Deltares Integrated Model Runner

DIMR

Deltares system

DRAFT

User Manual

Deltares

Deltares Integrated Model Runner (DIMR)

Manual

User manual

Version: 1.0
SVN Revision: 48961

2020-10-30

Deltares Integrated Model Runner (DIMR), User manual

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: info@deltares.nl
www: https://www.deltares.nl

For sales contact:
telephone: +31 88 335 81 88
e-mail: software@deltares.nl
www: https://www.deltares.nl/software

For support contact:
telephone: +31 88 335 81 00
e-mail: software.support@deltares.nl
www: https://www.deltares.nl/software

Contents

List of Tables

List of Figures

1 Introduction

This User Manual concerns the module Deltares Integrated Model Runner (DIMR).

DIMR is a small program to run combinations of simulation cores, components, ensuring that data is exchanged at the right timing. In a DIMR configuration file, the user can describe how the components should be combined and what data has to be exchanged.

1.1 About this document

To make this manual more accessible we will briefly describe the contents of each chapter.

Chapter 2: Usage describes the basics on DIMR usage.

1.2 Versions

This is the draft version of this document.

1.3 Manual version and revisions

This manual applies to:

- DIMR version 1.0–1.3

1.4 Changes with respect to previous versions

This is the first edition.

2 Usage

2.1 Basic functionality

2.1.1 Introduction

DIMR stands for Deltares Integrated Model Runner. The module is dedicated to deploy modules that simulate physical phenomena, such as hydraulics or waves, and the exchange of data between them.

2.1.2 Running a computation

The following recipe can be used for both Windows and Linux computations (modifications for Linux are in notes below the recipe):

- ◇ Locate the file <run_dimr.bat> on your system. It is in your installation folder within in sub-directory <...\plugins\DeltaShell.Dimr\kernels\x64\scripts>. (One directory up is a set of folders, one for each component.)
- ◇ Open a console/command box in the directory where you placed your exported model. The DIMR configuration file should be in this directory, see Figure 2.1
- ◇ Execute <run_dimr.bat> in this location

Note: Note the quotes around <run_dimr.bat> and its path; this is necessary when the path contains white spaces.

Note: When the DIMR configuration file has another name than <dimr_config.xml>, then this name must be given as argument of <run_dimr.bat>. A "usage" text is shown on errors and when adding argument "--help" to <run_dimr.bat>.

Note: When a kernel runs in parallel using MPI, the script <run_dimr_parallel.bat> should be used instead of <run_dimr.bat>; it is located in the same directory.

Note: On Linux, the recipe is identical. The name of the Linux run script is <run_dimr.sh>, located in directory <lnx64/bin>. A submit-script is needed when using a computation cluster with a queueing tool and when doing parallel computations. A script, named <submit_dimr.sh> is available for the Deltares cluster; this script can be used, after modification, for clusters outside Deltares.

Note: On Linux, when copying the binaries manually to another location, you have to execute the script <bin/libtool_install.sh> after the copy action.

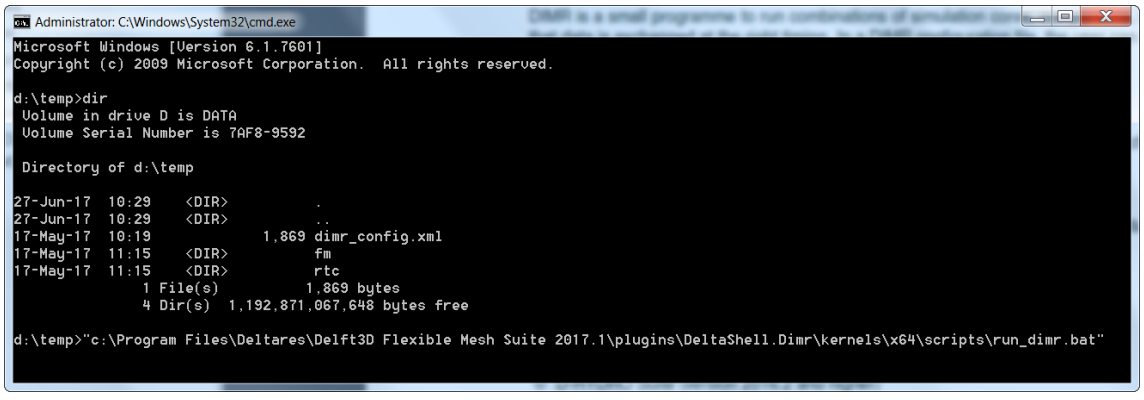


Figure 2.1: *Console example for running DIMR*

2.1.3 Time management

Time is essential and each component is free to handle time in its own way. Some components have *Boundary Conditions* which are specified in a particular calendar, for example the *Julian* calendar. Other components do not use a specific point of time.

Currently DIMR supports the following *Integrated models*:

- Stand alone execution of each individual component D-Flow 1D, D-Flow FM, D-Water Quality and D-Waves
- D-Flow 1D and D-RR: Both components must have the same start-date. Both models will start at time equals zero.
- D-Flow 1D and D-RTC: The reference-dates of both components must be equal.
- D-Flow 1D and D-Water Quality: First execute D-Flow 1D, then D-Water Quality. This is possible using one DIMR configuration file.
- D-Flow FM and D-RTC: See the combination of D-Flow 1D and D-RTC above.
- D-Flow FM and D-Waves: The reference-date of both components must be equal.
- D-Flow FM, D-Waves and D-RTC: See items above.
- D-Flow FM and D-Water Quality: First execute D-Flow FM, then D-Water Quality. This is possible using one DIMR configuration file.

Note: The D-RTC timestep must match the timestep as specified in the DIMR configuration file, since D-RTC will perform exactly one (privately defined) timestep on each call.

2.2 Technical Information

2.2.1 Introduction

Section 2.2.2 describes the BMI interface.

Section 2.2.3 to section 2.2.6 describe the DIMR configuration file.

2.2.2 BMI interface

The communcation between DIMR and the components is according to the BMI-interface (https://csdms.colorado.edu/wiki/BMI_Description). All child calculation components in DIMR (D-Flow 1D, D-Flow FM, D-Waves, etc.) are BMI-compliant. The basic BMI-interface implements:

- initialize
- update (perform one timestep, or more - upto the simulation timespan)
- finalize
- get (data)
- set (data)

Note: DIMR itself is also BMI-compliant; The DIMR executable uses the BMI-interface in the communication with DIMR library.

2.2.3 The DIMR configuration file

The DIMR input file is in XML format. Section 2.3 contains an example.

2.2.4 The DIMR schema file

The DIMR input file is in XML format it can be validated agains an XML schema. Section 2.4 contains an example.

2.2.5 Sequential and parallel simulations

DIMR enables sequential and parallel simulations in three ways:

1 **Sequential simulations**: Component 1 is executed for its full simulation period, output is produced, then component 2 is executed, optionally using the output produced by simulation component 1. Component 2 can not influence component 1. This is normally the configuration when doing a D-Flow FM calculation followed by a D-Water Quality calculation. See section 2.3.2 for an example.

2 **Parallel simulation**: Components 1 and 2 are both started, component 1 simulates a time period, exchanges data with component 2, component 2 is executed and exchanges data with component 1. This is repeated until the full simulation period is handled. This is normally the configuration when doing a D-Flow FM simulation containing a structure being controlled based on hydrodynamic results, for example the water level at a certain location. Also the parallel simulation of D-Flow FM with D-Waves uses this configuration. A combination of D-Flow FM, D-RTC and D-Waves, with different communication frequencies is possible. In section 2.3: Example configuration file for parallel, an example is given for the parallel simulation of D-Flow FM with D-RTC. Mark the following lines:

   ◇ *<parallel>* Parallel simulation
   ◇ *<start name="dflowfm" />* Start the "master" component
   ◇ *<startGroup>* Start a "slave" component
   ◇ *<start name="Real-Time Control" />*

   Note: Inside a parallel simulation, exactly one component must act as the "master" simulation. All other components must be defined using the *<startGroup>* block. Normally, the flow component is the "master" simulation.

   Note: The order of appearance in the *<control>* block defines the order of execution during an "update" event.

3 **Parallel simulation using a parallel component**: In the examples above, D-Flow FM itself can run in parallel, using several partitions to do the flow simulation, exchanging data directly via MPI. In the example in section 2.3: DIMR configuration file (XML-format), the line *<process>0 1 2</process>* indicates that in a multi processor computation, this component should run in the first three processes. Also the following "magic" line must be added: *<mpiCommunicator>DFM_COMM_DFMWORLD</mpiCommunicator>*.

2.2.6 Data exchange

When DIMR has to take care of data-exchange, it must be specified in the config file using a *<coupler>* block. Each coupler has a unique name and can optionally be used more than once in the *<control>* block. Currently, only scalar quantities can be exchanged by DIMR. In the example in section 2.3: DIMR configuration file (XML-format), two couplers are used; their definitions (source/target components, source/target itemNames) are at the end of the example.

2.3 DIMR configuration file (XML-format)

2.3.1 Protected file extensions

In Table 2.1 a listing of the protected file extension is given, these file extensions are used by the calculation components of DIMR to detect which importer should be used for importing the model.

Table 2.1: *Protected file name and extensions*

Extension or File	Importer
∗.mdu	D-Flow FM
∗.md1d	D-Flow 1D
settings.json	D-RTC

Remark:

- ◇ If the name of the input file for D-RTC is not given, the assumptions is made that the input xml-files are located at the sub-directory: <./rtc/∗.xml>

2.3.2 Example sequential configuration file

```
<?xml version="1.0"?>
<dimrConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
   <documentation>
      ...
   </documentation>
   <control>
      <start name="curved_bend_2d" />
      <start name="curved_bend_3d" />
   </control>
   <component name="curved_bend_2d">
      ...
   </component>
   <component name="curved_bend_3d">
      ...
   </component>
</dimrConfig>
```

2.3.3 Example configuration file for parallel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dimrConfig xsi:schemaLocation="http://schemas.deltares.nl/dimr
   http://content.oss.deltares.nl/schemas/latest/dimr.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://schemas.deltares.nl/dimr">
   <documentation>
      <fileVersion>1.03</fileVersion>
      <createdBy>Deltares, Coupling team</createdBy>
      <creationDate>2015-05-20T07:56:32+01</creationDate>
   </documentation>
   <control>
      <parallel>
         <startGroup>
            <time>0.0 60.0 99999999.0</time>
            <coupler name="flow2rtc"/>
            <start name="myNameRTC"/>
            <coupler name="rtc2flow"/>
         </startGroup>
         <start name="myNameDFlowFM"/>
      </parallel>
   </control>
   <component name="myNameDFlowFM">
```

```
      <library>dflowfm</library>
      <process>0 1 2</process>
      <mpiCommunicator>DFM_COMM_DFMWORLD</mpiCommunicator>
      <workingDir>fm</workingDir>
      <inputFile>weirtimeseries.mdu</inputFile>
      <setting key="verbose" value="DEBUG"/> <!-- Set level to many messages before initialize -->
      <setting key="bloomspecies" value="/opt/delft3dfm_latest/lnx64/share/delft3d/bloom.spe"/>
      <setting key="processlibrary" value="../../substances/proc_def.dat"/>
      <parameter key="verbose" value="FATAL"/> <!-- Set level to less messages before update -->
   </component>
   <component name="myNameRTC">
      <library>RTCTools_BMI</library>
      <process>0</process>
      <workingDir>rtc</workingDir>
      <!-- component specific -->
      <inputFile>settings.json</inputFile>
   </component>
   <coupler name="flow2rtc">
      <sourceComponent>myNameDFlowFM</sourceComponent>
      <targetComponent>myNameRTC</targetComponent>
      <item>
         <sourceName>observations/Upstream/water_level</sourceName>
         <targetName>input_ObservationPoint01_water_level</targetName>
      </item>
   </coupler>
   <coupler name="rtc2flow">
      <sourceComponent>myNameRTC</sourceComponent>
      <targetComponent>myNameDFlowFM</targetComponent>
      <item>
         <sourceName>output_weir_crest_level</sourceName>
         <targetName>weirs/weir01/crest_level</targetName>
      </item>
   </coupler>
</dimrConfig>
```

2.4 DIMR schema file (XSD-format)

2.4.1 Introduction

XSD (XML Schema Definition) specifies how to formally describe the elements in an Extensible Markup Language (XML) document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed.

XSD can also be used for generating XML documents that can be treated as programming objects. In addition, a variety of XML processing tools can also generate human readable documentation, which makes it easier to understand complex XML documents.

In general, a schema is an abstract representation of an object's characteristics and relationship to other objects. An XML schema represents the interrelationship between the attributes and elements of an XML object (for example, a document or a portion of a document). The process of creating a schema for a document involves analyzing its structure and defining each structural element encountered. For example, a schema for a document describing a website would define a website element, a webpage element, and other elements that describe possible content divisions within any page on that site. Just as in XML and HTML, elements are defined within a set of tags.

We have defined the <dimr.xsd> to validate our generated DIMR config xml-files before parsing the xml-file in the DIMR executable and running it via the DIMR library with the DIMR computational cores (D-Flow FM, D-Flow 1D, D-RR, etc).

Section 2.4.2 describes the definition of the main element of the DIMR configuration xml.
Section 2.4.3 describes the definition of the required elements of the DIMR configuration xml.

Section 2.4.4 describes the definition of the required elements of the DIMR configuration xml.

2.4.2 dimrConfig

This is our main element for the DIMR xml configuration.

Note: Please note it validates the xml sequential.

It need an element to provide information about the DIMR configuration, an element to control the flow between the computational cores and an element to define the computational cores as components.

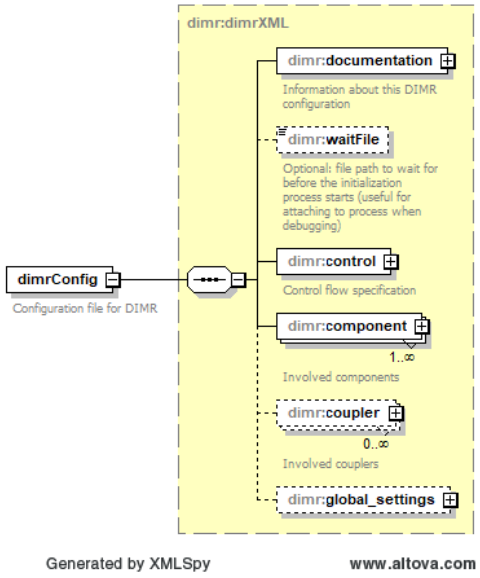


Figure 2.2: *Configuration for DIMR*

2.4.3 Required elements

The three 'required' elements (besides the obvious documentation element) for a DIMR configuration are:

1. control
2. component
3. coupler

2.4.3.1 Documentation

Although this is mandatory for the dimrConfig section is is not realy well set up. It is meant too check if the generated DIMR config xml can be validated agaist the xsd. It also has other informative elements but they are not really relevant to the configuration

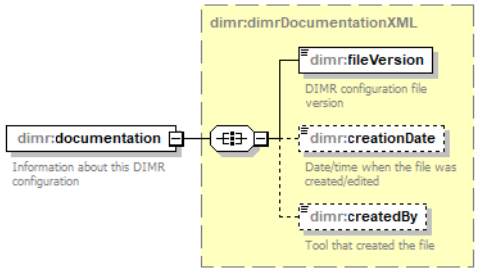


Figure 2.3: *Documentation*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dimrConfig xsi:schemaLocation="http://schemas.deltares.nl/dimr
         http://content.oss.deltares.nl/schemas/latest/dimr.xsd"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://schemas.deltares.nl/dimr">
   <documentation>
      <fileVersion>1.03</fileVersion>
      <createdBy>Deltares, Coupling team</createdBy>
      <creationDate>2015-05-20T07:56:32+01</creationDate>
   </documentation>
   ...
</dimrConfig>
```

2.4.3.2 Control

Note: The order of appearance in the *<control>* block defines the order of execution during an "update" event.

Control flow specification for the DIMR-execution. Executed sequentially, except for components in a "parallel" element

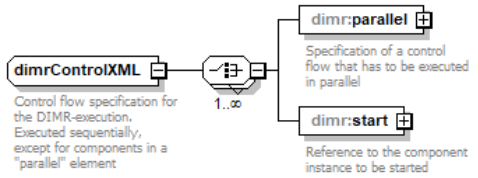


Figure 2.4: *Control flow*

If running sequentially we need to specify the computational core component in the **start** element with an attribute **name**

We can specify a control flow that has to be executed in parallel

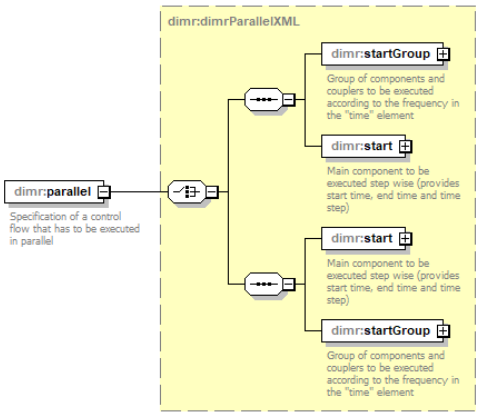


Figure 2.5: *Control flow parallel*

You can see we need to set a start group of components and couplers to be executed according to the frequency in the "time" element. The element **start** can be found here again but also the reference coupler connecting the computational core components. For coupler see section 2.4.3.5.

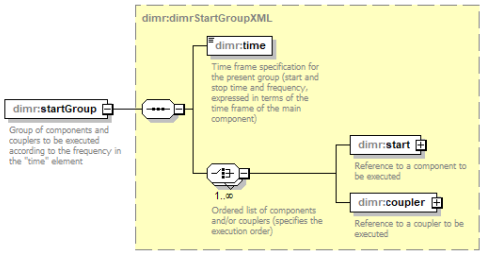


Figure 2.6: *Control flow parallel, Start group*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dimrConfig xsi:schemaLocation="http://schemas.deltares.nl/dimr
        http://content.oss.deltares.nl/schemas/latest/dimr.xsd"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://schemas.deltares.nl/dimr">
   ...
   <control>
      <parallel>
         <startGroup>
            <time>0.0 60.0 99999999.0</time>
            <coupler name="flow2rtc"/>
            <start name="myNameRTC"/>
            <coupler name="rtc2flow"/>
         </startGroup>
         <start name="myNameDFlowFM"/>
      </parallel>
   </control>
   ...
</dimrConfig>
```

2.4.3.3 Component

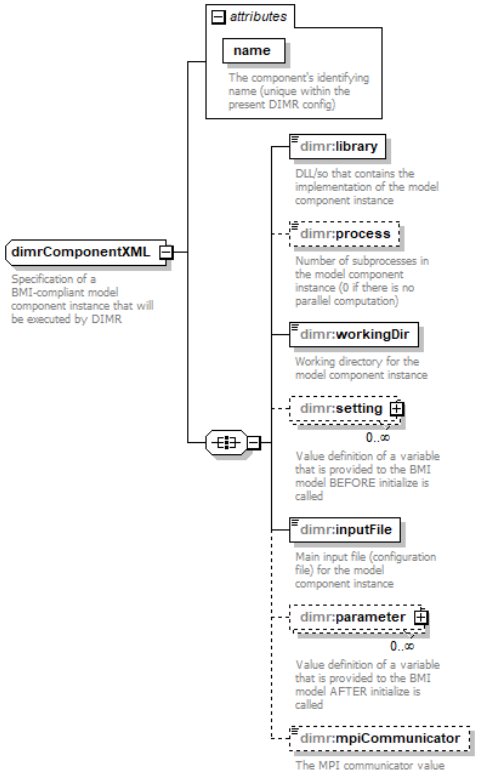


Figure 2.7: *Configuration for DIMR computational core component*

2.4.3.4 Computational core component settings and parameters

Note: Please note setting and parameter can be used 0 - multiple times with a key / value structure

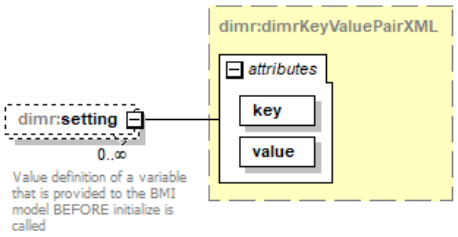


Figure 2.8: *Setting to be set to calculation core component before bmi model initialize is called*

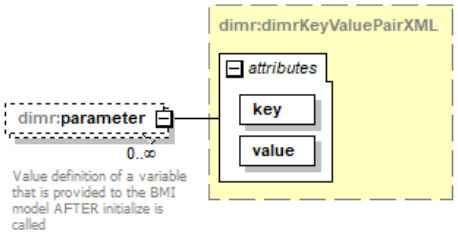


Figure 2.9: *Parameter to be set to calculation core component after bmi model initialize is called*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dimrConfig xsi:schemaLocation="http://schemas.deltares.nl/dimr
         http://content.oss.deltares.nl/schemas/dimr-1.3.xsd"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://schemas.deltares.nl/dimr">
   ..
   <component name="myNameDFlowFM">
      <library>dflowfm</library>
      <process>0 1 2</process>
      <mpiCommunicator>DFM_COMM_DFMWORLD</mpiCommunicator>
      <workingDir>fm</workingDir>
      <inputFile>weirtimeseries.mdu</inputFile>
      <setting key="verbose" value="DEBUG"/> <!-- Set level to many messages -->
      <setting key="bloomspecies" value="/opt/delft3dfm_latest/lnx64/share/delft3d/bloom.spe"/>
      <setting key="processlibrary" value="../../substances/proc_def.dat"/>
      <parameter key="verbose" value="FATAL"/> <!-- Set level to less messages -->
   </component>
..
</dimrConfig>
```

2.4.3.5 Coupler

The last required element is **coupler**. This is only required if you are running an integrated model with more than one computational core component. This element specifies the coupling action to be performed between two BMI-compliant model computation core components

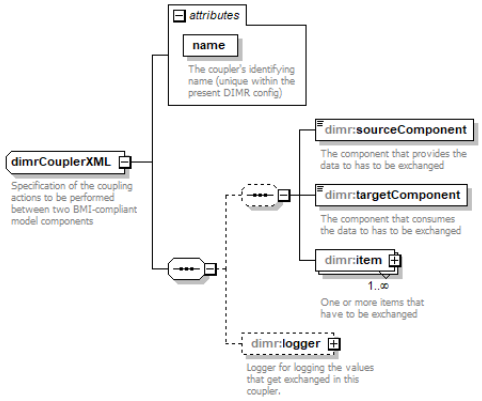


Figure 2.10: *coupler*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<dimrConfig xsi:schemaLocation="http://schemas.deltares.nl/dimr
         http://content.oss.deltares.nl/schemas/latest/dimr.xsd"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://schemas.deltares.nl/dimr">
   ...
   <coupler name="rtc2flow">
   <sourceComponent>myNameRTC</sourceComponent>
   <targetComponent>myNameDFlowFM</targetComponent>
      <item>
         <sourceName>output_weir_crest_level</sourceName>
         <targetName>weirs/weir01/crest_level</targetName>
      </item>
   </coupler>
</dimrConfig>
```

2.4.4 Optional elements

All the following sections are in principle optional elements for a DIMR configuration:

2.4.4.1 dimrConfig - waitFile

File path to wait for before the initialization process starts (useful for attaching to process when debugging)

2.4.4.2 dimrConfig - coupler

As described in section 2.4.3.5 this is only required if you need to calculate a model in parallel with more than one computational core components.

2.4.4.3 dimrConfig - global_settings

This optional element contains settings to be used by the DIMR executable or DIMR libary it self. It is not for the computational core components.

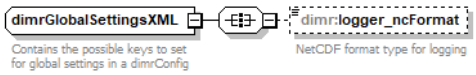


Figure 2.11: *Global DIMR settings*

2.4.4.4 dimrComponent - Setting / Parameter

See section 2.4.3.4

Table 2.2: *Setting and Parameter for Computational Core component*

	Description
Setting	key/value to be set to calculation core component before bmi model initialize is called
Parameter	key/value to be set to calculation core component after bmi model initialize is called

2.4.4.5 dimrComponent - process

Number of subprocesses in the model component instance (0 if there is no parallel computation)

2.4.4.6 dimrComponent - mpiCommunicator

The MPI communicator value: `DFM_COMM_DFMWORLD`

2.4.4.7 dimrCoupler - logger

Logger for logging the values that get exchanged in this coupler.

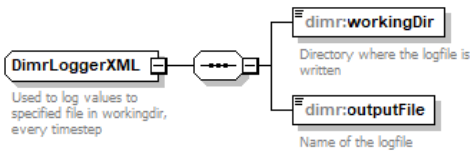
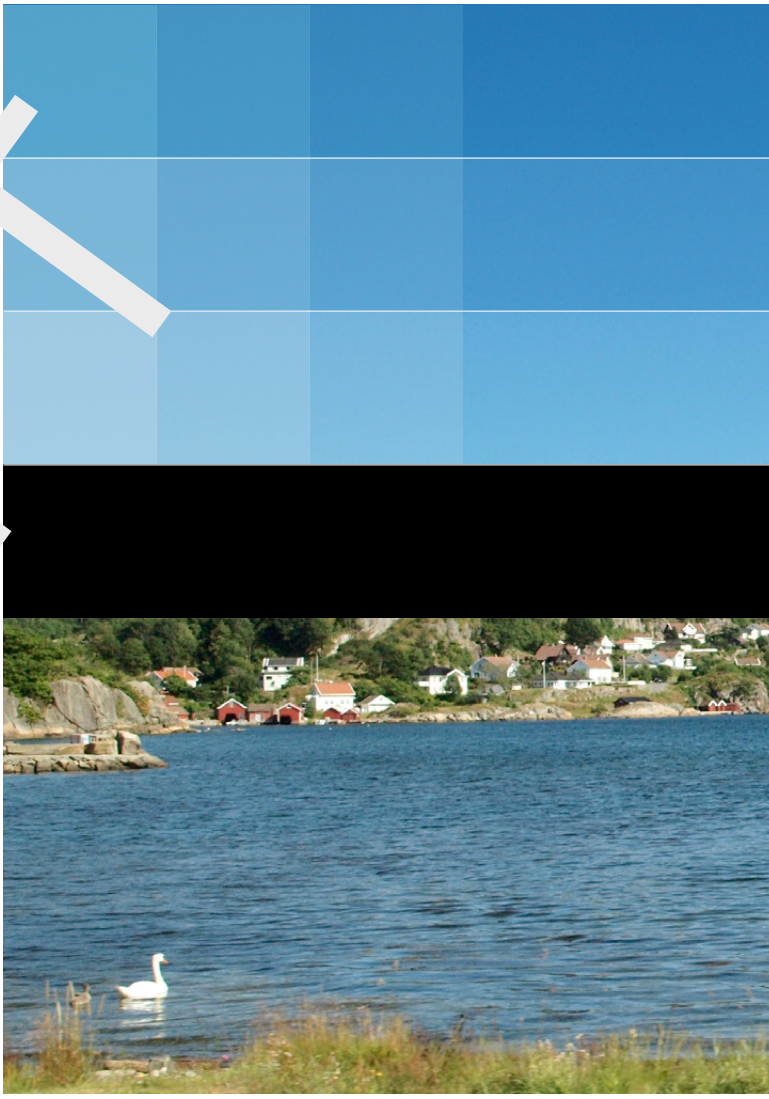


Figure 2.12: *Log exchanged values in this coupler between the computational core components*



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