Memo

To

Deltares MinIO tool users

Date	Our reference		Number of pages
2024-08-07	0.1		9
Contact person	**Direct line**	**E-mail**	
Dev-Ops		black-ops@deltares.nl	

Subject

How to upload data to MinIO using the MinIO tool

Copy to

—

Version control information

Location : https://git.deltares.nl/oss/delft3d/-/blob/main/test/deltares_testbench/tools/minio/doc/minio-tool.tex

Revision : 31917

Contents

1 Introduction

The 'Minio Tool' is a command line program used to update case and reference data in MinIO. The source code is hosted within the Delft3d git repository.

2 Preliminaries: The testbench and config files

The deltares testbench is a program that runs arbitrary programs on the input data of (one or more) test cases and compares the output of those programs to the reference data. The testbench requires a configuration file (a testbench config file), which tells the testbench:

- What test cases to run.
- What programs to run for each test case.
- Where to find the 'case' and 'reference' data.

The 'case' data of a test case is the input data. The 'reference' data contains the expected output of the test case. The actual output of the programs under test may deviate from the reference output, but only up to some preconfigured tolerance value. The case/reference data can be quite large. Over 100 megabytes for a single test case is not that uncommon. The case and reference data is to big to be stored with the delft3d source code in git. Therefore this data is stored in the MinIO artifact repository hosted by deltares. MinIO has the added benefit of versioning data by timestamp and md5 hashing together with being made for large file storage. The MinIO tool was created to simplify managing the data in the MinIO artifact repository. The testbench config files now support test case data versioning by supplying a version timestamp to each test case. The configuration of a single test case looks like this:

```
<testCase name="e02_f003_c041_thacker1d_squares" ref="dflowfm_default">
      <path version="2024-01-11T16:30:00">e02_dflowfm/f003_advection/c041_thacker1d_squares</path>
      <maxRunTime>15000.0000000</maxRunTime>
      <checks>
            <file name="dflowfmoutput/planar1dsq_his.nc" type="netCDF">
                  <parameters>
                        <parameter name="waterlevel" toleranceAbsolute="0.0001" />
                        <parameter name="x_velocity" toleranceAbsolute="0.0001" />
                        <parameter name="y_velocity" toleranceAbsolute="0.001" />
                  </parameters>
            </file>
            <file name="dflowfmoutput/planar1dsq_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" type="xyz" ignore="true" />
      </checks>
</testCase>
```

This is configuration for the test case `e02_f003_c041_thacker1d_squares`. Its data is stored in MinIO somewhere under the path `e02_dflowfm/f003_advection/c041_thacker1d_squares`. And the version timestamp of this data is `2024-01-11T16:30:00`. The version timestamp tells the testbench to download the case and reference data from MinIO as it was at that date and time. Using data versioning has benefits: Even when someone changes or removes the test case data, the testbench is still able to download the old data. The testbench is able to roll back the data to any point of time in the past. The downside is that whenever the case or reference data needs to be updated, two things need to happen:

- Someone needs to upload the new data to MinIO.
- The version timestamp needs to be updated.

If the version timestamp is not updated. The testbench will keep downloading the old data (by

using the old version timestamp). Doing these two steps manually turns out to be quite cumbersome. The MinIO tool is designed to do this job for you.

There is a large collection of testbench config files in the delft3d git repository. Some of these config files are used to run the pre-merge tests in TeamCity for the software hosted in the delft3d repository itself. The MinIO tool is built to work with the config files in the delft3d repository. The tool should make it easier to upload data for new test cases, and to update the data for existing test cases.

3 Installing the MinIO tool

The Minio tool is a python program that's bundled with the deltares testbench. It is assumed you have the delft3d git repository checked out. You can make sure you have the latest version of the source code by doing a 'git pull' on the 'main' branch. In addition, the MinIO tool requires some python libraries to work properly. Since the tool is bundled with the deltares testbench code, and shares the same dependencies, you may already have all the required libraries installed if you've installed the testbench dependencies recently. If not, or if you want to ensure you have the latest versions, there are installation steps for both linux and windows that you can follow here.

4 Authenticating to MinIO

MinIO is an 'object storage' service hosted within Deltares. The case and reference data for all test cases is stored in the dsc-testbench bucket. You can browse the data in the bucket here. You will need permission to access the bucket to use the testbench and the MinIO tool. To run the testbench you need 'read' access to the bucket. To upload new data or update data in MinIO, you need 'write' access to the bucket. Either way, you will need to create access keys and store them on your computer to use either the testbench or the MinIO tool. If you know you have the required permissions on the dsc-testbench bucket, you can go to this page to create a key.

After creating a key, you will get two pieces of information:

- An access key id
- A secret access key

You will not see the 'secret access key' anymore. So if you lose it, you will need to create a new key.

Both the testbench and the Minio tool will look for your key in a specific file on your computer (in your home directory):

- On Windows: `C:\Users\<your username>\.aws\credentials`
- On Linux: `/home/<your username>/.aws/credentials`

Note: If these files are not on your computer, and you run the testbench or the MinIO tool anyway, they should print out instructions on how to install the keys.

The content of the credentials file should have the following format:

```
[default]
aws_access_key_id = <your_minio_access_key_id>
aws_secret_access_key = <your_minio_secret_access_key>
```

5 Using the MinIO tool

To use the `minio` tool, navigate to the `'<delft3d_repo_root>/test/deltares_testbench'` directory. This directory will be your working directory throughout this guide. If you've installed the dependencies of the deltares testbench in a virtual environment, make sure you have your virtual environment activated. Try the following command to see if you can run the tool:

```
python -m tools.minio --help
```

You should get a list of subcommands you can use. There are three (at the time of writing):

- pull
- push
- update-references

5.1 Pull

The pull command downloads either the case or reference data from MinIO and stores them as files on your computer. Examples:
Print command line help

```
python -m tools.minio pull --help
```

Pull the 'case' data from test case 'e999_f42_c01_case1' in the config 'path/to/config.xml' to your computer. By default, the data will be stored in the directory configured in the 'config.xml' file. Usually this is'./data/cases'. The same location is used by the testbench to store case data.

```
python -m tools.minio pull --case --config path/to/config.xml
--test-case-name e999_f42_c01
```

Similar to the last command, except this time the 'reference' data is downloaded instead. Also, where possible, the 'short-form' command line options are used.

```
python -m tools.minio pull --reference -c path/to/config.xml -n
e999_f42_c02
```

Pull the 'reference' data from MinIO as it was on April first 2024 at midnight (timezone UTC/GMT)

```
python -m tools.minio pull --reference -c path/to/config.xml -n
e999_f42_c03 --timestamp 2024-04-01T00:00
```

Pull the latest 'case' data from MinIO. By default, the timestamp in the config is used to get a snapshot of the data.

```
python -m tools.minio pull --case -c path/to/config.xml -n
e999_f42_c04 --latest
```

Pull the 'case' data to the directory './foo/bar' instead of the default location.

```
python -m tools.minio pull --case --config path/to/config.xml
-n e999_f42_c02 --local-path ./foo/bar
```

Note:

- The 'config', 'test-case-name' and one of 'case' or 'reference' command line arguments are mandatory. All the other arguments are optional. Use the 'help' argument for a complete list.
- The MinIO tool uses the config file to look up the test case. The 'test case name' you pass as a command line argument does not have to be an exact match, as long as it matches exactly one test case in the config. At the time of writing, the MinIO tool only works on one test case at a time.

- If the matching test case in the config has a ’version timestamp’, the MinIO tool will pull the data from that timestamp. If you want to download the data from a specific time, or if you want to download the latest data instead, you can use the ’timestamp’ or ’latest’ command line arguments, respectively.

5.2 Push

The push command is used to create or update case or reference data in MinIO. This requires users to prepare a directory on their computer with the files they want to have uploaded to MinIO. The MinIO tool uses a compare and synchronize approach to update the data in MinIO. It will compare the contents of the files in the local directory to the test case data in MinIO and it reports the differences. There are four possibilities:

- A file exists in the local directory but not in MinIO. So the file is new and must be created.
- A file does not exist in the local directory but it does exist in MinIO. So the file must be removed.
- A file exists both in the local directory and in MinIO. They have the same content. Nothing happens.
- A file exists both in the local directory and in MinIO. The contents of the files are different. The file must be updated.

Because the data in MinIO is versioned, updates and removals of files is not destructive. New versions (or delete markers) are created in MinIO, and it’s always possible to roll back to older versions of individual files. Nevertheless, you should be careful not to upload unnecessary files, or to accidentally remove files that are necessary. By default, the MinIO tool will only include file ’updates’ in its plans. So this means any file creations or removals will be skipped. Sometimes it may be necessary to create new files, or to remove unnecessary ones. There is a command line option that needs to be added to explicitly allow these operations. After parsing the arguments provided, the tool will output the resulting changes. Normal execution will prompt you with a yes or no to continue with the interpreted instructions. Answering no to the prompt will result in the tool execution being aborted.

The MinIO tool will not only upload the changed files to MinIO. It will also update the version timestamp in the config for the affected test case. This will ensure that the next time the testbench runs, it will download the new data that you’ve just uploaded, instead of the old data. This is usually a one-line change to the config XML file, replacing the ’version’ attribute of the ’path’ element inside the test case. The MinIO tool will show you a diff of the changes to the config. It will prompt you to accept or discard the changes to the config file as well. Note that the config files are tracked by git, so changes made to the config files need to be staged, commited and pushed for them to become visible to others.

The following figure shows a complete session of a ’push’ command. Note that MinIO tool first shows users a plan of what it is about to do, and prompts the user before it changes anything:

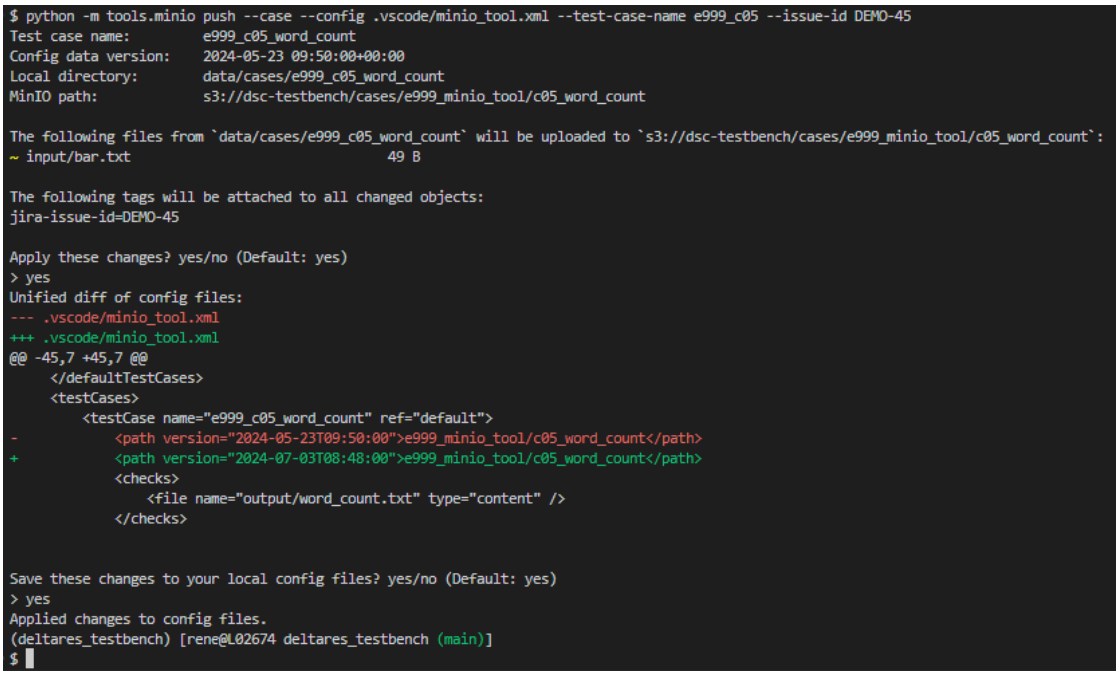

Figure 1: *A complete session of a 'push' command using MinIO tool.*

Examples:
Print command line help menu for the 'push' command.

```
python -m tools.minio push --help
```

Compare the local directory (configured in the config file, usually under './data/cases/') to the data in MinIO. Plan only 'updates' of files that already exist in MinIO (default behavior). If there are changed files, attach the JIRA issue id 'JIRA-123' to the updated files.

```
python -m tools.minio push --case --config path/to/config.xml
--test-case-name e999_f42_c01 --issue-id JIRA-123
```

Similar to the last command, except the 'reference' data is updated. Also, where possible, the 'short-form' command line options are used.

```
python -m tools.minio push --reference -c path/to/config.xml -n
e999_f42_c02 --issue-id JIRA-123
```

Override the configured 'default' local path (usually './data/cases/<test_case_name>') with './foo/bar'

```
python -m tools.minio push --reference -c path/to/config.xml -n
e999_f42_c03 --local-path ./foo/bar --issue-id JIRA-123
```

Explicitly allow 'create' and 'remove' operations in MinIO. Useful for adding new test cases, or cleaning up unnecessary files.

```
python -m tools.minio push --case -c path/to/config.xml -n
e999_f42_c04 --allow-create-and-delete --issue-id JIRA-123
```

Note:

- The 'config', 'test-case-name', one of 'case' or 'reference' and 'issue-id' command line arguments are mandatory. All the other arguments are optional. Use the 'help' argument for a complete list.
- Like 'pull', at the time of writing, the MinIO tool only works on one test case at a time. The 'test-case-name' does not have to be an exact match, but it must match exactly one test case in the given config file.
- The 'issue-id' is mandatory. Every time you need to update the data in MinIO, you update

the timestamp in the testbench config as well. These config files are tracked by git, so every change should be accompanied by a merge request in gitlab and an issue id in JIRA. The issue id is added to the updated files in MinIO as an 'object tag'. This tag is useful to keep track of why a certain file has been changed. The Jira issue can then be looked up to see who was responsible for changing the files in MinIO and why.
- You may be wondering what happens when multiple users are changing the data for a single test case at the same time (shortly after eachother). The MinIO tool always uses the 'version timestamp' for the test case in the config file to verify that there is no data uploaded after that date and time. It may happen that the 'version timestamp' is outdated, and changes have been made to the data by someone else that's working on the same test case. When this happens the MinIO tool will report this as a 'conflict'. When the MinIO tool reports these conflicts, the standard procedure should be to abort the operation, and to use the 'issue-id' tag of the conflicting file to find out who is responsible for changing the test case data.

5.3 Update-references

The update-references command is very similar to the push command. These are the most important differences:
- It is not necessary to pass the '--case' or '--reference' command line option because update-references only ever updates the reference data of a test case.
- The 'reference' data in MinIO is compared to the local 'case' data of the test case.
- Only file updates are allowed. It is not possible to create or remove files in MinIO using this command.

You may wonder why it is useful to compare the local 'case' data to the 'reference' data in MinIO? This command was added to support a common workflow that developers were already used to when updating test case references. The workflow roughly goes like this:
- Download the latest copy of the case data and store it in a local directory
- Run the program on the case data. The program is usually a newer version of the program, suitable to produce the new reference data.
- The program produces output files in the same local directory where the case data is stored (perhaps in an 'output' directory). The local directory now contains a mix of the case data files and 'output' data files.
- Some of the 'output' files are selected as the new 'reference' data.

The testbench can be used for the first three steps of the workflow above. The last step may require some manual steps to verify that the output files are suitable to be used as reference data. The idea behind the 'update-references' command is that updating the references very often boils down to uploading new versions of the same output files. Files are rarely created and removed from the references. Use 'update-references' only when references for an existing test case need to be updated. If you need to create new files or remove existing files from the references in MinIO, use 'push' instead. Examples:

Print help message for update-references command.
```
python -m tools.minio update-references --help
```

Compares the local 'case' directory to the 'references' in MinIO.
```
python -m tools.minio update-references --config
path/to/config.xml --test-case-name e999_f42_c01 --issue-id
JIRA-123
```

Override local directory to use './foo/bar' instead of the default 'case' data directory.
```
python -m tools.minio update-references -c path/to/config.xml
-n e999_f42_c01 --local-path ./foo/bar --issue-id JIRA-123
```

6 Dealing with conflicts

While working with the MinIO tool, you may run into ’conflicts’. Conflicts can happen while using any of the commands ’pull’, ’push’ or ’update-references’. The MinIO tool can detect the conflicts by comparing the ’version timestamp’ of the test case in the config with the upload timestamp of the files in MinIO. The MinIO tool expects the version timestamp in the config to be ahead of upload timestamps of all of the files in MinIO (belonging to the test case). If any of the files in MinIO has a timestamp later than the version timestamp in the config, the MinIO tool reports it as a conflict.

Conflicts indicate that some other person has updated the data in MinIO. This other person may also have updated the version timestamp in the config, but these changes are not yet reflected in your branch. If these changes have been merged to main already, then your branch is stale and you can solve this yourself by either rebasing your branch on the main branch (or merging the main branch in your branch). If not, then there are multiple people updating the data of a single test case at the same time, and you should discuss who merges their changes first.

The ’--issue-id’ command line argument has been made mandatory to make it easier to find out who has updated the test case data and why. The tool has no way of verifying that the Jira issue id actually exists in our issue tracker. Please always enter a valid Jira issue id. Updating the case or reference data should always be a task that has a corresponding issue in Jira.

The following figure shows an example of a situation where there are conflicts. Notice that the MinIO tool shows the Jira issue id of each conflicting file.

```
$ python -m tools.minio push --case --config .vscode/minio_tool.xml --test-case-name e999_c05 --issue-id DEMO-45
Test case name:         e999_c05_word_count
Config data version:    2024-05-23 08:04:00+00:00
Local directory:        data/cases/e999_c05_word_count
MinIO path:             s3://dsc-testbench/cases/e999_minio_tool/c05_word_count

Conflicts detected. The following changes have occurred since 2024-05-23 08:04:00+00:00:

  File name                                 Size     Last modified                JIRA issue
- input/foo.txt                                      2024-05-23 09:45:11+00:00
+ input/baz.txt                              28 B    2024-05-23 09:45:11+00:00    DEMO-42
~ input/bar.txt                              45 B    2024-05-23 09:45:11+00:00    DEMO-42

Continue anyway? yes/no (Default: no)
>
```

Figure 2: *Example of a situation where there are conflicts.*

7 Miscellaneous

7.1 Updating multiple test cases

At the time of writing, you can only pass a single config file and a single test case name to the MinIO tool. Meaning the tool is limited to working on a single test case at a time. Unfortunately, there may be situations where case or reference data has to be updated for many different test cases. The current workaround is to collect all of the configs and test cases that need to be updated in a list and invoke the MinIO tool once for every test case in a script. The MinIO tool is in ’interactive mode’ by default, meaning that it will show what it is about to do before prompting users to continue or abort. The MinIO tool also supports a ’batch mode’ where it skips the user interaction and uses the ’default’ prompt option every time.

When using batch mode, please be careful that the data you're uploading is prepared correctly so you don't have to clean it up later. Here's an example bash script that illustrates it:

```
#!/bin/bash

set -eo pipefail

# Long list.
TEST_CASES=( \
"e999_f42_c01" "e999_f42_c02" "e999_f42_c03" \
"e999_f42_c05" \
)

for arg in "${TEST_CASES[@]}"; do
python -m tools.minio update-references -c path/to/config.xml \
-n "${arg}" --issue-id JIRA-123 --batch
done
```

When conflicts are detected in batch mode, the default option is to abort the operation. You can combine the `--batch` flag with the `--force` flag to force the MinIO tool to continue anyway after a conflict. But please make sure you're not interfering with another person's work when you're doing this.

7.2 Terminal colors

By default, the MinIO tool uses ANSI terminal color codes to add color to the output of the tool. Powershell and most Linux graphical terminals support rendering colored text. It seems Windows `cmd.exe` terminal does not support it by default, and the color codes will be printed to the output verbatim. If you want to turn off the colors, add the `--no-color` command line switch.

7.3 Dealing with testbench config files with 'includes'

There are many testbench config files that contain an <include/> XML element, which contains a relative path to a different XML file. This tells the XML parser to find the specified file and include it verbatim, replacing the include tag. The MinIO tool is aware of these includes, and it is able to find and update the version timestamps of test cases that are found in included files. Sometimes, two different testbench config files (say, `config_lnx.xml` and `config_win.xml`) both have an include element pointing to the same XML file that contains some test cases (say `test_cases.xml`). After updating the version timestamp of a test case in the included file (`test_cases.xml`), the test case is updated in both testbench configs that included that file (`config_lnx.xml` and `config_win.xml`). This is confusing, because the location of the case/reference data is sometimes determined in the testbench config (not the included file). In essence, the data for the test case is stored in different places, but they share the same version timestamp. This is mostly used to store separate versions of the test case data for different platforms (Windows and Linux), because the data may be slightly different. Usually, when test case data needs to be updated, both the Windows and Linux versions need to be updated anyway so it should not be a big problem. It is just something to keep in mind.