Memo

To

Deltares Testbench users

Date	Our reference		Number of pages
2024-03-11	0.1		22
Contact person	**Direct line**	**E-mail**	
Dev-Ops	+31 (0)88 335 8568	black-ops@deltares.nl	

Subject

User documentation DSCTestbench

Copy to

—

Version control information

Location : https://repos.deltares.nl/repos/DSCTestbench/trunk/scripts/doc/readme-testbench-v3.tex

Revision : 31917

Contents

P.O. Box 177, 2600 MH Delft | Boussinesqweg 1, 2629 HV Delft | The Netherlands | T +31(0)88 335 8273 | F +31(0)88 335 8582 | www.deltares.nl

1 Introduction

1.1 Introduction to Testbench

This document is intended to give an overview of the functionalities that are offered by the Testbench software. The Testbench is aimed at supporting quality assurance of Deltares software products, by giving an accessible means of structured, repeatable testing. Specifically, the Testbench software is designed to test the *output* of the software products, rather than testing the software functionally. Testing the output of the products is what we will refer to as *validation*. By offering TeamCity compatible logging, the software can be used for testing nightly builds of the systems under test (SUT).

In this manual we will have 2 different sections depending on the use case. As an user of the testbench software you need to go to the user chapter. Here we will explain how to setup the deltares testbench for you local enviroment. You can then run a testcase from you computer before running test from teamcity. As a developer you can read under the developer chapter what the testbench does, but also: using testcases, setting up configs for testcases and how to setup a teamcity build configuration.

1.2 python

The testbench for D-Hydro is written for python. The new testbench can be run with python and the necessary requirements installed within the requirements.txt. The first thing to do is install python 3 which can be found here: https://www.python.org/downloads/ or within the windows app store by searching python 3. Installing the required python packages will be discussed under the Installation section section 2.1

1.3 General concepts

The nature of the Testbench software can be best described by the following characteristics:

The Testbench tests content Rather than testing whether a certain GUI functionality is working as expected, the focus of the Testbench is on the output of the software. Therefore, it is not suitable for functional testing.

The Testbench tests against a machine-made reference The values to test against are not prepared by hand, but are generated by an earlier version of the SUT. After manual inspection of the results, the outcome is either rejected or accepted. In case the result is accepted by a developer, it will be the *reference* against which future versions will be compared.

The Testbench can tolerate small deviations The fact that many SUTs are in fact solving differential equations numerically, has a negative impact on the result's reproducebility. Floating point operations are not always guaranteed to give identical results on different machines, and therefore, small deviations can occur, which are not considered harmful. Therefore, there is the possibility to set absolute and relative tolerance values.

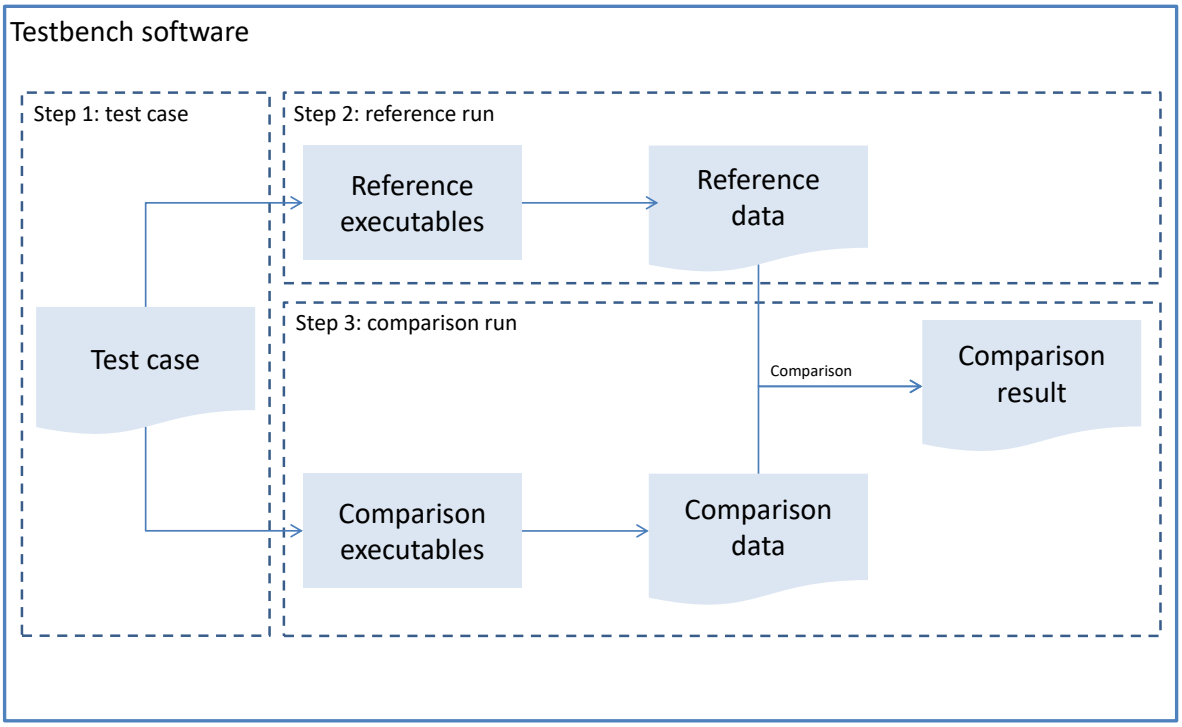

Figure 1: *Conceptual model of the Testbench.*

The conceptual model of the Testbench can be seen in Figure 1. It consists of the following components:

The Testbench software This is the software, written in Python, that runs the executables and test cases, generates reference data and performs the comparison between the reference data and the comparison data.

The executables are the SUTs that are necessary to run the test cases (such as Delta Shell, unstruc, tide, SOBEK, etc.). In addition, the executables also contain tooling that is necessary for pre-processing and post-processing, such as Subversion and data conversion tools (e.g. ncdump, ViewerSelector).

The test cases are the models that are executed by the SUT. In the conceptual model of this version of the Testbench, a test case does *not* include the reference data.

The reference executables are the executables that created the reference data. These are always a *specific version* of the executables, which is being made explicit by Subversion commit numbers or build numbers.

The reference data is the data that resulted from running a reference run. The reference data can contain both 'real' output (such as NetCDF files) and diagnostic/log files. More precisely, the reference data comprises all files that have been modified or added during the execution of the test case by the reference executable.

The comparison executables The version of the executables that generated the comparison data.

The comparison data Identical to the reference data, except that the data was generated by running the comparison executable.

1.4 Comparison vs Reference run

There are two ways to run the testbench.

Comparison A comparison run as the name implies compares results to the reference, this is usefull to see if changes made to the code don't result in changes in results for models. We run these comparisons before merging code to prevent unforseen consequences of changing the code. This prevents possible faulty code to be merged into the main branch. More info here: section 2.5

Reference A reference run is used to create new references to compare against. If a comparison run fails it could be decided that the references are wrong or outdated due to for example new ways of compiling code or changing numeric models. Running in this way allows us to update the references easily from within the testbench and then to be rolled out as new standard to compare against.

2 Users

2.1 Installation

The testbench requires python 3 to be installed. More information can be found here: section 1.2

The first step to using the testbench is to checkout the delft3d repo. The testbench folder is located under:

```
root
└─test
  └─deltares_testbench
```

Open this folder with visual studio code and go to extensions (ctrl + shift + x) and install the python extension.

The next step is to setup a python virtual environment. Using a virtual environment allows us to install the necessary requirements without doing it for all apps that use python. The venv can be installed anywhere. This guide will install it under the user folder: <c:/users/<username>/virtualenvs/> open a command prompt and locate the folder you want to install the venv under. This is our example: `cd c:/users/<username>/virtualenvs/`. Run the command: `python -m venv .testbenchenv` within the specified folder.

Once this is successfully completed we go to visual studio code and open the commands (ctrl + shift + p). Now select the python venv executable in the list that appears after you execute the command: `> Python: Select Interpreter` If it doesn't appear in the list you can manually set the path to the python.exe within the virtual enviroment you created: <c:/users/<username>/virtualenvs/testbenchenv/Scripts/python.exe>.

Note: a common issue is "<file> cannot be loaded because the execution of scripts is disabled on this system." Look here on how to troubleshoot: section 4

Now we need to fill this virtual environment with the required python packages. If the venv is successfully activated your terminal within vscode will show (venv) in front of the terminal. If the venv is successfully activated you can run the command `pip install -r requirements.txt` from within the deltares_testbench folder.

2.2 Run testcase

Once the installation steps are done you can now use the testbench. To run tests we need engines. They can be build locally or you can get them from TeamCity for windows here: https://dpcbuild.deltares.nl/buildConfiguration/Dimr_DimrCollectors_1aDimrCollectorDailyWin64.

Open the build tab artifacts and download the recent main branch dimrset zip file. Extract the contents of the zip file into <data/engines/teamcity_artifacts/> Figure 2.

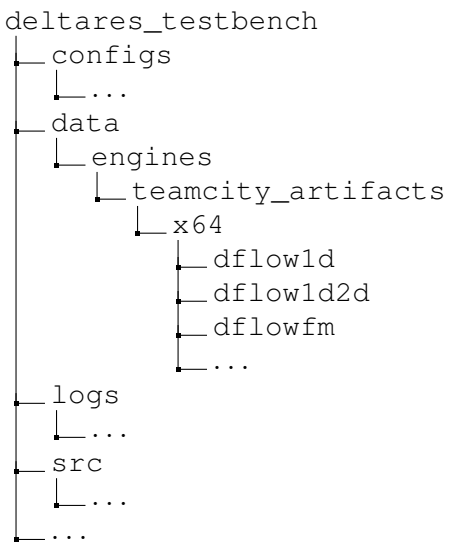

Figure 2: *Layout of the folder tree for teamcity artifacts.*

Running the testbench requires parameters to be supplied on the command-line. There are a few options of running the testbench.

- Use visual studio code and a launch.json that is located within a .vscode folder. More info here: section 2.2.1.
- Run using the provided bat script in the doc folder using standard parameters. More info here: section 2.2.2
- Run from a terminal using the python command and providing parameters to testbench.py. More info here: section 2.2.3

All available command-line parameters and their explanation are located here: section 2.3

2.2.1 Visual studio code

Create a .vscode folder in the deltares_testbench folder and inside it a launch.json with the code in Figure 3. When you press the F5 key now vscode will run testbench.py with the parameters under args. Make sure to replace <your username> with your deltares username credentials to be able to download testcases.

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch testbench",
            "type": "python",
            "request": "launch",
            "program": "Testbench.py",
            "console": "integratedTerminal",
            "args": [
                "--username",
                "<your username>",
                "--compare",
                "--interactive",
                "--config",
                "configs\\dimr\\dimr_dflowfm_win64.xml",
                "--log-level",
                "DEBUG",
                "--parallel"
            ],
            "justMyCode": true
        }
    ]
}
```

Figure 3: *VSCode launch json file for testbench parameters.*

2.2.2 Bat script

The bat file to run the testbench is located here: <doc/example_scripts/run_testbank.bat>. Make sure you run the testbench within the virtual environment. If necessary run activate.ps1.

2.2.3 Command line

It is possible to run the testbench directly from the command-line. Once again make sure you run in the virtual environment setup from the installation section. You can run the testbench script by going into the <delft3d/test/deltares_testbench> folder with the command prompt. After that run the command `python testbench.py --interactive --compare --config <config file> --log-level DEBUG`. Replace <config file> with the config file you wish to run. For example <configs/dimr/dimr_dflowfm_win64.xml>. There are more parameters available. They can be viewed here: section 2.3.

2.3 Testbench parameters

The testbench has a couple of parameters that can be supplied for a run resulting in different behaviour for the testbench script. This section will provide the available parameters and their function.

`--username`	Your username to download testcases.
`--password`	Your password to download testcases. It is recommended to use `--interactive` to supply your password for running the testbench.
`--interactive, -i`	Allows user to supply credentials to testbench by prompt.
`--reference, -r`	Perform a reference run. Mutually exclusive with compare.
`--compare, -c`	Perform a comparison run. Mutually exclusive with reference.
`--config`	Indicates which Testbench configuration file should be used. Default:

`config.xml`.

`--filter` Offers the possibility to filter test cases from the indicated configuration file.
Example:

```
--filter "program=waq,dflowfm:testcase=aaa,bbb:maxruntime=<10:startat=ccc"
    program    (program name   = at least one substring)
    testcase   (test case name = at least one substring)
    maxruntime (test run time  = float, larger, smaller, equals)
    startat    (test case name = substring is the first testcase)
```

`--log-level` Changes the log level of the software. Default: INFO. It is recommended to use INFO. DEBUG is used for debugging with the testbench. There is also ERROR and WARNING as options for the log level.

`--parallel` The testbench will execute one case on each available cpu core.

`--teamcity` Offers Teamcity-compatible logging.

2.4 Testcase repository

The main use of the testbench is to compare results of branches to the latest references of a stable version. This comparison will either succeed or fail and helps to determine whether a change impacted the test models in an unforeseen way. A course of action for the result can then be determined.

These testcases and references are stored in MinIO and the testbench will connect to the object repository to download the cases that are defined within the config that is specified and used by the run. The bucket for testcases and references is https://s3-console.deltares.nl/browser/. In the MinIO object repository the folder structure is setup as follows:Figure 4.

```
dsc-testbench
├── cases
│   ├── doc
│   ├── e01_d3dflow
│   │   └── f01_general
│   │       ├── c12-breukelermeer
│   │       └── c13-markermeer
│   ├── e02_flowfm
│   └── ...
└── references
    ├── lnx64
    └── win64
        ├── e01_d3dflow
        │   └── f01_general
        │       ├── c12-breukelermeer
        │       ├── c13-markermeer
        │       └── ...
        ├── e02_flowfm
        └── ...
```

Figure 4: *Layout of the cases and reference repository in MinIO.*

The configuration will, for example, point to a path to c12-breukelermeer and then the MinIO handler will download the references and cases to the testbench environment. A testcase will have the same sub path in the folder references as cases, so in a config only one path to the testcase is supplied.

The path used locally where the cases and references are downloaded is a bit different. We place everything under the supplied config case name. The folder structure will locally look like this after downloading a few testcases under e01_d3dflow for example: Figure 5

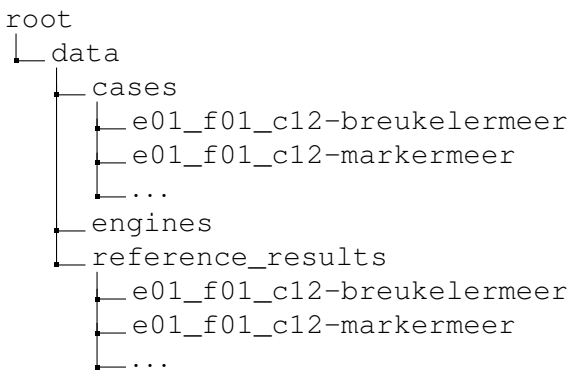

Figure 5: *Layout after some cases have been downloaded by the testbench.*

2.5 A comparison run

A comparison run will test the engines within the engines folder against the stable reference results that are selected by the version tag. This version tag is located in the configuration. If the results are different, it should be determined whether code is faulty or a new reference has to be applied.

A comparison run does 2 things:

- All files in the reference data should also be present in the comparison data (regardless of whether the contents of the files are equal or not).
- In the Testbench configuration file (to be discussed later in this memo), the developer can indicate per test case which parameters should be compared. For instance, the developer might be particularly interested in the waterlevel on specfic locations.
- The version of the file to be downloaded from MinIO by using the rewind feature is specified in the configuration.

3 developers

This section focuses on updating and maintaining the testbench. These things are:

- MinIO and specifically the maintaining of the object repository by updating/adding references and cases.
- XML configurations the layout description, tags, attributes and also maintaining the correct link between the cases and references.
- Teamcity how to setup build configurations for running the testbench and getting the appropriate output and results.

3.1 Testcase storage

The testbench runs a case by using input data stored by MinIO. MinIO can be accessed with an interface by going to https://s3-console.deltares.nl/. The bucket for testcases and references is located under <dsc-testbench>. The layout is described here: Figure 4.

3.1.1 Creating a MinIO access key

To download from MinIO using the testbench you need to create an access key. The MinIO api only accepts access keys to login, your normal credentials don't work. In the console there is a tab for access key: Figure 6, you can also directly open the page using this link:https://s3-console.deltares.nl/access-keys.

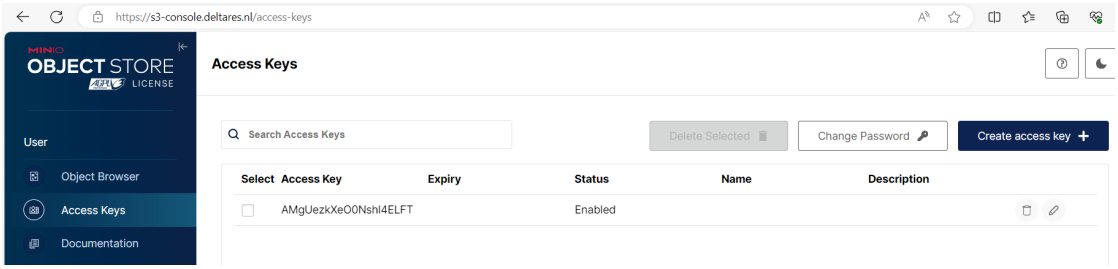

Figure 6: *The MinIO access tab.*

From this page we can create a new key, press the `Create access key +` . This will open a new window; name, command and description are optional. When you press create this will open a popup this will show your access key and secret key: Figure 7. Store these somewhere for example a password database like keepass.
This is the last time it will show you secret key if you lose it you need to create a new key.

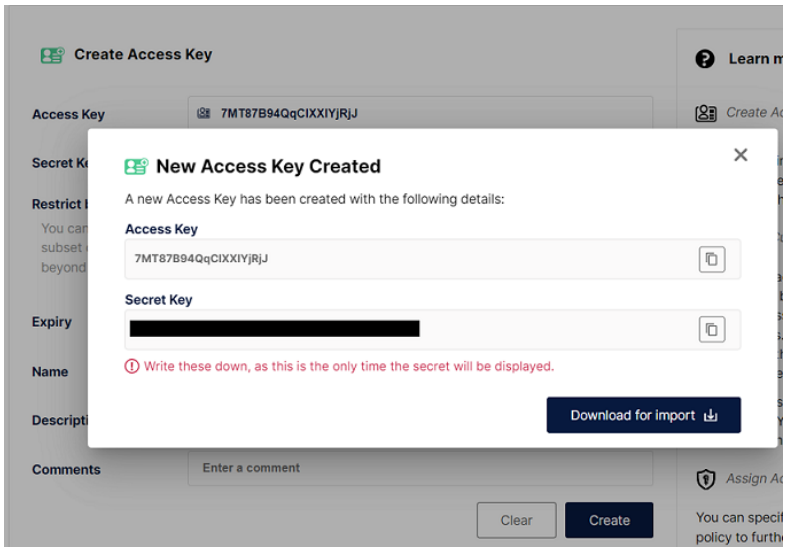

Figure 7: *The MinIO popup for creation of the key.*

Use the access key as your username and the secret key as your password to download cases from the MinIO api.

3.1.2 Updating or adding a case/reference

By going to the MinIO interface you can upload a new testcase or references.

- open the website and relevent folder to update or place new files under.
- upload press the upload button: and choose either file or folder depending on what you want to update. Figure 8
- update the configuration version by using the rewind time on your updated/new files.

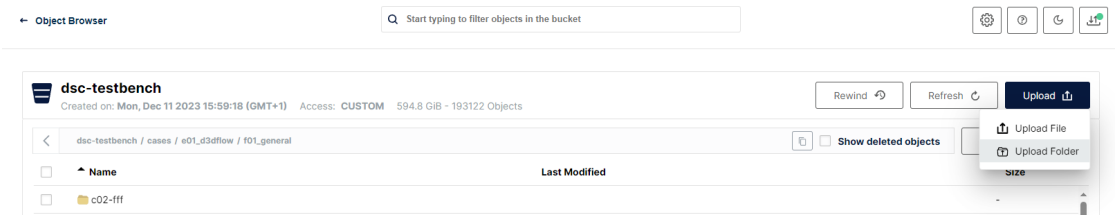

Figure 8: *Upload files to minio.*

3.1.3 Adding a rewind to a test configuration

The MinIO rewind feature works by grabbing the most recent change before the specified rewind date. What we need to check is last modified and use that timestamp with a few seconds to get our correct object(s) Figure 9. This date can then be updated in the xml configuration of which the step is described here: section 3.3

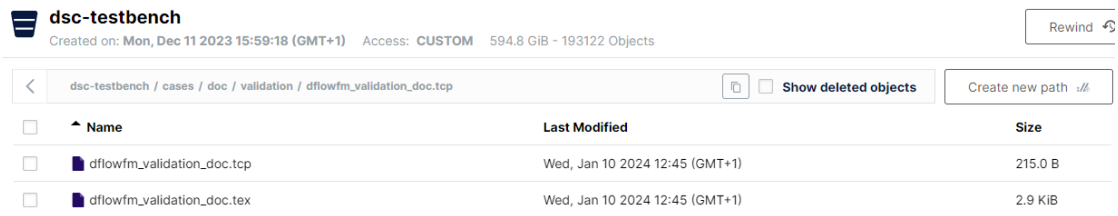

Figure 9: *Last modified for some files stored in MinIO.*

3.2 Xml configuration definition

The testbench runs testcases and compares the data based on input from xml configurations that are stored under the configuration folder. There are 4 main sections within the xml configuration that are defined. Figure 10

```
<?xml version="1.0" encoding="iso-8859-1"?>
<deltaresTestbench_v3 xmlns="http://schemas.deltares.nl/deltaresTestbench_v3"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xi="http://www.w3.org/2001/XInclude"
       xsi:schemaLocation="http://schemas.deltares.nl/deltaresTestbench_v3 http://content.oss.deltares.nl/sc
  <config>
    <!-- config basic configuration defenitions for input and output. -->
  </config>
  <programs>
    <!-- programs and parameters to run testcases -->
  </programs>
  <defaultTestCases>
     <!-- defaultTestCases to link programs to testcases -->
  </defaultTestCases>
  <testCases>
     <!-- testCases to be run. -->
  </testCases>
</deltaresTestbench_v3>
```

Figure 10: *Layout of the top layer of a xml configuration.*

config which contains directory locations and repository urls and credentials. section 3.2.1

programs which describes programs and the parameters to run them using the command line. Also contains the location for the executable and some enviroment variables. section 3.2.2

defaulttestcases is a default template for other testcases to use as a base, contains a link to the program to run as well as case and reference locations. section 3.2.3

testcases describes which testcase to download, path and name the defaulttestcase that it uses for executing and which parameters to check with which file system. section 3.2.4

Including xml files is supported, this is described in detail in https://www.w3.org/TR/xinclude/#include_element The top level of the included file should consist of a single element and the file should not have its own XML header. An XML file should be included by its path with the syntax:

```
<xi:include href="dflowfm\_all\_cases.xml"/>
```

3.2.1 config

Some basic configuration parameters are located under the config section. A basic layout can look like this: Figure 11.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<deltaresTestbench_v3>
  <config>
    <config>
    <localPaths>
      <testCasesDir>.\data\cases</testCasesDir>
      <enginesDir>.\data\reference_engines</enginesDir>
      <referenceDir>.\data\references_results</referenceDir>
    </localPaths>
    <locations>
      <location name="reference_results">
        <credential ref="command-line"/>
        <root>{server_base_url}/references</root>
       </location>
      <location name="cases">
        <credential ref="command-line"/>
        <root>{server_base_url}/references</root>
       </location>
      <location name="local">
         <root>./data/engines</root>
      </location>
    </locations>
  </config>
  </config>
  <programs>
    <!-- programs and parameters to run testcases -->
  </programs>
  <defaultTestCases>
     <!-- defaultTestCases to link programs to testcases -->
  </defaultTestCases>
  <testCases>
     <!-- testCases to be run. -->
  </testCases>
</deltaresTestbench_v3>
```

Figure 11: *Example layout of config section of a xml configuration.*

LocalPaths describe the location that cases and references are downloaded into, and also where the engines are located locally. These are used by the testbench.py directly.

locations describe what credentials will be used and where to download testcase and reference data from. The `{server_base_url}` is injected with the default url defined by testbench.py. The locations are referenced by the defaulttestcases to download from allowing for multiple sources of data. section 3.2.3

3.2.2 programs

This section contains links to the programs to execute a testcase. These are usually located under <data/engines/teamcity_artifacts>. The programs are referenced by defaulttestcases to setup a link for the runner to determine the program to be run and it's parameters. An example layout for this section Figure 12

```
<?xml version="1.0" encoding="iso-8859-1"?>
<deltaresTestbench_v3>
  <config>
    <!-- config basic configuration defenitions for input and output. -->
  </config>
  <programs>
    <!--
    The order is important. Start with the following programs:
    command_prompt
    svn
    mpi
    -->
    <program name="command_prompt">
      <path>cmd</path>
      <arguments>
        <argument>/C</argument>
      </arguments>
    </program>
    <program name="svn">
      <path>thirdparty\svn\win64\svn.exe</path>
    </program>
    <program name="dimr" logOutputToFile="true" programStringRemoveQuotes="true">
      <location ref="reference_engines" type="reference">
        <from>lnx64</from>
      </location>
      <location ref="local" type="check">
        <from>teamcity_artifacts/lnx64</from>
      </location>
        <path>bin/run_dimr.sh</path>
      <environments>
        <environment name="OMP_NUM_THREADS" type="raw">1</environment>
      </environments>
    </program>
  </programs>
  <defaultTestCases>
    <!-- defaultTestCases to link programs to testcases -->
  </defaultTestCases>
  <testCases>
    <!-- testCases to be run. -->
  </testCases>
</deltaresTestbench_v3>
```

Figure 12: *Example layout of a programs section of a xml configuration.*

order is important for the programs. The order should be command_prompt first then svn then mpi and other executables.

program in the list will contain a path towards the specific program the arguments with which to run it and a location from which to locate the program(s). In the case of program dimr the location for the engines is <teamcity_artifacts/lnx64> specified by the location tag using ref=local. This local already has a root location ./data/engines in the example Figure 11 and those two paths are joined resulting in <./data/engines/teamcity_artifacts/lnx64>. The specific program to run is pointed to by the attribute path <bin/run_dimr.sh>. A program is referenced by defaulttestcases to be used by a testcase to execute with this program.

arguments contains a list of arguments to pass with the command to run the program with. For example running <bin/mpiexec.exe> with multithreads you can add the argument `-n 3`.

enviroments You can pass enviroment variables with the configuration for execution like `OMP_NUM_THREADS`.

3.2.3 defaultTestCases

DefaultTestCases are used as a way to set common settings for testcases. This template can be used by testcases that use these settings.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<deltaresTestbench_v3>
  <config>
    <!-- config basic configuration defenitions for input and output. -->
  </config>
  <programs>
    <!-- programs and parameters to run testcases -->
  </programs>
  <defaultTestCases>
    <testCase name="dflowfm_default">
      <programs>
        <program ref="dimr">
          <arguments>
            <argument>-m dimr.xml</argument>
          </arguments>
        </program>
      </programs>
      <location ref="reference_results" type="reference">
        <from>win64</from>
      </location>
      <location ref="cases" type="input">
        <from>.</from>
      </location>
      <maxRunTime>60.0</maxRunTime>
    </testCase>
    <testCase name="dflowfm_config">
      <programs>
        <program ref="dimr">
          <arguments>
            <argument>-m dimr_config.xml</argument>
          </arguments>
        </program>
      </programs>
      <location ref="reference_results" type="reference">
        <from>win64</from>
      </location>
      <location ref="cases" type="input">
        <from>.</from>
      </location>
      <maxRunTime>60.0</maxRunTime>
    </testCase>
  </defaultTestCases>
  <testCases>
    <!-- testCases to be run. -->
  </testCases>
</deltaresTestbench_v3>
```

Figure 13: *Example layout of a defaultTestCases section of a xml configuration.*

programs a link to programs to determine which program to run when referenced to this defaultTestCase. An argument can be passed for the command-line from here too. In this example script we use the same program, but add a differenet argument for the defaultTestCase to run. A program that needs to run a testcase needs to be reachable with a defaultTestCase, this means that 3 programs need 3 defaultTestCases.

arguments additional arguments for running the program, that can be differentiated by using a different defaultTestCase instead of needing a different program too.

location a reference to the location to save results of a run and a reference to locate the testcase to be run. The type determines what it is used for by the testbench.py.

maxRunTime the total time in seconds that cases linked to this defaultTestCase can run for. When this time is exceeded it will be shutdown by the testbench. This can be overwritten by the testcase itself.

3.2.4 testCases

This tag contains all the testcases to be run by the xml configuration. It describes the testcase folder to be downloaded and from where to execute it locally. You can pass additional arguments and reference to programs from within the testcase, resulting in multiple engines being run on the testcase. An example of a testcase section: Figure 14

```
<?xml version="1.0" encoding="iso-8859-1"?>
<deltaresTestbench_v3>
  <config>
    <!-- config basic configuration defenitions for input and output. -->
  </config>
  <programs>
    <!-- programs and parameters to run testcases -->
  </programs>
  <defaultTestCases>
     <!-- defaultTestCases to link programs to testcases -->
  </defaultTestCases>
  <testCases>
    <testCase name="e02_f001_c220_drypoints_pol" ref="dflowfm_default">
    <dependency local_dir="e05_f03_zlayers_hydro" version="2024.01.18T10:00">e05_part/f03_zlayers/hydro</depe
      <path version="2024.01.15T09:00">e02_dflowfm/f001_general/c220_drypoints_pol</path>
      <programs>
        <program ref="DFlowFM">
          <arguments>
            <argument>norway.mdu</argument>
            <argument>--autostartstop</argument>
            <argument>--nodisplay</argument>
          </arguments>
        </program>
      </programs>
      <maxRunTime>15.0</maxRunTime>
      <checks>
        <file name="dflowfmoutput/norway_map.nc" type="netCDF">
          <parameters>
            <parameter name="mesh2d_s1" toleranceAbsolute="0.0001" />
            <parameter name="mesh2d_ucmag" toleranceAbsolute="0.0001" />
          </parameters>
        </file>
        <file name="depthgr.xyz" ignore="true" />
      </checks>
      <processCount>1</processCount>
    </testCase>
  </testCases>
</deltaresTestbench_v3>
```

Figure 14: *Example layout of a testcases section of a xml configuration.*

testcase the testcase name that is ran, as well as a reference to defaultTestCases, using the ref attribute linking to the defaultTestCase by name. It can also contain additional programs and include specific arguments for it. This also includes the version to download the appropriate reference and testcase data from MinIO section 3.3

dependency testcase data that is used for processing or running the testcase. This is for tests that use the same input across different tests. These dependencies are downloaded sequantial at the start of the testbench, preventing conflicts from multiple downloads at the same time.

programs additional programs to execute with arguments to run them with.

maxRunTime this overwrites the maxRunTime specified in defaultTestCases for larger testcases that might need more time to finish.

checks the output file is specified here as well as in what format the output is then this gets compared to the reference result to check if the results are different.

parameters determines which values will be compared. A tolerance is provided either absolute or relative and a location to determine where to check these values.

processCount determines the amount of processes used by the program that runs the testcase. When the testbench executes testcases in parallel this prevents a run: that uses 4 cores to be run with 3 other testcases that use 1 or more cores at the same time on those same cores.

3.3 Updating version attribute

After updating a reference or testcase in MinIO the xml configuration for this testcase needs to be updated too. This is relatively easy

- update reference and case data by following these steps: section 3.1.2.
- timestamp: get the last modified timestamp from MinIO, described here: section 3.1.3.
- format the timestamp to be used within the xml configuration. In the example picture Figure 9 the last modified dat is Wed, Jan 10 2024 12:45 (GMT+1). This needs to be formatted to fit YYYY.MM.DDTHH:MM using this definition we get: 2024.01.10T12:45. MinIO rewind grabs the most recent modified from before the timestamp we provide it.
- apply this timestamp to the xml configuration for the specific testcase example:

```
<testCase name="e02_f001_c220_drypoints_pol" ref="dflowfm_default" version="2024.01.15T09:00">
```

3.4 XML tags and attributes definition

The testbench accepts various parameters, most of which can be viewed in the examples under this chapter section 3.2. This is a full list of tags and attributes that the testbench recongnizes and uses. Take note this list is in order and multiple tags can be used under other tags serving different functions, for example: defining and referencing.

Table 1: *Description of the XML-file*

Tags	Attribute	Desription
<*config*>		wrapper for configuration tags
<localPaths>		Wrapper for location for local directories
<testCaseDir>		defines local path to testcases
<enginesDir>		defines local path to engines
<referenceDir>		defines local path to references
</localPaths>		
<locations>		wrapper for locations of source data
<location>	name	to be referenced by a program for downloading data
<credential>	ref	references which credentials to be used, command-line is used for user input with testbench.py
<root>		The url for checkout or local location of source files
</location>		
</locations>		
</*config*>		

Table 1: *Description of the XML-file*

Tags	Attribute	Desription
<*programs*>		wrapper for program definitions
<program>	name	name of the program to be referenced by testcases
	logOutputToFile	log output to file: true or false
	programStringRemoveQuotes	program remove quotes statement: true or false
	ignoreStandardError	ignore standard error: true or false
	ignoreReturnValue	ignore return value: true or false
	addSearchPaths	add search path to the program for running with the command-line: true or false
	shellStringRemoveQuotes	shell remove quotes statement: true or false
	seq	sequence group identifier to pass to program
	delay	delay before starting the program in seconds
<path>		path to the program to run
<workingDirectory>		the working directory
<arguments>		wrapper for all arguments to be used
<argument>		an argument to pass with executing the program
</arguments>		
<shell>		reference to a shell program that is declared earlier
<location>	name	the name of the location used for referencing to it
	ref	reference to another location as base
	type	the type of location: input, reference or check
<from>		the location to directory if it has a ref root is joined with from to root/from
</location>		
<modules>		the modules for the environment
<module>		a single module for the environment
</modules>		
<environments>		wrapper for environment variables for the program
<environment>	name	name of the operating environment
	type	the type of environment variable
</program>		

Table 1: *Description of the XML-file*

Tags	**Attribute**	**Desription**
</*programs*>		
<*defaultTestCases*>		wrapper for default values for test-cases
<testcase>	name	referenced by testcases to run with these parameters
<programs>		wrapper for references to programs
<program>	ref	reference to program
<arguments>		wrapper for arguments
<argument>		additional argument for the program to run with
</arguments>		
</program>		
</programs>		
<location>	ref	reference to a location
	type	type of location: input or reference
<from>		the location todirectory if it has a ref root is joined with from to root/from
</location>		
<maxRunTime>		the maximum runtime the testcase can run for
</testcase>		
</*defaultTestCases*>		

Table 1: *Description of the XML-file*

Tags	Attribute	Desription
<*testCases*>		wrapper for testcases to be executed
<testCase>	name	name of the testcase that is executed
	ref	reference to a default testcase
<dependency>	local_dir	Location of testcase data that is a dependency for this testcase
	version	the version tag used by MinIO to determine the correct version for the dependency.
<path>		path to the testcase in the object repository to download
	version	the version tag used by MinIO to determine the version of the object to download.
<programs>		wrapper for additional programs to execute with the testcase
<program>	ref	reference for additional program
<arguments>		arguments to execute with the program
<argument>		argument for the program that is executed
</arguments>		
</program>		
</programs>		
<checks>		wrapper for comparing files and variables from the output of a run
<file>	name	path to the file to use for comparison
	type	the type of output file to be compared with: ascii, nefis, his, netcdf, numbertext, dseriesregression, dseriesverification, timeseries_pi timerseries_csv
<parameters>		wrapper for paramaters to check with this file
<parameter>	name	name of the parameter to compare with
	location	Location where the parameter needs to be examined (Case sensitive)
	toleranceAbsolute	sets the absolute tolerance
	toleranceRelative	sets the relative tolerance
</parameters>		
</file>		
</checks>		

Table 1: *Description of the XML-file*

Tags	Attribute	Desription
<maxRunTime>		the maximum runtime the testcase can run for overwrite defaultestcase maxRunTime
<processCount>		the amount of cpu cores that are used by the testcase for parallel calculations, this is important when running the testbench in parallel
</testCase>		
</*testCases*>		

3.5 teamcity

Running the testbench on teamcity is quite easy. To run it python needs to be installed on the agent and then you execute it with a command and the appropriate arguments. To get correct teamcity logging in a way that teamcity understand which test we run and what the results, use the argument `--teamcity`. Credentials need to be passed to access the object repository within teamcity, that is then passed with `--username <username>` and `--password <password>`.

4 troubleshooting

Common problems and their solutions:

Credentials missing To download testcase and reference data from MinIO credentials need to be passed on the command line. The error can mean two things either you didn't provide the credentials or they are incorrect. Make sure your credentials are correct and you have access to view the dsc-testbench bucket.

"<Activate.ps1 cannot be loaded because the execution of scripts is disabled on this system." If the activate command for the python virtual enviroment generates the error that you are not allowed to execute a script, then you need to change the PowerShell execution policy to allow scripts to run. You do this by opening powershell as adminstrator and execute the command: `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned`

Issues running configuration You can validate your xml with the validation schema within <configs/xsd/deltaresTestbench.xsd>.

File missing log message If a result file is missing the most common cause is that your engine didn't execute the testcase. Troubleshooting starts with logs in the testcase folder.