

Simulation software for safe, sustainable and future deltas

DELFT3D FMI SUITE

Deltares systems

D-Flow Flexible Mesh

D-Flow Flexible Mesh

Developer's Guide

Version: 1.2.0
Revision: 78359

25 April 2024

D-Flow Flexible Mesh, Developer's Guide

Published and printed by:

Deltares
Boussinesqweg 1
2629 HV Delft
P.O. 177
2600 MH Delft
The Netherlands

telephone: +31 88 335 82 73

e-mail: [Information](#)

www: [Deltares](#)

For sales contact:

telephone: +31 88 335 81 88
e-mail: [Sales](#)
www: [Sales & Support](#)

For support contact:

telephone: +31 88 335 81 00
e-mail: [Support](#)
www: [Sales & Support](#)

Copyright © 2024 Deltares

All rights reserved. No part of this document may be reproduced in any form by print, photo print, photo copy, microfilm or any other means, without written permission from the publisher: Deltares.

Contents

List of Tables	v
List of Figures	vii
List of To Do's	ix
1 A guide to this developer's guide	1
1.1 Introduction	1
1.2 Document version and revisions	1
1.3 Typographical conventions	1
1.4 Changes with respect to previous versions	1
2 Source tree architecture of D-Flow FM	3
2.1 Introduction	3
2.2 Source builds on Windows: Intel, MS Visual Studio and MSBuild	3
2.2.1 Solution file, templates and project files	3
2.3 Source builds on Linux: Intel or GNU, automake and autoconf	3
3 The Basic Model Interface (BMI) for D-Flow FM	5
3.1 Introduction	5
3.2 Basic use of the API	5
3.3 Full API description	5
3.4 List of available model variables via BMI	5
3.5 Adding new model variables to the BMI interfaces	8
4 Parallelization	9
4.1 Introduction	9
4.2 Code dependencies on MPI	9
4.3 Partition mesh and MDU files in the command line	9
4.4 Parallel runs and debugging on Windows	10
4.5 Some related development	11
5 Deployment of D-Flow FM	13
5.1 Introduction	13
5.2 TeamCity specifics	13
5.3 Deployment on Windows platforms	13
5.4 Deployment on Linux platforms	13
5.4.1 Source distribution for Linux	13
5.4.2 Subversion distribution for Linux	13
5.4.3 Binary distribution for Linux	13
5.4.3.1 Known possible issues with binary distributions	14
6 Miscellaneous stuff	15
6.1 Debugging dflowfm.dll running under Delft3D Flexible Mesh Suite or D-HYDRO Suite	15
7 Test bench	17
7.1 Introduction	17
7.2 Location of all parts of the test bench	17
7.3 How to add a new test case	17
Index	21

DRAFT

List of Tables

3.1	List of available variables via D-Flow FM's BMI API.	5
-----	--	---

DRAFT



DRAFT

List of Figures

DRAFT



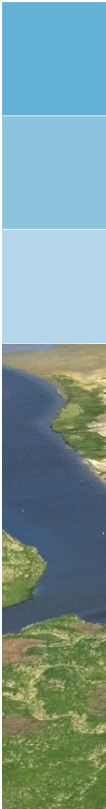
DRAFT

List of To Do's

5.1 sort out Ubuntu issues 13

5.2 include actual error here 14

DRAFT



DRAFT

1 A guide to this developer's guide

1.1 Introduction

This Developer's Guide concerns the hydrodynamic module, D-Flow FM, of the Delft3D Flexible Mesh software suite. This guide is intended for people working with the source code, for example developers, or users of D-Flow FM's API.

The recommended way of using this guide is either via the index on page ??, or via the search function of your viewer.

1.2 Document version and revisions

[yet empty]

1.3 Typographical conventions

[yet empty]

1.4 Changes with respect to previous versions

[yet empty]



DRAFT

2 Source tree architecture of D-Flow FM

2.1 Introduction

This chapter contains a description of the source tree architecture of D-Flow FM, and the development/build environments that can be used on Windows and Linux.

2.2 Source builds on Windows: Intel, MS Visual Studio and MSBuild

Recommended tools:

- ◇ Intel Fortran Composer XE
- ◇ Microsoft Visual Studio (with MS Visual C++)

Regular development and manual builds are done within Visual Studio. Batch-mode build on Windows is done via an MSBuild file `<dflowfm.proj>`. Both depend on the solution file `<dflowfm.sln>` discussed in the next section.

2.2.1 Solution file, templates and project files

The list below summarizes the relevant build configuration files, and which settings are where.

`dflowfm.sln`

When making changes to this file, make sure to copy them also into the leading template file `scripts/template/dflowfm_template.sln`

`scripts/template/dflowfm_template.sln`

The leading solution template file, that is used to produce the actual solution file `rootdir/dflowfm.sln`

`src/*.f90`

The majority of all D-Flow FM source files.

`src/dflowfm_kernel/dflowfm_kernel.vfproj`

The Visual Fortran project file for the kernel as a static library.

The solution file `dflowfm.sln` is based on a template, such that it can be generated for various combinations of versions of Visual Studio and the Intel Fortran compiler. To generate the actual solution file, in the root dir run:

```
python prepare_sln.py
```

2.3 Source builds on Linux: Intel or GNU, automake and autoconf

[yet empty]

See <http://publicwiki.deltares.nl/display/DFLOWFM/Building+on+Linux>

DRAFT

3 The Basic Model Interface (BMI) for D-Flow FM

3.1 Introduction

The *Basic Model Interface (BMI)* is a set of standardized subroutine interfaces that a simulation engine/model may use to implement its API. The BMI implementation in D-Flow FM enables easy access to and interaction with a running D-Flow FM model schematisation from various host languages, for example Python and C#.

3.2 Basic use of the API

[yet empty]

3.3 Full API description

[yet empty]

3.4 List of available model variables via BMI

Table 3.1: List of available variables via D-Flow FM's BMI API.

Variable name	Unit	Description	Shape
DFM_COMM_DFMWORLD	—	The MPI communicator for dflowfm (FORTRAN handle)..	(/ 0 /)
iglobal_s	—	global flow node numbers to help output aggregation later. Should exactly correspond with the original unpartitioned flow node numbers! (as opposed to iglobal).	(/ ndx /)
Uorb	m/s	orbital velocity.	(/ ndx /)
twav	s	wave period.	(/ ndx /)
shx	m	current position.	(/ nshiptxy /)
shy	m	current position.	(/ nshiptxy /)
shi	m	current position.	(/ nshiptxy /)
zsp	m	ship depth at flownodes.	(/ nshiptxy /)
shL	m	ship size L/2, B/2, D ! for now, fixed max nr =2.	(/ 2 /)
shB	m	ship size L/2, B/2, D ! for now, fixed max nr =2.	(/ 2 /)
shd	m	ship size L/2, B/2, D ! for now, fixed max nr =2.	(/ 2 /)
stuw	N	actual thrust force in ship dir.	(/ 2 /)
fstuw	—	thrust setting 0-1.	(/ 2 /)
continued on next page			

Table 3.1 – continued from previous page

Variable name	Unit	Description	Shape
stuwmx	N	max thrust.	(/ 2 /)
roer	<i>degree</i>	actual rudder angle.	(/ 2 /)
froer	<i>degree</i>	actual rudder setting 0-1.	(/ 2 /)
roermx	<i>degree</i>	max rudder angle.	(/ 2 /)
wx	m/s	wind x velocity (m/s) at u point.	(/ 1nx /)
wy	m/s	wind y velocity (m/s) at u point.	(/ 1nx /)
s0	m	waterlevel (m) at start of timestep.	(/ ndx /)
s1	m	waterlevel (m) at end of timestep.	(/ ndx /)
a0	m^2	storage area at start of timestep.	(/ ndx /)
a1	m^2	storage area at end of timestep.	(/ ndx /)
vol0	m^3	volume at start of timestep.	(/ ndx /)
vol1	m^3	volume at end of timestep.	(/ ndx /)
hs	m	waterdepth at cell centre = s1 - bl (m).	(/ ndx /)
ucx	m/s	cell center velocity, global x-dir (m/s).	(/ ndkx /)
ucy	m/s	cell center velocity, global y-dir (m/s).	(/ ndkx /)
ucz	m/s	cell center velocity, global z-dir (m/s).	(/ ndkx /)
sa0	$1e-3$	salinity (ppt) at start of timestep.	(/ ndkx /)
sa1	$1e-3$	salinity (ppt) at end of timestep.	(/ ndkx /)
satop	$1e-3$	salinity (ppt) help in initialise , deallocated.	(/ ndx /)
tem0	<i>degC</i>	water temperature at end of timestep.	(/ ndkx /)
tem1	<i>degC</i>	water temperature at end of timestep.	(/ ndkx /)
u1	m/s	flow velocity (m/s) at end of timestep.	(/ 1nkx /)
frcu	<i>todo</i>	friction coefficient set by initial fields.	(/ 1nx /)
viusp	m^2/s	user defined spatial eddy viscosity coefficient at u point (m2/s).	(/ 1nx /)
diusp	m^2/s	user defined spatial eddy diffusivity coefficient at u point (m2/s).	(/ 1nx /)
kfs	—	node code flooding.	(/ ndx /)
kfst0	—	node code flooding.	(/ ndx /)
ba	m^2	bottom area, if < 0 use table in node type.	(/ ndx /)
continued on next page			

Table 3.1 – continued from previous page

Variable name	Unit	Description	Shape
bl	m	bottom level (m) (positive upward).	(/ ndx /)
ln	—	link (2,*) node administration, 1=nd1, 2=nd2 linker en rechter celnr.	(/ 2, lnkx /)
lncn	—	link (2,*) corner administration, 1=nod1, 2=nod2 linker en rechter netnr.	(/ 2, lnkx /)
iadv	—	type of advection for this link.	(/ lnx /)
bob	m	left and right inside lowerside tube (binnenkant onderkant buis) HEIGHT values (m) (positive upward).	(/ 2, lnx /)
vort	$s - 1$	vorticity at netnodes.	(/ ndx /)
xzw	m	centre of gravity.	(/ nump /)
yzw	m	centre of gravity.	(/ nump /)
xk	—	Net node x coordinate.	(/ numk /)
yk	—	Net node y coordinate.	(/ numk /)
zk	—	Net node z coordinate.	(/ numk /)
kn	—	Net links: kn(1,:)=from-idx, kn(2,:)=to-idx, kn(3,:)=net link type (0/1/2).	(/ 3, numl /)
zbnd1d2d1	m	1d2d boundary points 1d water level at new time level.	(/ nbnd1d2d /)
zbnd1d2d0	m	1d2d boundary points 1d water level at previous time level.	(/ nbnd1d2d /)
zcrest1d2d	m	1d2d helper array with crest levels.	(/ nbnd1d2d /)
edgenumbers1d2d	m	1d2d helper array with edge numbers.	(/ nbnd1d2d /)
kbnd1d2d	—	1d2d boundary points index array.	(/ 5, nbnd1d2d /)
width_1d	m	width 1D SOBEK channel –2D FM coupling.	(/ nbnd1d2d /)
qzeta_1d2d	m^3s^{-1}	1d2d output array via BMI for qzeta in 1D SOBEK–2D FM coupling.	(/ nbnd1d2d /)
qlat_1d2d	m^3s^{-1}	1d2d output array via BMI for qlat in 1D SOBEK–2D FM coupling.	(/ nbnd1d2d /)
qtotal_1d2d	m^3s^{-1}	1d2d output array via BMI for qlat in 1D SOBEK–2D FM coupling.	(/ nbnd1d2d /)

3.5 Adding new model variables to the BMI interfaces

New model state variables can be made available to the outside world by adding them to the BMI subroutines. The various BMI-subroutines can be automatically generated using a Python-script and this is strongly advised. Take the following steps:

- 1 **Make the variable available.** Make sure your module variable is `public` (possibly implicitly), and pointerable, by adding the `target` attribute (see below).
- 2 **Add self-describing documentation to the variable.** Behind the variable declaration in your module, add a correct documentation string in the following format:

```
double precision, allocatable, target :: sl(:) !< [m] waterlevel at end of
    timestep {"shape": ["ndx"]}
```

(Don't use newlines in the documentation string.)

The syntax is:

```
<type>, target ::<var>(:,...) !< [<unit>] <some description> {"shape": [<isize>]%, .
```

- 3 **Special case for derived type fields** Alternatively, when wanting to expose a particular member field of a user defined type variable as a regular BMI variable, the special prefix `!$BMIEXPORT` is available, in combination with the attribute `"internal"` in the JSON string.

After the actual variable, put one or more comment lines, each one for a single member field that you want to expose. For example:

```
type(stmtype), target :: stmpar ←
    !< All relevant parameters for sediment-transport-morphology module.

!$BMIEXPORT double precision :: bodsed(:, :) ←
    !< [kg m-2] Available sediment in the bed in flow cell center. ←
    {"location": "face", "shape": ["stmpar%morlyr%settings%nfrac", "ndx"], ←
    "internal": "stmpar%morlyr%state%bodsed"}
!$BMIEXPORT double precision :: dpsed(:) ←
    !< [m] Sediment thickness in the bed in flow cell center. ←
    {"location": "face", "shape": ["ndx"], ←
    "internal": "stmpar%morlyr%state%dpsed"}
```

(Don't use newlines in the documentation string, the `←`s are for readability only.)

The specified internal variable expression will be exposed in the BMI under the specified name (for example, `get_var("bodsed")` will return `stmpar%morlyr%state%bodsed`).

- 4 **Run the BMI-generator script.** Open a DOS-box located at your source scripts dir and run `generate.cmd`:

```
$ cd D:\your_dfm_sourcecode\scripts
$ generate.cmd
```

Note that the `generate.cmd` file contains a short list of FORTRAN files, only those will be scanned for BMI-comments. Add your file if it is not in the list yet.

- 5 **In case of new modules.** In addition to the automatically generated BMI-interface code parts, some parts require manual editing. Open `unstruc_bmi.F90` and find all subroutines that contain a statement similar to:

```
include "bmi_set/get_var(_shape/_rank/_name/_role/_type).inc"
```

At the top of subroutines, verify that you module is being made available via a `use` statement.

4 Parallelization

4.1 Introduction

D-Flow FM can run parallel calculations and uses the Message Passing Interface standard (MPI) for that.

4.2 Code dependencies on MPI

MPI is now a requirement for D-Flow FM, not directly, but via an implicit MPI-dependency required by `deltares_common`. Still, the D-Flow FM code contains preprocessing directives to enable the MPI-specific code:

```
#ifdef HAVE_MPI
    use mpi
#endif
```

4.3 Partition mesh and MDU files in the command line

A partitioned model can be run interactively on Windows, for the purpose of debugging with Visual Studio. Running a model as a parallel calculation requires preparing and partitioning your model input files. Both the mesh and the MDU files need to be partitioned. See [Deltares \(2024\)](#) for more details on how to implement the partitioning.

There are two types of partitioning the mesh: METIS and manually partition with a user-specified polygon. The command that uses METIS partitioner reads:

```
> dflowfm-cli --partition:ndomains=n <meshfile>
```

where `ndomains=n` specifies that `n` subdomains are to be generated. This command results in subdomain mesh files `<example_000j_net.nc>`, `j=0, 1, ..., n-1`. An advanced command, which enables more options, is:

```
> dflowfm-cli --partition:ndomains=n[:method=0|1][:genpolygon=0|1][:contiguous=0|1] <meshfile>
```

where the partition method can be chosen via setting `method=0` the Recursive Bisection approach (default), and `method=1` the K-Way approach. We refer to [Karypis \(2013\)](#) for more details about these two approaches.

Option `genpolygon` is used to specify whether or not a partition polygon is generated. Such polygon file stands for the boundaries of subdomains, and can be used to generate subdomain cell coloring in the initialization stage of a parallel simulation.¹ By default, no such polygon file

¹ METIS results in subdomain cell coloring information, i.e. cells that are in the same subdomain have the same color. This information is important in the initialization stage of a parallel run, where the old D-Flow FM computes this information using a partition polygon which is generated by the partition command. This method could not be used for the 1D network where it is difficult to define such a polygon. The new development (default setting) is that no such polygon is generated or used anymore. The cell coloring information is written to partition mesh files, and then read for the parallel simulation. In this way, both 1D and 2D networks can be simulated.



is generated (`genpolygon=0`), and the subdomain cell coloring information are written into the partition mesh files. When `genpolygon=1`, a polygon file will be generated and the cell coloring won't be written to the resulting mesh files.

Moreover, option `contiguous` enforces the contiguous partition when specifying both `contiguous=1` and `method=1`. (Only the K-Way method enables the contiguous partition.) It is not switched on by default. Comparing to the previous command, this advanced command additionally generates a partition polygon file `<example_part.pol>` when `genpolygon=1` is specified.

To manually partition a mesh, a user-specified polygon file `<userpols.pol>` has to be provided. The corresponding command reads:

```
> dflowfm --partition <meshfile> <userpol.pol>
```

This generates files the same as before.

There is an efficient way to partition both the mesh and MDU files, by:

```
> dflowfm-cli --partition:ndomains=n[:method=0|1][:genpolygon=0|1][:contiguous=0|1]
[:icgsolver=i] <mdu-file>
```

This command reads the name of the mesh file from `mdu-file`, and generates `n` subdomain mesh files. Accordingly, it creates `n` subdomain MDU files where the `icgsolver` is set to `i`. If the user specifies `genpolygon=1`, then additionally a partition polygon file `example_part.pol` is generated, and the resulting MDU files contains `PartitionFile = example_part`

4.4 Parallel runs and debugging on Windows

Once the model input is complete, a parallel debugging session is started as follows:

- 1 **Start the MPI daemon.** This is a one-time operation. Open a DOS-box, make sure the Intel runtime programs are available and start the SMPD program in debug mode:

```
$ call "%IFORT14_COMPILER%\bin\ipsxe-comp-vars.bat" intel64 vs2012
$ smpd.exe -d
```

(Replace `IFORT14` in case you've got a different Intel Fortran version.)

- 2 **Start the parallel model run.** In another DOS-box, where again the Intel runtime programs are available, use the `mpiexec` command to start the parallel run. Do this from inside the model directory where the MDU-files are located. Specify the full path to the `dflowfm.exe` program which you will be debugging:

```
$ mpiexec.exe -localonly -np 3 D:\your_dfm_sourcecode\bin\x64\Debug\dflowfm.exe
--autostartstop --pressakey yourmodel.mdu
```

(Here `localonly` specify that you are going to run on a local machine and `-np 3` specifies the number of processes, specific for your simulation.) Notice the use of the `--pressakey` option. This will put the parallel processes on hold so that you first have the opportunity to attach the debugger in the following step. Also note that you may add the `--nodisplay` option if you don't need the Interacter GUI during debugging.

- 3 **Attach the Visual Studio debugger to the parallel processes.** Make sure you are in the Debug configuration (and x64 platform in the above command, but Win32 is also possible). Click the menu item *DEBUG > Attach to Process...* In the attach-dialog look for the `dflowfm.exe` processes and select all of them that you have just launched in the previous step (use Ctrl-key + mouse click to select multiple processes).
- 4 **Start the actual debugging.** Prepare for debugging as you would do normally, e.g., placing breakpoints or watches. Start the actual run but returning to the mpiexec-DOS box and pressing the Enter-key once. All processes will then start running. Return to Visual Studio and start debugging.

4.5 Some related development

- ◇ When partition a mesh, the cell information (i.e. `netcell%nod` and `netcell%lin` in the code) of each subdomain is written to the subdomain mesh files, so that in the parallel run, the expensive step of finding cell information is skipped (i.e. skip subroutines `findcells` and `find1dcells`). Moreover, if there is no cell information in the mesh file, the step of finding cells is automatically switched on. This also works in sequential run.
- ◇ In the command line, we can save net file with cell information, by option `convertnetcells`.

DRAFT

5 Deployment of D-Flow FM

5.1 Introduction

This chapter contains descriptions of the various ways how D-Flow FM can be installed on various platforms, and how the source and binary distributions can be built.

5.2 TeamCity specifics

[yet empty]

5.3 Deployment on Windows platforms

[yet empty]

5.4 Deployment on Linux platforms

5.4.1 Source distribution for Linux

D-Flow FM can be shipped as a source distribution `dflowfm-1.1.xxx.tar.gz`. The target audience for this format is users that are no developers (hence, no SVN-access), but who want to build on their own specific system with specific libraries or compilers. The goal is that they do not need any autotools packages per se (because `configure` and all `Makefile.ins` are in the `.tar.gz`. This works well on CentOS, but on Ubuntu sometimes Automake and/or Libtool is still required on the user's machine.

TODO 5.1: sort out Ubuntu issues

TODO

5.4.2 Subversion distribution for Linux

When building from an SVN working copy, run `./autogen.sh` once. Next, follow the normal source build steps. Make sure recent enough versions of automake, autoconf and libtool are in your path (e.g., run `module load automake`, etc.)

5.4.3 Binary distribution for Linux

Binary distributions including third-party dependencies has its drawbacks, and RPM packages may be preferred, but sometimes it is necessary. D-Flow FM is then shipped with its `lib/` directory filled with all known dependency libraries (`.sos`).

Dependencies are listed via:

```
ldd dflowfm
```

5.4.3.1 Known possible issues with binary distributions

1 **Issue:** error 'cannot open shared object file'.

```
dfLOWfm: error while loading shared libraries: libXXX.so: cannot open
shared object file: No such file or directory
```

Answer

Not all required libraries can be found, make sure the `.so` is in the `LD_LIBRARY_PATH` (either `export` it, or use `module load`).

2 **Issue:** opal help file error.

```
couldn't open the help file:
/opt/openmpi/1.8.1_intel_14.0.3/share/openmpi/help-opal-runtime.txt:
No such file or directory. Sorry!
```

Answer

This error may occur when D-Flow FM call `mpi_init`. OpenMPI has the problem that the binary libraries contain a hard path to its help files, e.g., as it was built on our systems. No doubt, the user uses different locations, so we should not ship OpenMPI ourselves.

Solution: advise user RPM/debpkg/self-built OpenMPI. Or fallback: OpenMPI offers environment variable `OPAL_PREFIX`, we could set it to `dfLOWfm-cli-location/dfLOWfm/somedir`, but then we would need to ship the OpenMPI help files. Undesirable.

3 **Issue:** double `-l -l` error

TODO *TODO 5.2: include actual error here*

Answer

Caused by a combination of the MPI compiler and some version of `libtool`. A call to `mpif90 -gen-dep` produces whitespace between `-l` and the libname, which `libtool` parses incorrectly. More info on <https://issuetracker.deltares.nl/browse/UNST-732>

6 Miscellaneous stuff

6.1 Debugging dflowfm.dll running under Delft3D Flexible Mesh Suite or D-HYDRO Suite

In many cases it is desirable to be able to debug dflowfm.dll from the Visual Studio IDE, as it is running under Delft3D Flexible Mesh Suite or D-HYDRO Suite. There are cases in which the dflowfm.dll behaves differently from dflowfm.exe or dflowfm-cli.exe and cases in which one would like to see how Delft3D Flexible Mesh Suite or D-HYDRO Suite is calling API functions. The procedure is straightforward:

- ◇ In the Visual Studio solution, set dflowfm_dll_2012 as the startup project.
- ◇ Modify the properties of dflowfm_dll_2012 as follows:
 - General -> Output Directory :
`%DSPATH%\plugins\DeltaShell.Plugins.FMSuite.FlowFM\dflowfm_kernel\x64`
 - Debugging -> Command :
`%DSPATH%\bin\DeltaShell.Gui.exe`
 - Debugging -> Working Directory :
`%DSPATH%\bin`
- ◇ Shift-F5 Go !!
- ◇ Due to the project's new settings, dflowfm.dll and its .pdb will be rebuilt in the path where the Delft3D Flexible Mesh Suite or D-HYDRO Suite expects them.
- ◇ Ignore Visual Studio's complaints about missing symbols by pressing 'YES'
- ◇ Be patient while Visual Studio attempts to load all kinds of irrelevant symbols....
- ◇ Delft3D Flexible Mesh Suite or D-HYDRO Suite will fire up.
- ◇ Either load an existent Delft3D Flexible Mesh Suite or D-HYDRO Suite project or create a new one while importing a dflowfm model.
- ◇ **Set 'Use RPC' to FALSE** for the dflowfm model (right-clicking in the project tree left on the screen on the dflowfm model and select 'Properties', opens the properties pane on the right.)
Otherwise, dflowfm is run as a remote instance, which is inaccessible to Visual Studio for debugging.
- ◇ Set breakpoints in Visual Studio
- ◇ Run the model in the Delft3D Flexible Mesh Suite or D-HYDRO Suite, which will hit the breakpoints



DRAFT

7 Test bench

7.1 Introduction

The test bench of DFLOW-FM is an important tool to keep all parts working as expected, while developing new features, fixing bugs or reorganizing the code.

The first section describes where to find the scripts of the test bench, the test cases, the location of the website to monitor the results. The second section describes how to add a new test case.

The test bench runs automatically: if something changes in the source code of DFLOW-FM, or in the test scripts or test cases, new executables are built, and tests are run, and the test results are compared with reference results.

The test scripts are also used for other projects than D-Flow FM (e.g. Delft3D-FLOW and D-Water Quality), and works on Windows and Linux.

7.2 Location of all parts of the test bench

TeamCity (see <https://www.jetbrains.com/teamcity/>) is used as the test bench environment. The overall project page for DFlowFM with TeamCity is located at <https://build.deltares.nl/project.html?projectId=DFlowFlexibleMesh&tab=projectOverview>. On this page you will find three subprojects: 3Di-related, Build FM distributions and Testbenches FM.

The svn repository of the test bench scripts is:
<https://repos.deltares.nl/repos/DSCTestbench/scripts/trunk>.

The test bench itself is a python project with as the main file: `TestBench.py`. As a default within Deltares, we use Anaconda (<https://anaconda.org/>) as the python environment.

The test cases itself are located on two locations: the input of the cases are at: https://repos.deltares.nl/repos/DSCTestbench/cases/trunk/e02_dflowfm, the output to which we compare our results (references) are at: <https://repos.deltares.nl/repos/DSCTestbench/references/trunk>.

7.3 How to add a new test case

We assume that the test case input is already in the right location in the repository, if not, add the test case to it.

To add test cases, you should first choose the configuration where your test case belongs to. There are currently about 60 configurations. The most important configurations for DFLOW-FM are:

- ◇ `dflowfm_win64.xml`
- ◇ `dflowfm_lnx64_single.xml`
- ◇ `dflowfm_lnx64_parallel.xml`
- ◇ `dflowfm_3dcases_win64.xml`

As the extension suggests, the configuration is an xml file. The xml file is defined by its xsd: https://repos.deltares.nl/repos/DSCTestbench/scripts/configs/deltaresTestbench_v3-2.00.xsd



Each test case is defined within the tags `<testCase>`. An example is:

```
<testCase name="e02_f14_c060_sill_free_griddir" ref="dflowfm_default">
  <path>e02_dflowfm/f14_parallel/c060_sill_free_griddirection</path>
  <programs>
    <program ref="DFlowFM">
      <arguments>
        <argument>weirfree.mdu</argument>
        <argument>--autostartstop</argument>
        <argument>--nodisplay</argument>
      </arguments>
    </program>
  </programs>
  <maxRunTime>15000.0</maxRunTime>
  <checks>
    <file name="dflowfmoutput/pillar_small_0000_his.nc" type="netCDF">
      <parameters>
        <parameter name="waterlevel" toleranceAbsolute="0.00001" />
        <parameter name="x_velocity" toleranceAbsolute="0.00001" />
        <parameter name="y_velocity" toleranceAbsolute="0.00001" />
      </parameters>
    </file>
  </checks>
</testCase>
```

Most settings can be copied from this example. Next settings must be altered:

tag	description
name	identification for test case
path	relative path to input of test case
argument	name of the mdu-file
file name	path to output path
type	type of output file
parameter	variable to compare with selected tolerance (absolute or relative)

Once the test case is included in the configuration file, the test script can be used to run this case to produce references.

```
python --config dflowfm_lnx64_parallel.xml --filter f14_c060 -r
```

The option `-r` is for reference run.

If everything goes right, the reference value can be found in the directory:
`data/references/trunk/<platform>/<testset1>/<testset2>/<name> .`

If everything looks fine (which can be checked, locally, to run the testscript with the `-c` option, for comparison), these references can be submitted to the repository.

Finally check in the changes to the configuration file.

Repeat the steps for the other configurations.

Deltares, 2016. "BIBTEX key with no entry, needed if no citations are made in the document."

Deltares, 2024. *D-Flow FM Hydro- and Morphodynamics User Manual*. Deltares, 1.1.124 ed.

Karypis, G., 2013. *METIS - A Software Package for Partitioning Unstructured Graph, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 5.1.0*. Tech. rep., Department of Computer Science and Engineering, University of Minnesota.

Sleutelwoord

DRAFT

DRAFT

Index

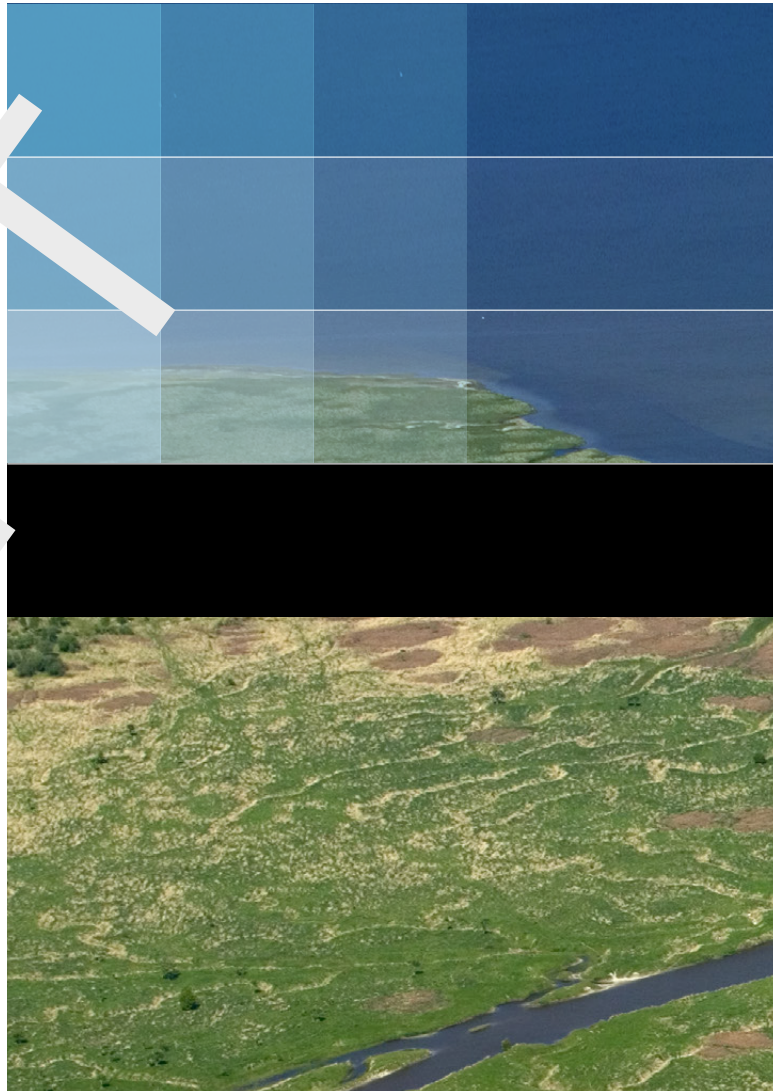
sleutelwoord, [19](#)

DRAFT



DRAFT

DRAFT



Deltares systems

PO Box 177
2600 MH Delft
Boussinesqweg 1
2629 HV Delft
The Netherlands

+31 (0)88 335 81 88
software@deltares.nl
www.deltares.nl/software

