcessing::*GlobalDataReader*`readOceanK`(std::string *path*)

Read in a custom defintion for the ocean boundary.

Parameters

- `path`: Where to read from

void GlobalFlow::DataProcessing::*GlobalDataReader*`readRiver`(std::string *path*)

Read in a custom river defintion file Structured as: global_ID, Head, Bottom, Conduct.

Parameters

- `path`: Where to read the file from

void GlobalFlow::DataProcessing::*GlobalDataReader*`readElevation`(std::string *path*, std::vector<std::string> *files*)

Read elevation data from a specified path Used for multiple files.

Note !Uses setElevation() function. Should only be called after all layers are build as it affects layers below

Parameters

- `path`: Where to read the file from
- `files`: If different files for different regions are given

void GlobalFlow::DataProcessing::*GlobalDataReader*`readElevation`(std::string *path*)

Read elevation data from a specified path.

Note !Uses setElevation() function. Should only be called after all layers are build as it affects layers below

Parameters

- `path`: Where to read the file from

void GlobalFlow::DataProcessing::*GlobalDataReader***readSlope** (std::string *path*, std::vector<std::string> *files*)

Read slope data from a specified path.

Parameters

- `path`: Where to read the file from
- `files`: If different files for different regions are given

void GlobalFlow::DataProcessing::*GlobalDataReader***readEfold** (std::string *path*, std::vector<std::string> *files*)

Read e-folding data from a specified path.

Parameters

- `path`: Where to read the file from
- `files`: If different files for different regions are given

void GlobalFlow::DataProcessing::*GlobalDataReader***readEqWTD** (std::string *path*)

Read equilibrium water-table information used for the dynamic river computation.

Parameters

- `path`: Where to read the file from

void GlobalFlow::DataProcessing::*GlobalDataReader***readGWRecharge** (std::string *path*)

Read difuse gw-recharge.

Parameters

- `path`: Where to read the file from

void GlobalFlow::DataProcessing::*GlobalDataReader***readConduct** (std::string *path*)

Read cell conductance defintion.

Note currently does check if val > 10 m/day

Parameters

- `path`: Where to read the file from

template <typename ConversionFunction>
void GlobalFlow::DataProcessing::*GlobalDataReader***readGWRechargeMapping** (std::string *path*, ConversionFunction *convertToRate*)

Read difuse groundwater recharge from a file and map the value using a conversion function.

Template Parameters

- `ConversionFunction`: Allows the dynamic recalcualtion of recharge base don cell area

Parameters

- `path`: Where to read the file from
- `convertToRate`: The conversion function

std::unordered_map<int, std::array<double, 3>> GlobalFlow::DataProcessing::*GlobalDataReader*`calulateRiverStage`

Helper function for.

See *readBlueCells*

Return a map of bankfull depth, stream width, and lenght

Parameters

- `path`: Where to read the file from

void GlobalFlow::DataProcessing::*GlobalDataReader*`readBlueCells`(std::string *file*, std::unordered_map<int, std::array<double, 3>> *bankfull_depth*)

Reads in river defintions based on a specific elevation data-set.

See *calulateRiverStage*

Parameters

- `file`: to read from
- `bankfull_depth`: A map with addition information

void GlobalFlow::DataProcessing::*GlobalDataReader*`readLakesandWetlands`(std::string *pathGlobalLakes*, std::string *pathGlobalWetlands*, std::string *pathLokalLakes*, std::string *pathLokalWetlands*)

Reads in lakes and wetlands defintions based on Lehner and Döll.

Parameters

- `pathGlobalLakes`:
- `pathGlobalWetlands`:
- `pathLokalLakes`:
- `pathLokalWetlands`:

void GlobalFlow::DataProcessing::*GlobalDataReader*`addEvapo`()

Adds an evapotraspiration module to cells.

void GlobalFlow::DataProcessing::*GlobalDataReader*`addDrainageHack`()

A drainage component similar to de Graaf 2014.

CHAPTER

FIVE

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

5.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

5.2 TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

> a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
>
> b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
>
> c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
>
> d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

> a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
>
> b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
>
> c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
>
> d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
>
> e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
>
> b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
>
> c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
>
> d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
>
> e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
>
> f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE

THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

5.3 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>
>
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

> <program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type `show w`. This is free software, and you are welcome to redistribute it under certain conditions; type `show c` for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

INDEX

G

I

M

N

O

P

R