Delft-FEWS-DIMR-adapter

The DIMR-software is used to run a SOBEK 3 model in Delft-FEWS. In the *General Adapter* of Delft-FEWS we refer to the different components of the dimr-tool. This wiki will describe what you need to do to convert a SOBEK3 model to a dimr-model and how to set-up the different components of your Delft-FEWS configuration. Read the steps in this guide to implement your SOBEK 3 model in Delft-FEWS using the dimr-software. When updating an existing model, some steps are not necessary.

How to set-up a DIMR Modules folder?

The Delft-FEWS DIMR-adapter requires a fixed folder structure:

dimr_bin	sub-folder with binaries for the DIMR-software.
dimr_model	sub-folder with DIMR-files that describe the SOBEK3 model.
fews-dimr-adapter-bin	sub-folder with binaries for the fews adapter that communicates with the DIMR.
Input	sub-folder with input files requried to run the model.
Logs	sub-folder for the log-files that are created during the model run.

Follow the steps below to set-up a DIMR model folder in your Delft-FEWS system. Step 1 can be skipped when updating an existing model.

Step 1: create the dimr model folder structure in Delft-FEWS

Create a rootDir folder for your SOBEK 3 model at *$REGION_HOME$/Modules/<put_your_rootDir_here>*. All sub-folders described in the table above can be put in this Modules folder.

The contents of the Modules folder can now be filled. We will start with the getting the **dimr_bin** files.

Step 2: getting the contents of the dimr_bin

There are two ways for getting the dimr_bin files:

- The preferred method is getting the dimr_bin files from the used SOBEK 3 installation in which the model is created. Typically, the files are found in c:\Program Files (x86)\Deltares\SOBEK (xxx)\plugins\DeltaShell.Dimr\kernels\x64, where xxx is the SOBEK version. Copy the x64 directory to dimr_bin folder.
- Another method is to getting the dimr_bin files from the Deltares Build server which can be found using the following link. This is not preferred as these are the latest development releases of DIMR Which version you will need depends on your used SOBEK 3 model version. Which DIMR version belongs to which SOBEK 3 version is therefore currently not known. If you are sure which version to use, l ick the artifacts button  , select either the Windows or Linux zip file to download and unpack in the dimr_bin directory.

It is good practice to add a README.txt file in the dimr_bin directory where you document to DIMR version and from where you got the DIMR files.

The actual model files must be created by converting your SOBEK 3 model. DeltaShell contains standard functionality to aid you in this step.

Step 3: generate the dimr_model files

- In DeltaShell open your SOBEK 3 model.

It is possible that a message will appear in DeltaShell stating that your project was made in an older version and needs to be migrated to the latest version of DeltaShell. Press "Yes" and the model will automatically migrate to this version. If this is not preferred, because of new model developments or stability, use the DeltaShell which belongs to the SOBEK 3 version you used to create the model.

- Before you continue it is recommended to run the model by pressing the Run All button in the DeltaShell GUI. This is an easy way to check if all functionality is up-to-date. Check the log messages and fix all issues that are mentioned.

When your model can finish a Run All without displaying errors it is possible to convert the SOBEK 3 model to a dimr-model.

- In DeltaShell right-click on your Integrated Model (e.g. Rijn) and select the option "Export...". In the menu that opens you now select the option "DIMR configuration" and you follow the instructions.
- Put the all model files that are generated in the folder **dimr_model**.

The contents of the **fews-dimr-adapter-bin** folder can be downloaded.

Step 4: download the contents of the fews-dimr-adapter-bin folder

The DIMR and Delft-FEWS are communicating through the fews adapter. This adapter is part of Delft-FEWS and must be downloaded from the build.deltares.nl:

- http://build.deltares.nl/project.html?projectId=FewsDevelopment&tab=projectOverview
- Be sure the get the adapter from the environment corresponding to your FEWS version (click on to view other version)
- Select the latest stable version (or the Delft-FEWS development version): FEWS – Development > install adapters > artifacts > fews-dimr-adapter-bin.XXXXX.zip.
- Put all these unzipped files in the folder fews-dimr-adapter-bin

The folders **Input** and **Logs** will be filled when you run the model.

How to run the DIMR model?

Run DIMR model

When the contents of every directory is set-up, we can run the DIMR model. The model can be executed using a batch file with an argument pointing to the DIMR configuration xml file.

- To run the model, execute the following batch file: .\dimr_bin\x64\dimr\scripts\run_dimr.bat DIMR_CONFIG.xml (e.g. Rhine_LobRTK.xml or IJG.xml)

You could enter this command in a seperate batch file for easier starting the model run.

How to get the desired output from the DIMR model?

Run DIMR model

In SOBEK 3 you can specify for which parameters you want to write the output. This is the same in the DIMR model and can be specified in the *.md1d file located in .\dimr_model\dflow1d.

Default, all results options are set to None. Apply the same output options as used in the SOBEK 3 model (e.g. Current, Average, ...)

How to generate cache files to read the SOBEK 3 DIMR model faster?

SOBEK 3 DIMR cache files

DIMR has the option to make cache files of the SOBEK 3 model to read the model itself faster. This can be up to 100 times faster, because it does not have to processing large ASCII files. Therefore, it is good practice to generate the cache files.

To make the cache files, do the following:

- Go to *FEWS_YourSystem / Modules / NewFolder / dimr_model / dflow1d* and open the **.md1d* file.
- In the file, enter under \[AdvancedOption] 'CacheMode = Write' and run the DIMR model

```
[AdvancedOptions]
CacheMode = Write
CalculateDelwaqOutput = 0
ExtraResistanceGeneralStructure = 0.0
FillCulvertsWithGL = 0                    # 0=false, 1=true
```

- In the dflow1d folder the .cache files are made.
- Add these .cache files to the dflow1d folder in your *ModuleDataSetFiles.*
- Lastly, set CacheMode = Write to CacheMode = Read to ensure the DIMR does not generate new cache files as this is the same as not using cache files.

```
[AdvancedOptions]
CacheMode = Read
CalculateDelwaqOutput = 0
ExtraResistanceGeneralStructure = 0.0
FillCulvertsWithGL = 0                    # 0=false, 1=true
```

How to generate ColdStateFiles?

ColdStateFiles

Delft-FEWS will need a state file to be able to run the SOBEK model. In order to generate the first state files of the SOBEK model it is recommended to do a stand-alone run of the DIMR from the command line (see How to run the DIMR model? how to do that).

- Go to *FEWS_YourSystem / Modules / NewFolder / dimr_model / dflow1d* and open the **.md1d* file.
- Set the correct restart options:
  - Older version:
    - Go to option WriteRestart and make sure to set this setting on true. The model will now generate its state files.

```
[SimulationOptions]
Debug = 0                    # 0=false, 1=true
DebugTime = 0
DispMaxFactor = 0.45
DumpInput = 0                    # 0=false, 1=true
Iadvec1D = 2
Limtyphu1D = 1
Momdilution1D = 1
TimersOutputFrequency = 1
UseRestart = 1                    # 0=false, 1=true
WriteRestart = 1                    # 0=false, 1=true
UseTimers = 1                    # 0=false, 1=true
WriteNetCDF = 1                    # 0=false, 1=true
```

  - Newer versions:
    - Under \[Restart] there are 5 options available. These reflect the restart options in SOBEK 3.
    - To make a restart file, set UseRestart = 0 and WriteRestart = 1. The start and stop time should be set very wide like in the example below to ensure a restart will always be written. Intermediate restartfiles can be written using the RestartTimeStep option. Note: on the end of the simulation run, a restart will always be written if WriteRestart is set to 1 independent of the RestartTimeStep.

```
[Restart]
UseRestart = 0                    # 0=false, 1=true
WriteRestart = 1                    # 0=false, 1=true
RestartStartTime = 1001-01-01 00:00:00 # yyyy-MM-dd HH:mm:ss
RestartStopTime = 3001-01-01 00:00:00 # yyyy-MM-dd HH:mm:ss
RestartTimeStep = 36000                # in seconds
```

- The SOBEK 3 model has default timeseries. We can thus do a stand-alone run of the model without providing the model with data.
- Run the DIMR model.

```
FEWS_Vecht_SA\Modules\Sobek3\ovd_dv\dimr_model>..\dimr_bin\run_dimr.bat .\ovd_dv.x
```

or in newer versions of the software

```
d:\1_FEWS_Projecten\NEDERLAND\RWS\IWP\1. Delft-FEWS\FEWS_IWP_DEV\Model\IJG-SOBEK3\dimr_model>..\dimr_bin\win64\scripts\run_dimr.bat IJG
```

- After the model has run succesfully it is possible to create a ColdState-file for Delft-FEWS.
- In the Config\ColdStateFiles\ create a zip for the cold state files. In our example this is the Walrus_Hydr_Sobek3_Update Default.zip
- The zip file for the ColdState files contains the following information
  - Walrus_Hydr_Sobek3_Update Default \ dflow1d \ sobek.rda
  - Walrus_Hydr_Sobek3_Update Default \ dflow1d \ sobek.rdf
  - Walrus_Hydr_Sobek3_Update Default \ rtc \ state_import.xml
- When you analyse the model results you will find a sobek.nda and sobek.ndf file in the *dimr_model* \ *dflow1d* folder and a state_export.xml in the *dimr_model* \ *rtc* folder. You can use these files. Make sure to rename the extension of these files.
- Set UseRestart to 1 in the *.md1d file to ensure the model will be using the restart.

```
[Restart]
UseRestart = 1                    # 0=false, 1=true
WriteRestart = 1                    # 0=false, 1=true
RestartStartTime = 1001-01-01 00:00:00 # yyyy-MM-dd HH:mm:ss
RestartStopTime = 3001-01-01 00:00:00 # yyyy-MM-dd HH:mm:ss
RestartTimeStep = 36000                # in seconds
```

for newer versions of the software

What information should I put in the ModuleDataSetFiles?

ModuleDataSetFiles

Delft-FEWS can distribute model files to FSS machines. It is recommended to zip the model files and put it in the Delft-FEWS configuration.

- In the Config\ModuleDataSetFiles create a zip file for the (update) model run. In our example the name for this zip file is Walrus_Hydr_Sobek3_Update.zip
- The zip file for the ModuleDataSetFile contains the following information:
  - .\dlfow1d\.*.
  - .\rtc\.*.
  - DIMR_CONFIG_NAME.xml (e.g. Rhine_LobRTK.xml or IJG.xml)
  - Be sure to exclude .log, .rda, .rdf, .nda, .ndf, .\dlfow1d\ouput dir, state_import.xml and state_export.xml
- As you might have notice this is the content of the dimr_model file created previously.

How do I get hold of the import and export timeseries of the SOBEK 3 / DIMR model?

Not yet completed

How to create the General Adapter Module?

For more detailed information on the use of the General Adapter Module, see this link.

An example of the **general** section of the general adapter can be found below.

General section of Module

general		
	description	SOBEK 3 Model IJsselmeergebied
	rootDir	$REGION_HOME$/Modules/Sobek3/IJS
	workDir	%ROOT_DIR%/dimr_model
	exportDir	%ROOT_DIR%/input
	exportDataSetDir	%ROOT_DIR%/dimr_model
	exportIdMap	Id_SBK3_IJS_Export
	importDir	%WORK_DIR%/dflow1d/output
	importIdMap	Id_SBK3_IJS_Import
	dumpFileDir	%REGION_HOME%/Dump
	dumpDir	%ROOT_DIR%
	diagnosticFile	%ROOT_DIR%/Logs/diagnostic.xml
	convertDatum	true
	timeZone	
		timeZoneOffset +00:00

In the example configuration above it can be seen that the working directory of the DIMR should always be the folder *./dimr_model*. The Delft-FEWS exports timeseries to the folder *.\input*. After the modelrun is completed the output of the model is expected in the folder *.\dflow1d\output*.

The **activities** section is divided into four sub-sections: startUp-, export-, execute- and importActivities. The contents of all of these sections will be discussed in the info sections below. When you want to test your configuration during the building process then it is recommended to start with the startUpActivity, ExportActivities and ExecuteActivity sections. When no information (e.g. timeseries, states) are provided to the model DIMR will take its own default timeseries and states to perform a model run. Hence, this trick enables you to test if the model can be run from Delft-FEWS. The *importDir* can be checked to see if output has been generated.

Similarly, it is also possible to define a single input timeseries in the exportActivities. The DIMR will add default values to the missing timeseries in order to start the run. This trick allows you to check early on whether your ExportActivity and IdMapping is working properly.

startUpActivities

The startUpActivities are typically used to purge the model directory in the *Modules* folder to ensure a clean slate. See example below

```
<startUpActivities>
    <purgeActivity>
        <filter>%ROOT_DIR%/input/*.*</filter>
    </purgeActivity>
    <purgeActivity>
        <filter>%ROOT_DIR%/logs/*.*</filter>
    </purgeActivity>
    <purgeActivity>
        <filter>%WORK_DIR%/*.*</filter>
    </purgeActivity>
    <purgeActivity>
        <filter>%WORK_DIR%/dflowld/*.*</filter>
    </purgeActivity>
    <purgeActivity>
        <filter>%WORK_DIR%/dflowld/output/*.*</filter>
    </purgeActivity>
    <purgeActivity>
        <filter>%WORK_DIR%/rtc/*.*</filter>
    </purgeActivity>
</startUpActivities>
```

exportActivities

The exportActivities are used to export all the necessary data and the model itself. These typically consists of (in order): exportStateActivity, exportTimeSeriesActivities, exportDataSetActivity and exportRunFileActivity. See examples below.

exportStateActivity

This activity exports the state of the model. This can be a cold state (the one generated under How to generate ColdStateFiles?) or a warm state saved in the Delft-FEWS database.

```
<exportStateActivity>
    <moduleInstanceId>SBK3_LobRTK_Update</moduleInstanceId>
    <stateExportDir>%WORK_DIR%</stateExportDir>
    <stateSelection>
        <warmState>
            <stateSearchPeriod unit="hour" start="-192" end="-1"/>
        </warmState>
    </stateSelection>
</exportStateActivity>
```

exportTimeSeriesActivity

This activity exports all the necessary data from FEWS to be used in the model. There can be multiple activities.

```
<exportTimeSeriesActivity>
    <exportFile>export_pi_flowId.xml</exportFile>
    <timeSeriesSets>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_QBC_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>Q.ubc</parameterId>
            <locationSetId>SBK3_Lob_QBC</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <relativeViewPeriod unit="hour" end="0"/>
            <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_QBC_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>Q.ubc</parameterId>
            <locationSetId>SBK3_LobRTK_QBC</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <relativeViewPeriod unit="hour" end="0"/>
            <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_HBC_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>H.ubc</parameterId>
            <locationSetId>SBK3_LobRTK_HBC</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <relativeViewPeriod unit="hour" end="0"/>
            <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
    </timeSeriesSets>
</exportTimeSeriesActivity>
```

exportDataSetActivity

This activity exports the ModuleDataSet file to the working directory.

```
<exportDataSetActivity>
    <moduleInstanceId>SBK3_LobRTK_Update</moduleInstanceId>
</exportDataSetActivity>
```

exportRunFileActivity

This activity makes a run xml file which the pre- and post adapter uses to distribute the timeseries to the correct locations for the model to pick up.

```
<exportRunFileActivity>
    <exportFile>pi-run.xml</exportFile>
    <properties>
        <string key="modelDir" value="."/>
        <string key="flowldModelDir" value="dflowld"/>
        <string key="flowldModelOutputDir" value="dflowld/output"/>
        <string key="rtcModelDir" value="rtc"/>
        <string key="dimrConfigFile" value="Rhine_LobRTK.xml"/>
        <string key="mdldFile" value="rijn-flow-model.mdld"/>
        <string key="bcFile" value="BoundaryConditions.bc"/>
        <string key="piFileForFlowld" value="inputTimeSeriesFile-1"/>
        <string key="piFileForRtc" value="inputTimeSeriesFile-2"/>
    </properties>
</exportRunFileActivity>
```

executeActivities

```
<executeActivities>
    <executeActivity>
        <description>Run Sobek3 FEWS pre-adapter</description>
        <command>
            <className>nl.deltares.sobek.Sobek3PreAdapter</className>
            <binDir>%ROOT_DIR%/fews-dimr-adapter-bin</binDir>
        </command>
        <arguments>
            <argument>%ROOT_DIR%/input/pi-run.xml</argument>
        </arguments>
        <!--10 minutes-->
        <timeOut>600000</timeOut>
    </executeActivity>
    <executeActivity>
        <description>Run DIMR</description>
        <command>
            <executable>%ROOT_DIR%/dimr_bin/x64/dimr/scripts/run_dimr.bat</executabl
        </command>
        <arguments>
            <argument>Rhine_LobRTK.xml</argument>
        </arguments>
        <!--30 minutes-->
        <timeOut>1800000</timeOut>
        <!--The postprocessing step handles the ERROR messages-->
        <ignoreDiagnostics>true</ignoreDiagnostics>
        <ignoreExitCode>true</ignoreExitCode>
    </executeActivity>
    <executeActivity>
        <description>Run Sobek3 FEWS post-adapter</description>
        <command>
            <className>nl.deltares.sobek.Sobek3PostAdapter</className>
            <binDir>%ROOT_DIR%/fews-dimr-adapter-bin</binDir>
        </command>
        <arguments>
            <argument>%ROOT_DIR%/input/pi-run.xml</argument>
        </arguments>
        <!--10 minutes-->
        <timeOut>600000</timeOut>
    </executeActivity>
</executeActivities>
```

importActivities

In the ImportActivities the state files and the timeseries can be imported to Delft-FEWS.
It is important to convert the extension of the state files before you import it to Delft-FEWS, this can be done by adding the </relativeExportFile> to your configuration.

importActivities

Comment Import state

importStateActivity

stateImportDir %WORK_DIR%

stateFile (3)

	importFile	relativeExportFile
1	./dflow1d/sobek.nda	./dflow1d/sobek.rda
2	./dflow1d/sobek.ndf	./dflow1d/sobek.rdf
3	./rtc/state_export.xml	./rtc/state_import.xml

```
<importNetcdfActivity>
    <importFile>observations.nc</importFile>
    <timeSeriesSets>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>Q.us</parameterId>
            <locationSetId>SBK3_LobRTK_output</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <readWriteMode>add originals</readWriteMode>
        </timeSeriesSet>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>H.us</parameterId>
            <locationSetId>SBK3_LobRTK_output</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <readWriteMode>add originals</readWriteMode>
        </timeSeriesSet>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>H.us</parameterId>
            <locationSetId>SBK3_LobRTK_river_kilometers</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <readWriteMode>add originals</readWriteMode>
            <expiryTime unit="day" multiplier="$EXPIRYTIME_DAY_SIMULATED_LONGITUDINALPROFILE$"
        </timeSeriesSet>
    </timeSeriesSets>
</importNetcdfActivity>
<importNetcdfActivity>
    <importFile>structures.nc</importFile>
    <timeSeriesSets>
        <timeSeriesSet>
            <moduleInstanceId>SBK3_LobRTK_Update</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>H.us.crest</parameterId>
            <locationSetId>SBK3_LobRTK_structures</locationSetId>
            <timeSeriesType>simulated historical</timeSeriesType>
            <timeStep unit="hour"/>
            <readWriteMode>add originals</readWriteMode>
        </timeSeriesSet>
    </timeSeriesSets>
</importNetcdfActivity>
```

Improved DIMR adapter

The new DimrPreAdapter needs to be called with as only argument a reference to the run file.

Diagnostics can be ignored because it will write a plain text file dimr_pre_adapter.log in the workDir.

Execute DIMR pre adapter activity

```
<executeActivity>
        <description>Run DIMR FEWS pre-adapter</description>
        <command>
                <className>nl.deltares.dimr.DimrPreAdapter</className>
                <binDir>%ROOT_DIR%/fews-dimr-adapter-bin</binDir>
        </command>
        <arguments>
                <argument>%ROOT_DIR%/input/pi-run.xml</argument>
        </arguments>
        <timeOut>1200000</timeOut>
        <ignoreDiagnostics>true</ignoreDiagnostics>
</executeActivity>
```

In the pi run file property with key="dimrConfigFile" must be present with a reference to the dimr configuration file. This reference must be relative to the parent directory of the pi run file.

If multiple pi time series files are exported, it must be specified which file is meant for which component, there are 3 different properties for the components:

key="piFileForFlow1d"

key="piFileForFm"

key="piFileForFbc"

The time series xml files must be numbered in the order they appear in the <inputTimeSeriesFile> elements

PI-run.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Run xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.wldelft.nl/fews/PI" xsi:
schemaLocation="http://www.wldelft.nl/fews/PI http://fews.wldelft.nl/schemas/version1.0/pi-schemas/pi_run.xsd"
version="1.5">
    <timeZone>0.0</timeZone>
    <startDateTime date="2017-03-26" time="06:00:00"/>
    <endDateTime date="2017-03-29" time="06:00:00"/>
    <time0 date="2017-03-26" time="06:00:00"/>
    <workDir>X:\XXX\DimrPreAdapterTest\piRunFile1d2dRTC\dimr_model</workDir>
    <inputTimeSeriesFile>X:\XXX\piRunFile1d2dRTC\Input\export_pi_flow1d.xml</inputTimeSeriesFile>
    <inputTimeSeriesFile>X:\XXX\piRunFile1d2dRTC\Input\export_pi_fm.xml</inputTimeSeriesFile>
    <inputTimeSeriesFile>X:\XXX\piRunFile1d2dRTC\Input\export_pi_fbc.xml</inputTimeSeriesFile>
    <outputDiagnosticFile>notUsed.xml</outputDiagnosticFile>
    <properties>
                <!-- Reference to dimr configuration file, relative to parent directory of the/this pi-run.xml
file -->
                <!-- This file will be read to see which components of the dimr are used and find references to
.md1d, .mdu, flow1d2d.ini and/or fbc runtime config -->
        <string key="dimrConfigFile" value="../dimr_config.xml"/>
        <!-- Specify which of the above time series xml files is meant for flow1d -->
        <string key="piFileForFlow1d" value="inputTimeSeriesFile-1"/>
        <!-- Overwrite RstInterval value in the .mdu file with this property -->
        <string key="restartIntervalForFm" value="600"/>
        <!-- Specify which of the above time series xml files is meant for FM -->
        <string key="piFileForFm" value="inputTimeSeriesFile-2"/>
        <!-- Specify which of the above time series xml files is meant for FBC (rtc1) -->
        <string key="piFileForFbc" value="inputTimeSeriesFile-3"/>
        <!-- Will write the pi time series values for fbc as binary in timeseries_import.bin-->
        <string key="piTimeSeriesAsBin" value="true"/>
                <!-- Will convert the specified boundary NetCDF file to ASCII write the pi time series values
for fbc as binary in timeseries_import.bin-->
                <string key="dWaveNetcdfBoundaryFileToConvert" value="input/boundary.nc"/>
                <!-- Optional settings for conversion of boundary file to ASCII, values shown here are default--
>
        <string key="dWaveBoundaryFilePrefix" value="WW3_"/>
                <string key="dWaveBoundaryFileNumberOfCopies" value="3"/>


    </properties>
</Run>
```

The dimr configuration file will be read to find the components that are being used so it knows which files need to be changed.

dimr_config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dimrConfig xsi:schemaLocation="http://schemas.deltares.nl/dimr http://content.oss.deltares.nl/schemas/dimr-1.0.
xsd" xmlns="http://schemas.deltares.nl/dimr" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <documentation>
    <fileVersion>1.00</fileVersion>
    <createdBy>Deltares, Coupling Team</createdBy>
    <creationDate>2018-07-07T14:49:48.9157275Z</creationDate>
  <control>
    <parallel>
      <startGroup>
        <time>1 60 2678500</time>
        <coupler name="1d2d_to_rtc"/>
        <start name="real-time control"/>
        <coupler name="rtc_to_1d2d"/>
      </startGroup>
      <start name="1d2d"/>
    </parallel>
  </control>
  <component name="real-time control">
    <library>FBCTools_BMI</library>
    <workingDir>rtc</workingDir>
    <inputFile>.</inputFile>
  </component>
  <component name="1d2d">
    <library>flow1d2d</library>
    <workingDir>1d2dcoupler</workingDir>
    <inputFile>1d2d.ini</inputFile>
  </component>
  <coupler name="rtc_to_1d2d">
    <sourceComponent>real-time control</sourceComponent>
    <targetComponent>1d2d</targetComponent>
    <item>
      <sourceName>output_Langel_zu_Crest level (s)</sourceName>
      <targetName>water flow 1d/weirs/Langel_zu/structure_crest_level</targetName>
    </item>
    <item>
      <sourceName>output_02_Wehr_Duis~~1_Crest level (s)</sourceName>
      <targetName>water flow 1d/weirs/02_Wehr_Duis~~1/structure_crest_level</targetName>
    </item>
  </coupler>
  <coupler name="1d2d_to_rtc">
    <sourceComponent>1d2d</sourceComponent>
    <targetComponent>real-time control</targetComponent>
    <item>
      <sourceName>water flow 1d/observations/P_Langel/water_level</sourceName>
      <targetName>input_P_Langel_Water level (op)</targetName>
    </item>
    <item>
      <sourceName>water flow 1d/observations/RuhrOWDuisburg/water_level</sourceName>
      <targetName>input_RuhrOWDuisburg_Water level (op)</targetName>
    </item>
  </coupler>
</dimrConfig>
```

Also when the <time> element is present within <control> <parallel> <startGroup> its contents will be replaced with the appropriate values.

FBC component

Will be used if library "FBCTools_BMI" is configured as component within the dimr config file

FBC component

```
  <component name="real-time control">
    <library>FBCTools_BMI</library>
    <workingDir>rtc</workingDir>
    <!-- Look hardcoded for  rtcRuntimeConfig.xml -->
    <inputFile>.</inputFile>
  </component>
```

It will look for the hardcoded file name rtcRuntimeConfig.xml in working directory relative to the parent dir of dimr config file.

It will replace start date and end date in rtcRuntimeConfig.xml with the appropriate values.

It will look for the hardcoded file name timeseries_import.xml in the working dir and insert all timeseries from the time series file configured by "piFileForFbc" (or the first time series xml if there is only 1) by matching headers.

DFlowFM component

Will be used if library "dflowfm" is configured as component within the dimr config file

DFlowFM component

```
  <component name="dflowFM">
    <library>dflowfm</library>
    <workingDir>fm</workingDir>
    <inputFile>mackay.mdu</inputFile>
  </component>
```

It will look for the .mdu file in working directory relative to the parent dir of dimr config file.

It will read "ExtForceFileNew" value.

Find .ext file

```
[external forcing]
ExtForceFile      =
ExtForceFileNew   = FlowFM_bnd.ext
```

It will look for .ext file relative to .mdu file parent directory and extract all "forcingfile" properties from it.

Find .bc files

```
[boundary]
quantity       = waterlevelbnd
locationfile   = mackay_bnd.pli
forcingfile    = mackay_bnd.bc

[boundary]
quantity       = rainfall
locationfile   = mackay_ugrid_wgs84_net.nc
forcingfile    = ..\input\mackay_rain.bc
```

Insert all timeseries from "piFileForFm" (or the first time series xml if there is only 1) into all .bc files (relative to .ext file parent dir) by matching headers.

In the .mdu file itself it will replace Tstart, Tstop and RstInterval with the appropriate values.

RstInterval can be overwritten by a property in the run info file called "restartIntervalForFm"

DFlow1D component

Will be used if library "cf_dll" is configured as component within the dimr config file

Flow1D component

```
  <component name="Hydrodynamics">
    <library>cf_dll</library>
    <workingDir>dflow1d</workingDir>
    <inputFile>Hydrodynamics.md1d</inputFile>
  </component>
```

It will look for the .md1d file in working directory relative to the parent dir of dimr config file.

It will replace StartTime and StopTime with the appropriate values.

It will look for "boundCondFile" to find .bc file (relative to .md1d file parent dir) and insert all timeseries from "piFileForFlow1d" (or the first time series xml if there is only 1) by matching headers.

Flow1D2D component

Will be used if library "flow1d2d" is configured

Flow1D2D component

```
  <component name="1d2d">
    <library>flow1d2d</library>
    <workingDir>1d2dcoupler</workingDir>
    <!-- Will be read to find references to .md1d and. mdu file-->
    <inputFile>1d2d.ini</inputFile>
  </component>
```

It will look for the .ini file in working directory relative to the parent dir of dimr config file.

1D2D ini

```
[Model]
    type                  = Flow1D
    name                  = water flow 1d
    directory             = ..\dflow1d
    modelDefinitionFile   = water flow 1d.md1d

[Model]
    type                  = FlowFM
    name                  = FlowFM
    directory             = ..\dflowfm
    modelDefinitionFile   = FlowFM.mdu
```

It will extract .md1d file (with directory and model definition file relative to parent dir of the .ini file) by looking for model = Flow1D

It will extract .mdu file (with directory and model definition file relative to parent dir of the .ini file) by looking for model = FlowFM

It will process the .md1d and .mdu file the same way as described in DFlow1D and DFlowFM component

D-Waves component

Will be used if library "wave" is configured

Flow1D2D component

```
<component name="1d2d">
      <library>wave</library>
      <workingDir>wave</workingDir>
      <inputFile>stmartin.mdw</inputFile>
</component>
```

It will look for the .mdw file in working directory relative to the parent dir of dimr config file.

At this point the DIMR adapter only performs a conversion of the boundary conditions NetCDF file to ASCII format. The relative path of the netCDF file to convert must be provided in the PI runinfo file using the **dWaveNetcdfBoundaryFileToConvert** property as shown in the example runinfo file above. Two additional properties are provided, to summarize:

- dWaveNetcdfBoundaryFileToConvert = relative path of the boundary conditions netCdf file
- dWaveBoundaryFilePrefix = prefix for the filenames of the output ASCII files, the default prefix is "WW3_"
- dWaveBoundaryFileNumberOfCopies = the number of file copies to provide for each boundary, the default number is 3