GLOFRIM Documentation

Release 2.1

J.M. Hoch, D. Eilander, H. Ikeuchi

Jul 17, 2020

CONTENTS

Main development by Jannis M. Hoch (Utrecht University, Deltares), Dirk Eilander (VU Amsterdam, Deltares), Hiroaki Ikeuchi (University of Tokyo), Hessel Winsemius (Deltares) and Fedor Baart (Deltares).

Contact: Jannis M. Hoch (j.m.hoch@uu.nl), Dirk Eilander (dirk.eilander@vu.nl)

We also want to acknowledge the contributions of all colleagues involved in the development of GLOFRIM.

CHAPTER

ONE

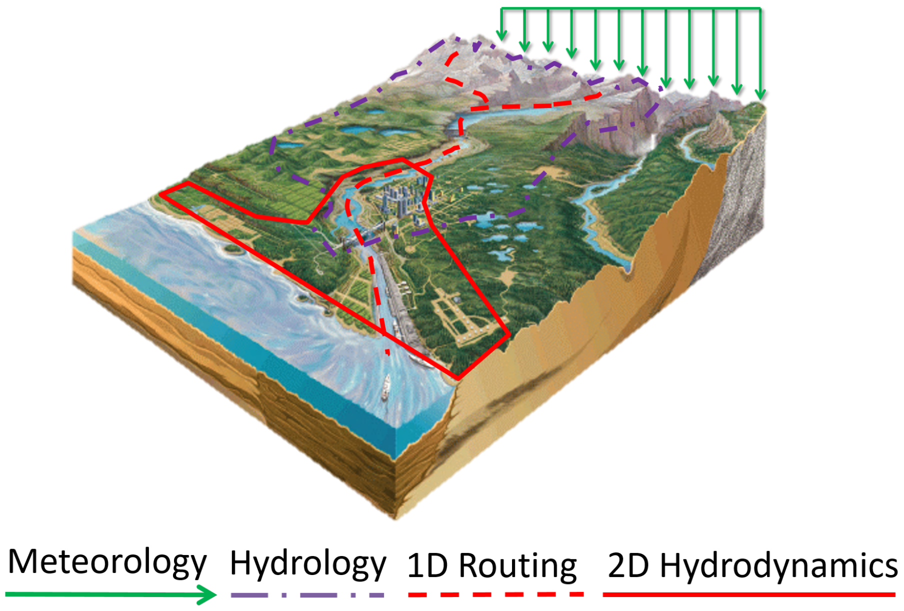
DESCRIPTION

GLOFRIM offers a flexible and modular tool to couple hydrologic, routing, and hydodynamic models across scales. This enables integration of physical processes from different models. The coupling process is spatially explicit (i.e. on grid-to-grid basis) and model information is exchanged online (i.e. per time step).

GLOFRIM is designed as a "human interface" with additional and user friendly Python functions on top of the basic model interface (BMI), which makes it easy to setup and run coupled model simulations. For the model developer, only the BMI needs to be implemented in the model in a scripting language of choice, which makes it easy to develop and maintain.

With the available models, different coupled hydrologic and hydrodynamic model runs can be done, for instance:

- 2-step coupling: hydrology -> 1D routing or hydrology -> full 2D hydrodynamics
- 3-step coupling: hydrology -> 1D routing -> full 2D hydrodynamics



CHAPTER TWO

GLOFRIM DOWNLOAD

The current stable release is GLOFRIM 2.0 and downloadable from Zenodo. Version 2.0 was used in the description artile published in NHESS.

Note: GLOFRIM 2.0 still works with Python 2.7. On GitHub, an unreleased but tested GLOFRIM version working with Python 3.x is available.

GLOFRIM is hosted on GitHub. All unreleased developments can be found there.

All GLOFRIM code is licensed under the GNU General Public License v3.0.

CHAPTER THREE

ACKNOWLEDGMENTS

The development of GLOFRIM was only possible due to the contribution of many colleagues at various departments and universities.

Particular shout-outs go to:

Jeffrey Neal, Paul Bates, Arthur van Dam, Herman Kernkamp, Edwin Sutanudjaja, Rens van Beek, and Marc Bierkens.

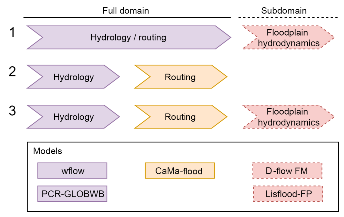
Research was funded by the European Institute of Innovation and Technology (EIT) Climate-KIC and Netherlands Organisation for Scientific Research (NWO).

STRUCTURE

4.1 About

With GLOFRIM it is possible to couple models *spatially explicit* (that is, on a grid-to-grid basis) and *online* (that is, on a timestep basis).

Currently, two different coupling strategies can be realized with GLOFRIM, either 'two-model coupling' or 'three-model coupling'. The former allows to couple hydrology to either routing or hydrodynamic model, while the latter represents the full modelling cascade hydrology to routing to hydrodynamics.



Depending on modelling requirements of a study, the coupling scheme can be configured to your liking. After all, not every physical process and the highest level of detail are required to answer a research question. As such, GLOFRIM can be a flexible tool to avoid under- and overfitting of the modelling question.

Furthermore, the fact that GLOFRIM makes use of the BMI technology allows for extending the current modelling cascade with additional (non-)physical models, ever increasing the representation of relevant processes in a consistent manner.

4.1.1 Supported models

The models currently supported by GLOFRIM are:

Hydrologic models:

- *PCR-GLOBWB*: Global water balance model developed at Utrecht University [Sutanudjaja2018] running at 30 or 05 arc-min spatial resolution; water balance computed based on meteorological forcing, humand and industrial water demand and abstractions; solves kinematic wave approximation for discharge simulations
- *WFLOW*: modelling suite for distributed hydrological models within the Deltares OpenStreams project [WFLOW_docs]; contains various hydrologic models, amongst others SBM and HBV; runs on a regular grid but grid resolution is variable; also solves kinematic wave approximation for discharge simulations

Routing models:

- *CaMa-Flood*: Global routing models designed for fast flood wave propagation simulations; consists of 1D channels plus water balance simulations for floodplains; basin delineation as unit catchments; developed by the University of Tokyo [Yamazaki2011]

Hydrodynamic models:

- *Delft 3D Flexible Mesh*: solving the full shallow water equations; model discretizations can be either flexible meshes, unstructured grids or regular grids; supports 1D, 2D, 1D-2D, and 3D modelling; developed by Deltares [Kernkamp2011]
- *LISFLOOD-FP*: specifically designed for inundation modelling; solving the local inertia equations; allows for 1D, 2D or sub-grid channel discretizations; model grid is always regular; developed by University of Bristol, School of Geographical Studies [Bates2010]

A slighly more elaborated description of the models can be found in the next chapter and in the referenced articles.

4.1.2 Possible applications of GLOFRIM

GLOFRIM is a modular tool allowing for a flexible combinations of different hydrologic and hydrodynamic models. Depending on the design of the coupled model, GLOFRIM can be used for applications such as:

1. *Benchmarking of hydrodynamic models*: By providing identical spatially varying and explicit hydrologic forcing, the performance of hydrodynamic models can be compared;
2. *Nested modelling*: By adding detailed 2D hydrodynamics to large-scale hydrology and routing, inundations can locally be simulated with more accuracy;
3. *Large-scale routing*: For large-scale discharge simulations, replacing kinematic wave approximations with routing models increased accuracy while not increasing run time unneccesarily.

GLOFRIM was so far applied and documented in [Hoch2017a], [Hoch2017b], [Hoch2018], and [Hoch2019].

4.2 Model descriptions

4.2.1 Supported models in GLOFRIM

In the current version, five different models are supported by GLOFRIM: two hydrologic models (PCR-GLOBWB and WFLOW), one routing model (CaMa-Flood), and two hydrodynamic models (Delft3D Flexible Mesh and LISFLOOD-FP).

PCR-GLOBWB

PCR-GLOBWB (hereafter PCR) is a global hydrology and water resources model, fully integrating water use. Sector-specific water demand, groundwater and surface water withdrawal, water consumption, and return flows are dynamically calculated at every time step and interact directly with the simulated hydrology.

The model simulates vertical exchanges between surface, two soil moistere layers, and a groundwater layer. Exchange precipitation is, after accounting for irrigatoin and water use abstractions, routed along a local drainage network using the kinematic wave approximation.

For more information about PCR, we refer to [Sutanudjaja2018].

Model availability

The default and published stable version of PCR is available at Zenodo which is, however, not suitable for application within GLOFRIM.

To apply PCR within GLOFRIM, a separate model version of PCR-GLOBWB needs to be used. It can be found here.

Note: Due to the evolving nature of model code and thus also of PCR-GLOBWB, recently added functionality of PCR-GLOBWB may not be available in the here downloadable version. The overall functionalities of the supported PCR-BMI model are nevertheless equal to those of PCR-GLOBWB 2.0 [Sutanudjaja2018]. If you encounter any issues, please contact the developers!

Note: PCR within GLOFRIM was successfully applied at 30 arc-min spatial resolutions. The 05 arc-min version or schematization at any finer spatial resolution were so far only used in test model and not for detailed studies.

WFLOW

WFLOW (hereafter WFL) is a framework for distributed hydrologic modelling, currently including the following hydrologic models: SBM, HBV, GR4, W3RA and PCRGLOB-WB. In addition, routing can be simulated using the kinematic wave approximation. Depending on the chosen model, the processes described differ and therefore we do not provide a more detailed description of the model routines.

Despite the different models available, WFL ensures that the main model properties and structure (nomenclature, grid properties etc.) are mantained.

For a full documentation of WFL code, models, application, and other functionalities, please visit the WFLOW readthedocs page.

Model availability

WFL contains a BMI adapter by default and therefore no alternative version has to be applied to use it within GLOFRIM.

All code is available at WFLOW GitHub and distributed under the GNU GPL 3.0 license.

Note: For GLOFRIM, only the SBM and W3RA models were tested successfully. Application of other WFL models should be straightforward, but that is uncharted territory at the moment.

CaMa-Flood

CaMa-Flood (hereafter CMF) is designed to simulate the hydrodynamics in continental-scale rivers. It has global extent and routes water along river networks solving the local inertia equation. Water level and flooded area are determined from the water storage at each unit catchment. Since water balance per unit catchment is the only prognostic variable, simulation are efficient.

Since CMF is a global model, only little additional work has to be done to obtain smaller discretizations. Also, this faciliates comparability between test studies since differences in input data are marginal.

For further model description and application, please see [Yamazaki2011] as well as the model's manual.

Note: For the development of GLOFRIM, we applied version 3.6.2. Higher version may differ.

Model availability

The original non bmi'ed model code is available upon request here.

Note: To obtain a bmi'ed version (v.3.6.2) to be applied within GLOFRIM, please contact the developers.

Delft3D Flexible Mesh

Delft3D Flexible Mesh (hereafter DFM) is a fully fledged hydrodynamic model, solving the full shallow water equations. Models can be set up in 1D, 2D, 3D or 1D/2D. For large-scale inundation modelling, a 1D/2D set-up is preferrable and thus also the default design supported by GLOFRIM.

While DFM is used for inundation modelling within GLOFRIM, it also supports other applications, such as salt intrusion or sediment transport simulations. In principle, the coupled simulations by GLOFRIM could be extended with other morphogeohydrologic variables, further increasing the representation of relevant processes.

As the name indicates, DFM allows for discretizing the model domain with a flexible mesh; that is, varying cell geometry and size. By using flexible meshes, the representation of topographical features such as as river bends can be realized with fine spatial resolution, while areas with less dynamic processes require only coarser grids. As a result, the run time needed can be reduced significantly.

Since DFM genercially contains a BMI adapter, any DFM version can be used as long as it satisfies the minimal version requirement.

For documentation of the model scheme, [Kernkamp2011] provides the theoretical and computatonal background. Besides, extensive technical and user manuals are available.

Model availability

The model is free to use, but currently not yet openly available. The DFM development team needs to be contacted for a DFM version. Please see the DFM website for contact information.

Note: DFM version higher than 1.1.201 is required to work with GLOFRIM, the framework has successfully been tested with version 1.1.201.

LISFLOOD-FP

LISFLOOD-FP (hereafter LFP) is a well tested and widely used hydrodynamic model specifically designed to simulate floodplain inundation in a computationally efficient manner over complex topography. It computes water depths in each grid cell at each time step, and hence can simulate the dynamic propagation of flood waves over fluvial, coastal, and estuarine floodplains.

While LFP also allows for 1D and 2D set-ups, only the sub-grid channel design was employed within GLOFRIM due to is improved accuracy.

A major advantage of LFP is its easy model creation which requires, for the simplest set-up, only ascii files describing the DEM, the channel width and bed level elevation, as well as the river bank height. The computational grid is regular in all applications.

The initial paper documenting LFP's computational scheme is [Bates2010]. More model and background information can be found on the LISFLOOD-FP website.

Model availability

The bmi'ed version of LFP (v. 5.9) can freely be downloaded from Zenodo. A test version of the default model can be requested via this form.

Note: The downloadable bmi'ed version is based on LFP version 5.9 and not updated with recent updates. The computational scheme is, nevertheless, identical and inundation simulations are not affected.

Adding new models

It's (relatively) easy to extend GLOFRIM with new models. A requirement is that the model to be added contains BMI functions and follows the conventions used in the python-BMI files of the other models.

4.3 Model coupling

When different models with different scopes are coupled, this may in a range of possible coupling designs. To cater for this flexibility, GLOFRIM allows for various ways of coupling grids and exchanging model information.

To establish online and spatially explicit coupling between models, we employ the Basic Model Interface. An elaborated outline of this concept is provided at *The BMI*.

4.3.1 Spatial coupling of model grids

2D to 2D

For models containing only of 2D model grid, meshes, or unit catchments, for instance PCR and CMF.

By means of the developed grid API, GLOFRIM detects the corresponding cells of different model grids and assigns a 1-n index list which is subsequently used to attribute the right volumes to the right grids.

If n>1, then the volumes from the providing model are devided equally over all coupled cells in the receiving model to maintain a correct water balance between models.

2D to 1D

If coupling a 2D grid/mesh/unit catchment to the channel network of either DFM or LFP.

The workflow is identical as for *2D to 2D*, but before the 1-n index list is created, GLOFRIM finds those indices in the receiving model that belong to river channels. To that end, GLOFRIM employs model-specific properties which are unique to river channels and can therefore be used for the separation.

Note: GLOFRIM aims at using the default input files of the models to perform the spatial coupling of grids and networks. For CMF, however, it is still needed to convert bin/ctl-files to geoTiff-files, particularly the nextxy.bin file for look-up of the flow direction.

We provide a script to convert bin/ctl-files to tif-files:

```
python <path/to>/ cama_maps_io.py <map/folder>/nextxy.ctl.
```

4.3.2 Exchange of model information

With GLOFRIM, model variables can be exchanged for the entire grid/network of a model or only for the most upstream cells/nodes, depending on model set-up and envisaged application.

To define the way model information is exchanged, this has to be specified in the ini-file as explained in *The GLOFRIM configuration file*.

Note: The way model information is exchanged may differ per variable!

Entire domain

If the extent of the models domains of the different models match, information can be exchanged over the entire domain. Alternatively, this option may be chosen in a nested setting if states are exchanged that were not routed in the upstream model.

Only upstream

In case the downstream model is nested into the upstream model (i.e. the spatial extent is smaller), the upstream model may already route discharge until the edge of the downstream model is reached. This may happen if a hydrodynamic model only represents the river delta while the routing and/or hydrologic model capture the entire basin. To account for the processes already simulated upstream, the resulting fluxes/states at the edge must only there be added to the downstream model.

4.4 GLOFRIM requirements and set-up

Note: GLOFRIM is available for Python 3.x on GitHub. The v2.0 release on Zenodo is still based on Python 2.7 though!

To install and run GLOFRIM on your Linux machine, several python packages are required. For convenience, we recommend to set up GLOFRIM within its own python environment. You can do so using conda environments with the provided envrionment.yml.

This should also install the required python BMI-wrapper for you.

```
conda env create -f environment.yml
conda activate glofrim
```

To install GLOFRIM do:

```
git clone git@github.com/openearth/glofrim.git
cd glofrim/py-glofrim
pip install -e <path/to/glofrim>/py-glofrim
```

Once the glofrim environment is successfully set up, the actual models can be installed.

Note: GLOFRIM installation via pip is currently not yet supported.

4.4.1 Note for installing PCR and WFL

Both hydrological models use the PCRaster library within its modelling framework. For a smooth GLOFRIM experience, please ensure PCRaster has been installed before installing PCR or WFL. By default this happens while setting up the environment, but better double-check.

WFL is also directly installed within the environment. PCR, however, must be downloaded and installed separately.

4.4.2 Note for installing CMF, LFP and DFM

All hydrodynamic models are written in other languages than Python, such as Fortran anc C++. These models need to be compiled on your linux machines, check the documentation of the individual models for more info.

The paths to the share libraries of each need to be provided to GLOFRIM. These can be set in the GLOFRIM ini file or provided in a file called environment.env in the glofrim root dir in the *engines* section, see *The GLOFRIM configuration file*.

Note: In addition to the source code, to set up local models and to generate the interpolation matrix for runoff inputs to CMF, additional Fortran scripts need to be compiled. GLOFRIM requires the generate_inpmat.F script to setup the interpolation scheme based on domain of the upstream model in the model cascade. Carefully follow the steps laid out in the README.txt in the glofim/script/generate_inpmat folder for more info.

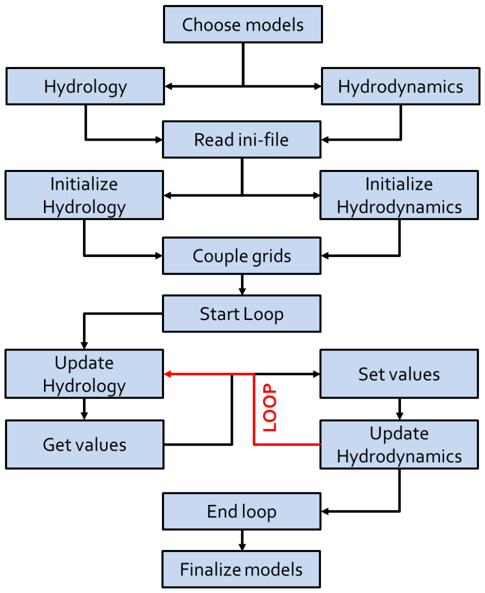
4.4.3 Testing your installation

Once you've followed all steps above: installed GLOFRIM; the models you want to work with and you have set the environment.env file, you can test your setup by running the unit tests provided in glofrim/tests folder.

4.5 Running GLOFRIM

4.5.1 GLOFRIM workflow

By applyting the BMI functions (see :*The BMI*), the following workflow is followed in GLOFRIM. The example shows coupling between a hydrologic and a hydrodynamic model, but would be identical if other or more models were coupled.



After the model are chosen and coupling settings are defined in the GLOFRIM ini-file, the ini-file is read by GLOFRIM to first initialize the model-specific configuration files and then initialize both models as a coupled entity.

Essential part of creating the coupled entity is the coupling of grids which is explained in more detail in *Model coupling*.

Once the coupled models are initialized, a loop is entered, starting at the start time and terminating at the end time as specified in the python executiont command (see *Run GLOFRIM from command line*).

During the loop, models are individually updated from the upstream end of the model cascade to the most downstream model, depending on the number of models coupled.

After a model is updated, the variables to be exchanged (as defined in GLOFRIM ini-file) are retrieved from the providing model, if necessary aligned, and inserted to the receiving model. Only then, model time of the receiving model is forward integrated until the model time of the providing model is reached.

If model end time is reached, model execution is finalized.

4.5.2 Run GLOFRIM from Python

GLOFRIM exists of a series of uniformed BMI wrappers for each model and a overarching BMI wrapper for running coupled models.

Coupled run

To run a coupled model from python use the following lines. The glofrim.ini (see example in root directory) configuration file hold the information of the individual model configuration files and exchanges between the models.

```
# import GLOFRIM
from glofrim import Glofrim
# initialize coupled bmi
cbmi = Glofrim()
# initialize the coupling with the glofrim.ini configuration file
cbmi.initialize_config(/path/to/glofrim.ini)
```

A basic model run uses the following statements:

```
# optional: get the model start time
bmi.get_start_time()
# initialize model
bmi.initialize_model()
# run until set endtime
bmi.update_until(bmi.get_end_time())
# finalize model
bmi.finalize()
```

Stand-alone run

To run stand alone models via the GLOFRIM BMI wrapper you can use the lines below, followed by the same statements as before.

```
# import the CaMa-Flood bmi wrapper
from glofrim import CMF
# intialize bmi with reference to engine (only for non-python models)
bmi = CMF(/path/to/model_engine)
bmi.initialize_config(/path/to/model_configuration_file)
```

4.5.3 Run GLOFRIM from command line

The GLOFRIM library contains a script to run combined and single models with a single line from a terminal. This script can be found in the glofirm-py/scripts folder.

GLOFRIM can be executed as follows on Linux command line:

```
python glofrim_runner.py run /path/to/glofrim.ini --env /path/to/glofrim.env -s␣
↪startdate -e enddate
```

Both *startdate* and *enddate* must be in yyyy-mm-dd format.

For more info on coupled runs, check:

```
python glofrim_runner.py run -help
```

and for stand-alone runs:

```
python glofrim_runner.py run_single -help
```

4.5.4 The GLOFRIM configuration file

The GLOFRIM configuration (or .ini) file has four sections, the engines, models, coupling and exchanges settings, each is shortly explained here.

engines

The engines section contains paths to the shared libraries of each non-python model. For convenience the absolute paths in the engine and models sections may also be set in a seperate environment.env file in the GLOFRIM root folder.:

```
[engines]
# path to model engines; only required for the non-python models used
# these settings can also be set in environment.env
CMF = /path/to/libcama.so
DFM = /path/to/libdflowfm.so
LFP = /path/to/liblisflood.so
```

models

The models section needs the paths to all model configuraiton files. Together with the model engine, this allows GLOFRIM to know the model schematisation and to communicate with that model via BMI. Add only models which are part of the (coupled) run. The paths should be either relative to the root_dir option (if set), this ini file directory or absolute.:

```
[models]
# alternative root dir for relative ini-file paths, by default the directory of this␣
↪ini file is used;
# this setting can also be set in environment.env
root_dir = /path/to/models

# all models which are listed here are run during update
# format: model_short_name = /path/to/configuration_file
PCR=/path/to/pcrglobwb.ini
WFL=/path/to/wflow.ini
CMF=../rel_path/to/input_flood.nam.org
LFP=/path/to/lisflood.par
DFM=rel_path/to/dflowfm.mdu
```

Note: Note that the user can change model options through the GLOFRIM API. For all models but WFL, a new configuration file name ending with *_glofrim* is written to communicate these changes with the model before model initialization. For WFL it's possible to communicate these changes directly via BMI.

Note: CMF only listens to the configuration file if it is called *input_flood.nam*, therefore the original configuration file should be called different, for instance input_flood.nam.org.

coupling

The coupling section contains general settings for the exchanges between models. dt indicates the time step at which information should be exchanged between models. This usually should be at least one full time step of the model that runs with the largest time step. In the example we assume that a WFlow model dictates daily time steps, and that a coupled lisflood model has smaller time steps.

The section furthermore contains projections of the different model instances. GLOFRIM then reprojects the models to enable spatially correct coupling. The projections can be provided in EPSG code (e.g. "EPSG:4326" would indicate regular WGS84 lat lon projection) or as proj string, as shown in the example.:

```
[coupling]
# timestep for exchanges [sec]
dt=86400
WFL=+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
LFP=+proj=utm +zone=34 +south +ellps=WGS84 +datum=WGS84 +units=m +no_defs
```

exchanges

The exchanges section contains the information about how the models communicate on run time. This part has a slightly complex syntax as it contains a lot of information. Every line indicates one exchange from the left (upstream/get) model.variable to the right (downstream/set) model.variable. This can be further extended by multipliers which can be model variables or scalar values in order to make sure the variable units match. Behind the @ the spatial location to get (upstream) and set (downstream) the model variables. Current options are @1d, @1d_us (the most upstream 1d cells or nodes) and @grid_us (the upstream cell for each grid cell). Finally, behind the location of the downstream/set model, a user may set a | sign and then specify the grid cell coordinates (in the projection of the model) in python list form, that should be coupled with the upstream grid cells of the upstream/get model. This should be done as follows:

```
[[x1, y1], [x2, y2], [x3, y3], ...., [xn, yn]]
```

GLOFRIM will then only couple these specific grid cells rather than automatically lookup which cells are coupled. This is an important feature when river networks of the upstream/get and downstream/set models are not entirely commensurate. Examples are provided below:

```
[exchanges]
# setup exchanges which are executed during the coupled update function.
# format: From_model.var1*var2*multiplier@index = To_model.var*multiplier@index
# the multiplier is optional; if no index is set, by default the whole 2D domain is␣
↪coupled

# Example 1: PCR runoff [m] to CMF runoff [m]
# The interal CMF interpolation matrix is used to convert from the PCR grid to the␣
↪CMF U-Grid.
PCR.runoff=CMF.roffin

# Example 2: PCR runoff [m] & upstream discharge [m3/s] to DFM rain [mm] (used as api␣
↪for lateral inflows)
# both sides are converted to volumes per exchange timestep [m3/day]
```

(continues on next page)

(continued from previous page)

```
PCR.runoff*cellArea=DFM.rain*ba*1000@1d
PCR.discharge*86400@grid_us=DFM.rain*ba*1000@1d_us

# Example 3: upstream WFL discharge (RiverRunoff) is fed into a limited set of user␣
↪specified LFP grid cells at the upstream bounds of the model domain.
# The user must ensure that the selected grid cells are overlapping with the intended
# WFL major streams.
WFL.RiverRunoff*86400@grid_us=LFP.SGCQin*86400@1d_us|[[677250, 8346250], [733250,␣
↪8428750], [839750, 8398750], [688750, 8452250], [792750, 8295750]]
```

Note: Note that only fluxes were tested as receiving variables. While states can be used as well, their rather static nature (i.e. using m3 instead of m3/s) can lead to numerical stabilities per time step. Careful testing of the established model coupling is thus necessary!

4.6 The BMI

4.6.1 Introduction

The technical key concept behind GLOFRIM is the Basic Model Interface (BMI).

It was first introduced by [Peckham2013] and constitues the main technical infrastructure for the Community Surface Dynamics Modeling System (CSDMS).

Once a model is bmi'ed, an ensemble of functions is available to retrieve or change information about model properties, variables, time, and many more. See *GLOFRIM API* for a full overview of BMI and GLOFRIM functions.

The most central functions are shown in the following table.

Table 1: *Main functions of the Basic Model Interface.*

bmi.initialize()	initializes the model
bmi.get_value()	retrieves value of a model variable from memory
bmi.set_value()	overwrites value of a model variable on memory
bmi.update()	updates model states a number of timesteps
bmi.finalize()	finalizes model processes

A BMI is language-independent; that is, the programming language of the model is no restriction. For instance, LFP is written in C++, PCR and WFL in Python, and CMF as well DFM in Fortran.

To establish a Python-based coupling framework, however, an additional BMI Wrapper is needed to "translate" the model-specific BMI functions into Python-compatible information.

Why the BMI?

Compared to other model coupling techniques (such as internal or external coupling), using the BMI allows for creating a flexible coupling framework where models can be continiously be developed without affecting the coupling. This is because the BMI is **non-invasive**, i.e. no changes to the code have to be made.

Since models do only interact via the interfaces, unnecessary entanglement of model code is avoided (as it is by internal coupling). Besides, models are executed simultaneously which is the basis for further extension of model coupling (as it is by external coupling) and any dynamic coupling between states/fluxes of different mdoels can be set up flexibly, depending on study purpose.

Also, all 'bmi'ed' models can still be executed in stand-alone mode.

Compared to 'non-bmi'ed' version of a model, run times of the 'bmi'ed' version are near-identical.

4.7 List of References

4.8 GLOFRIM API

4.8.1 Initialization

Before a coupled model can be run, the individual model have to be initialized and all exchanges between them must be defined.

Glofrim.initialize_config	Initializing the model configuration file.
Glofrim.initialize_model	Initializes the model, ie loading all files and checking for consistency.
Glofrim.initialize	Initializes the model following a two-step initialization procedure.
Glofrim.set_exchanges	Defines variable exchanges between models as well as unit and time conversions.

glofrim.Glofrim.initialize_config

Glofrim.**initialize_config**(*config_fn*, *env_fn=None*)

Initializing the model configuration file. aligning GLOFRIM specifications for coupled runs to be consistent with overall run settings; with environment file (env-file) local paths to model engines can be defined

Arguments: config_fn {str} – path to model configuration file

Keyword Arguments: env_fn {str} – path to environment file (default: {None})

Raises: Warning – Warning is raised if two-step model initialization is not correctly executed ValueError – Raised if config-files of models to be coupled are not defined in ini-file ValueError – Raised if engines (ie executables) for models to be coupled are not specified in ini/env-file

glofrim.Glofrim.initialize_model

`Glofrim.`**`initialize_model`**`(`*`**kwargs`*`)`

Initializes the model, ie loading all files and checking for consistency. can only be executed after the model-specific config-file was initialized; essential for a successful two-step model initialization and aligned model data.

Raises: Warning – Warning is raised if two-step model initialization is not correctly executed

glofrim.Glofrim.initialize

`Glofrim.`**`initialize`**`(`*`config_fn`*`)`

Initializes the model following a two-step initialization procedure. first, the config-file is initialized and where necessary aligned with overall model settings (e.g. output-dir, start and end time); second, the model is actually initialized.

Arguments: config_fn {str} – path to model configuration file

glofrim.Glofrim.set_exchanges

`Glofrim.`**`set_exchanges`**`()`

Defines variable exchanges between models as well as unit and time conversions. exchanges can be defined in ini/env-file; it is important that all fluxes/states in ini/env-file are to m3 for mass balance.

Raises: ValueError – Raised if no exchanges are specified in GLOFRIM ini-file ValueError – Raised if an unspported model is specified ValueError – Raised if unknown variables are specified

Returns: list – list describing all models, states, and fluxes that will be exchanged

4.8.2 Execution

GLOFRIM provides a range of functions to retrieve and change time information, update models, as well as exchange content between models during execution. These exchange functions are used inside the update function to exchange variables according to the setting in the exhanges section of the GLOFRIM ini file.

Glofrim.get_current_time	Provides current model time.
Glofrim.update	Updating model for a certain time step interval (default: None).
Glofrim.update_until	Updates the model until time t is reached with an update time step dt.
Glofrim.exchange	Exchanges variable content from specified source variable in source model to a specified destination variable in the destination model.
Glofrim.exchange_same_grid	Exchanges variable content from specified source variable in source model to a specified destination variable in the destination model for the entire grid.
Glofrim.exchange_at_indices	Exchanges variable content from specified source variable in source model to a specified destination variable in the destination model for user-specified indices.

glofrim.Glofrim.get_current_time

`Glofrim.`**`get_current_time`**`()`

Provides current model time.

Returns: date – current model time of PCR-GLOBWB

glofrim.Glofrim.update

`Glofrim.`**`update`**`(`*dt=None*`)`

Updating model for a certain time step interval (default: None). checks whether model end time is already reached; requires that model-specific time step is a whole number of update time step.

Keyword Arguments: dt {int} – update time step interval [sec] at which information is exchanged between models (default: {None})

Raises: Warning – Raised in models are not initialized before updating Exception – Raised if model end time is already reached; no further updating possible

glofrim.Glofrim.update_until

`Glofrim.`**`update_until`**`(`*t*, *dt=None*`)`

Updates the model until time t is reached with an update time step dt.

Arguments: t {int} – Model time until which the model is update

Keyword Arguments: dt {int} – update time step interval [s] at which information is exchanged between models (default: {None})

Raises: Exception – Raised if specified time is smaller than time step or later than model end time

glofrim.Glofrim.exchange

`Glofrim.`**`exchange`**`(`*from_mod*, *to_mod*, *from_vars*, *to_vars*, *coupling*, *add=False*, ***kwargs*`)`

Exchanges variable content from specified source variable in source model to a specified destination variable in the destination model. exchange is possible for entire grid or at specified indices; coupling either as totals or fractional values depending on exchange; source variable can either add to destination variable or overwrite it

Arguments: from_mod {str} – string defining the source model to_mod {str} – string defining the destination model from_vars {str} – string defining the source variable to_vars {str} – string defining the destination variable coupling {SpatialCoupling} – object defining the spatial coupling structure

Keyword Arguments: add {bool} – if True, source values are added to destination values; if False, overwritten (default: {False})

glofrim.Glofrim.exchange_same_grid

`Glofrim.`**`exchange_same_grid`**`(`*from_mod*, *to_mod*, *from_vars*, *to_vars*, *coupling*, *add=False*, ***kwargs*`)`

Exchanges variable content from specified source variable in source model to a specified destination variable in the destination model for the entire grid. source variable can either add to destination variable or overwrite it

Arguments: from_mod {str} – string defining the source model to_mod {str} – string defining the destination model from_vars {str} – string defining the source variable to_vars {str} – string defining the destination variable coupling {SpatialCoupling} – object defining the spatial coupling structure

Keyword Arguments: add {bool} – if True, source values are added to destination values; if False, overwritten (default: {False})

glofrim.Glofrim.exchange_at_indices

`Glofrim.`**`exchange_at_indices`**(*from_mod*, *to_mod*, *from_vars*, *to_vars*, *coupling*, *add=False*, ***kwargs*)

Exchanges variable content from specified source variable in source model to a specified destination variable in the destination model for user-specified indices. coupling either as totals or fractional values depending on exchange; source variable can either add to destination variable or overwrite it

Arguments: from_mod {str} – string defining the source model to_mod {str} – string defining the destination model from_vars {str} – string defining the source variable to_vars {str} – string defining the destination variable coupling {SpatialCoupling} – object defining the spatial coupling structure

Keyword Arguments: add {bool} – if True, source values are added to destination values; if False, overwritten (default: {False})

Variable Getter and Setter Functions

Functions to retrieve and overwrite values for either entire grid or only certain indices.

Glofrim.get_value	Gets values of a certain exposed variable.
Glofrim.get_value_at_indices	Gets values at a specific index of a certain exposed variable.
Glofrim.set_value	Overwriting of all values of a certain exposed variable with provided new values.
Glofrim.set_value_at_indices	Overwriting of value at specific entry of a certain exposed variable with provided new values.

glofrim.Glofrim.get_value

`Glofrim.`**`get_value`**(*long_var_name*, ***kwargs*)

Gets values of a certain exposed variable. entries with fill_value are replaced with NaN values

Arguments: long_var_name {str} – name of exposed model variable

Returns: array – array with values

glofrim.Glofrim.get_value_at_indices

`Glofrim.`**`get_value_at_indices`**(*long_var_name*, *inds*, ***kwargs*)

Gets values at a specific index of a certain exposed variable. entries with fill_value are replaced with NaN values

Arguments: long_var_name {str} – name of exposed model variable inds {int} – Index pointing to entry within entire array of variable values

Returns: float – value at specific index of retrieved variable

glofrim.Glofrim.set_value

`Glofrim.`**`set_value`**`(`*long_var_name*, *src*, ***kwargs*`)`

Overwriting of all values of a certain exposed variable with provided new values. entries with NaN value are replaced with fill_value; provided new values must match shape of aim variable

Arguments: long_var_name {str} – name of exposed model variable src {array} – array with new values

glofrim.Glofrim.set_value_at_indices

`Glofrim.`**`set_value_at_indices`**`(`*long_var_name*, *inds*, *src*, ***kwargs*`)`

Overwriting of value at specific entry of a certain exposed variable with provided new values. entries with NaN value are replaced with fill_value

Arguments: long_var_name {str} – name of exposed model variable inds {int} – Index pointing to entry within entire array of variable values src {array} – array with new values

Returns: [type] – [description]

4.8.3 Finalization

It is possible to change model end times and finalize model states after execution.

`Glofrim.finalize`	Finalizes the model.

glofrim.Glofrim.finalize

`Glofrim.`**`finalize`**`()`

Finalizes the model. shuts down all operations and closes output files.

4.8.4 Auxiliary functions

In addition, there are several auxiliary functions built in GLOFRIM to check states and properties of models as well as their components, variables, and attributs.

Model Information Functions

General information about model structure and variables.

`Glofrim.get_model_type`	Returns type of model.
`Glofrim.get_component_name`	Returns component name of specified model.
`Glofrim.get_input_var_names`	Returns list with all possible input variable names that can be used as exchange TO the specified model.
`Glofrim.get_output_var_names`	Returns list with all possible output variable names that can be used as exchange FROM the specified model.

glofrim.Glofrim.get_model_type

Glofrim.**get_model_type**()

Returns type of model. possible types are hydrologic, routing, or hydrodynamic model

Returns: str – type of specified model

glofrim.Glofrim.get_component_name

Glofrim.**get_component_name**()

Returns component name of specified model.

Returns: str – name of specified component

glofrim.Glofrim.get_input_var_names

Glofrim.**get_input_var_names**()

Returns list with all possible input variable names that can be used as exchange TO the specified model. must be specified in model specific BMI class.

Returns: list – list of all possible input variables

glofrim.Glofrim.get_output_var_names

Glofrim.**get_output_var_names**()

Returns list with all possible output variable names that can be used as exchange FROM the specified model. must be specified in model specific BMI class.

Returns: list – list of all possible output variables

Variable Information Functions

Providing information about properties of model variables.

Glofrim.get_var_type	Returns the type of a user-specified model variable exposed via BMI.
Glofrim.get_var_units	Provides units of variable.
Glofrim.get_var_rank	Provides number of dimensions of variable.
Glofrim.get_var_size	Providestotal number of values contained in variable.
Glofrim.get_var_shape	Provides shape of variable.
Glofrim.get_var_nbytes	Provides number of bytes of variable.
Glofrim.get_time_units	Provides time unit of model.

glofrim.Glofrim.get_var_type

`Glofrim.`**`get_var_type`**`(`*long_var_name*`)`

Returns the type of a user-specified model variable exposed via BMI.

Arguments: long_var_name {str} – long name of exposed model variable

Returns: str – type of specified variable

glofrim.Glofrim.get_var_units

`Glofrim.`**`get_var_units`**`(`*long_var_name*`)`

Provides units of variable.

Arguments: long_var_name {str} – long name of exposed model variable

Returns: str – units of specified variable

glofrim.Glofrim.get_var_rank

`Glofrim.`**`get_var_rank`**`(`*long_var_name*`)`

Provides number of dimensions of variable.

Arguments: long_var_name {str} – long name of exposed model variable

Returns: int – rank of specified variable

glofrim.Glofrim.get_var_size

`Glofrim.`**`get_var_size`**`(`*long_var_name*`)`

Providestotal number of values contained in variable.

Arguments: long_var_name {str} – long name of exposed model variable

Returns: int – size of specified variable

glofrim.Glofrim.get_var_shape

`Glofrim.`**`get_var_shape`**`(`*long_var_name*`)`

Provides shape of variable.

Arguments: long_var_name {str} – long name of exposed model variable

Returns: tuple – shape of specified variable

glofrim.Glofrim.get_var_nbytes

`Glofrim.get_var_nbytes(long_var_name)`

Provides number of bytes of variable.

Arguments: long_var_name {str} – long name of exposed model variable

Returns: int – byte number of specified variable

glofrim.Glofrim.get_time_units

`Glofrim.get_time_units()`

Provides time unit of model.

Returns: str – time unit of model

Grid Information Functions

Information about model grid.

Glofrim.get_grid_type	Provides grid type of model.

glofrim.Glofrim.get_grid_type

`Glofrim.get_grid_type()`

Provides grid type of model. can be regular, flexible or unit catchment grid

Returns: str – Grid type of model

Attribute/ Config Information Functions

Functions providing insights in settings of model configuration files. Note that the set functions can only be used before model initialization.

Glofrim.set_out_dir	Setting output directory of model.
Glofrim.get_time_step	Provides time step of model.
Glofrim.set_start_time	Overwriting default model start time with user-specified start time.
Glofrim.get_start_time	Provides start time of model.
Glofrim.get_end_time	Provides end time of model.
Glofrim.set_end_time	Overwriting default model end time with user-specified end time.
Glofrim.get_attribute_names	Provides list with all model attribute names from model config file.
Glofrim.get_attribute_value	gets attribute value in in underlying model.
Glofrim.set_attribute_value	sets attribute value in underlying model.

glofrim.Glofrim.set_out_dir

`Glofrim.`**`set_out_dir`**(*out_dir*)

Setting output directory of model. overwrites the default output directory

Arguments: out_dir {str} – path to output directory

glofrim.Glofrim.get_time_step

`Glofrim.`**`get_time_step`**()

Provides time step of model.

Returns: date – time step of model

glofrim.Glofrim.set_start_time

`Glofrim.`**`set_start_time`**(*start_time*)

Overwriting default model start time with user-specified start time. format of provided start time must be yyyy-mm-dd

Arguments: start_time {date} – user-specified start time

glofrim.Glofrim.get_start_time

`Glofrim.`**`get_start_time`**()

Provides start time of model.

Returns: date – start time of model

glofrim.Glofrim.get_end_time

`Glofrim.`**`get_end_time`**()

Provides end time of model.

Returns: date – end time of model

glofrim.Glofrim.set_end_time

`Glofrim.`**`set_end_time`**(*end_time*)

Overwriting default model end time with user-specified end time. format of provided end time must be yyyy-mm-dd

Arguments: end_time {date} – user-specified end time

glofrim.Glofrim.get_attribute_names

`Glofrim.`**`get_attribute_names`**`()`

Provides list with all model attribute names from model config file.

Returns: list – list with model attribute names

glofrim.Glofrim.get_attribute_value

`Glofrim.`**`get_attribute_value`**`(`*attribute_name*`)`

gets attribute value in in underlying model. attribute_name should use the following convention model_name.section_name:attribute_name

Arguments: attribute_name {str} – Name of BMI attribute

Returns: float – value of BMI attribute

glofrim.Glofrim.set_attribute_value

`Glofrim.`**`set_attribute_value`**`(`*attribute_name*, *attribute_value*`)`

sets attribute value in underlying model. attribute_name should use the following convention model_name.section_name:attribute_name

Arguments: attribute_name {str} – Name of BMI attribute attribute_value {float} – value to be set

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Hoch2017a] Hoch, J. M., Haag, A. V., van Dam, A., Winsemius, H. C., van Beek, L. P. H., and Bierkens, M. F. P.: Assessing the impact of hydrodynamics on large-scale flood wave propagation – a case study for the Amazon Basin, Hydrol. Earth Syst. Sci., 21, 117-132, https://doi.org/10.5194/hess-21-117-2017, 2017.

[Hoch2017b] Hoch, J. M., Neal, J. C., Baart, F., van Beek, R., Winsemius, H. C., Bates, P. D., and Bierkens, M. F. P.: GLOFRIM v1.0 – A globally applicable computational framework for integrated hydrological–hydrodynamic modelling, Geosci. Model Dev., 10, 3913-3929, https://doi.org/10.5194/gmd-10-3913-2017, 2017.

[Hoch2018] Hoch, J.M., van Beek, R., Winsemius, H.C., and Bierkens, M.F.P.:Benchmarking flexible meshes and regular grids for large-scale fluvial inundation modelling,Adv. Water Resour., 121, 350-360, https://doi.org/10.1016/j.advwatres.2018.09.003, 2018.

[Hoch2019] Hoch, J.M., Eilander, D., Ikeuchi, H., Baart, F., and Winsemius, H.C.:Evaluating the impact of model complexity on flood wave propagation and inundation extent with a hydrologic–hydrodynamic model coupling framework,Nat. Hazards Earth Syst. Sci., 19, 1723–1735, https://doi.org/10.5194/nhess-19-1723-2019, 2019

[Sutanudjaja2018] Sutanudjaja, E. H., van Beek, R., Wanders, N., Wada, Y., Bosmans, J. H. C., Drost, N., van der Ent, R. J., de Graaf, I. E. M., Hoch, J. M., de Jong, K., Karssenberg, D., López López, P., Peßenteiner, S., Schmitz, O., Straatsma, M. W., Vannametee, E., Wisser, D., and Bierkens, M. F. P.: PCR-GLOBWB 2: a 5arcmin global hydrological and water resources model, Geosci. Model Dev., 11, 2429-2453, https://doi.org/10.5194/gmd-11-2429-2018, 2018.

[Bates2010] Bates, P. D., Horritt, M. S., and Fewtrell, T. J.: A simple inertial formulation of the shallow water equations for efficient twodimensional flood inundation modelling, J. Hydrol., 387, 33–45, https://doi.org/10.1016/j.jhydrol.2010.03.027, 2010.

[Kernkamp2011] Kernkamp, H. W. J., van Dam, A., Stelling, G. S., and de Goede, E. D.: Efficient scheme for the shallow water equations on unstructured grids with application to the Continental Shelf, Ocean Dynam., 61, 1175–1188, https://10.1007/s10236-011-0423-6, 2011.

[Yamazaki2011] Yamazaki, D., Kanae, S., Kim, H. and Oki, T.: A physically based description of floodplain inundation dynamics in a global river routing model, Water Resour. Res., 47, 1–21, https://10.1029/2010WR009726, 2011.

[WFLOW_docs] Deltares: OpenStreams WFLOW documentation, https://wflow.readthedocs.io/en/latest/

[Peckham2013] Peckham, S. D., Hutton, E. W. H., and Norris, B.: A component-based approach to integrated modeling in the geosciences: The design of CSDMS, Comput. Geosci., 53, 3–12, https://doi.org/10.1016/j.cageo.2012.04.002, 2013.

PYTHON MODULE INDEX

INDEX