DHI MATLAB Toolbox

User Guide for Version 2021

DHI headquarters
Agern Allé 5
DK-2970 Hørsholm
Denmark

+45 4516 9200 Telephone
+45 4516 9333 Support
+45 4516 9292 Telefax

mikebydhi@dhigroup.com
www.mikebydhi.com

CONTENTS

DHI MATLAB Toolbox
User Guide for Version 2021

1 Introduction

This document constitutes the user guide and documentation for the DHI MATLAB Toolbox, and especially for reading and writing dfs0 files.

For details on working with other file types, consult the DFS user guide and .NET API documentation, included when installing the MIKE SDK

This version of the toolbox only works together with a MIKE by DHI product version 2021 or later.

1.1 Why a DHI MATLAB Toolbox

MATLAB[1] provides a compact high-level technical programming/ scripting language, which allows swift handling of time series data, analysis, visualisation and presentation. The MATLAB environment is very much hands-on and can be used without special programming skills for custom analysis of results from numerical models.

However, data produced or required by DHI Software packages are most often stored in DHI's file formats like DFS (Data File System). DHI MATLAB Toolbox provides tools and examples to read and write different DHI files, and other tasks related to DHI files.

1.2 DFS Support

From version 2011 of the MIKE by DHI software suite there is full support for reading and writing any DFS file from within MATLAB, without the need to install other components. The DHI MATLAB toolbox is strictly not required for working with DFS files, though it provides a number of convenience tools and also a number of examples of reading and writing different DFS files. The support for the `dfsTSO` class has been deprecated from Release 2021. Therefore the Matlab Toolbox support to this class has also been removed.

DFS files are handled by the `DHI.Generic.MikeZero.DFS` component, which is a .NET component installed with MIKE Zero, MIKE URBAN PLUS or MIKE SDK. A user guide, class library documentation and more, is available when installing the MIKE SDK (from Release 2014), and can be found from the Windows start menu:

“MIKE by DHI 2014” – “Mike Zero” – “Documentation” – “MIKE SDK Documentation Index”

For earlier versions of MIKE by DHI the documentation is included with the MIKE Zero and MIKE URBAN installer:

“MIKE by DHI 2011/2012” – “Mike Zero” – “Documentation” – “MIKE Zero Documentation Index” – Look for the "File Formats" section.

[1] Matlab is developed by MathWorks, Inc. A description of Matlab is available at http://www.mathworks.com/products/matlab/description1.html

1.3 User Support

The DHI MATLAB Toolbox is not a part of DHI Software Products and is delivered ‘as is’. Thus the DHI Software Service & Maintenance Agreement (SMA) does not cover support and hotline assistance to this tool.

If you find any bugs, have comments, questions or requests for new features, you are welcome to contact mikebydhi@dhigroup.com - please add ‘DHI MATLAB Toolbox’ in the email subject line.

1.4 Disclaimer

A disclaimer is included in the installation[2] .

[2] DHI Matlab Toolbox Disclaimer.pdf document

2 Installation

2.1 Requirements

MATLAB must be installed.

The 32 bit and the 64 bit version of MATLAB will work with the 32 bit and the 64 bit version of MIKE by DHI respectively, i.e. you must make sure to install either 32 bit version of both or 64 bit version of both. They cannot be mixed. From Mike Release 2017 and forth, Only 64 bits versions exist.

To read and write DFS files other than dfs0, MATLAB must be able to interact with .NET objects. For MIKE by DHI release 2011, MATLAB version R2009a or later is required (tested on R2010a). For MIKE by DHI release 2012 or later, MATLAB version R2011a or later is required, but version R2012b or later is recommended, due to better .NET compatibility.

A From Release 2021, the installation of a MIKE Software such as Mike Zero, Mike SDK or Mike Urban Plus is not necessary to run the DHI Matlab toolbox, as the required binaries can be accessed without installing then into the global assembly cache (GAC) for .NET support and will work in any fairly recent version of MATLAB.

2.2 Installing the DHI MATLAB Toolbox

Download the DHI MATLAB Toolbox (per February 2021):

https://github.com/DHI/DHI-MATLAB-Toolbox

Get the DHIMatlabToolbox_XXXX.zip, where XXXX represents the version number. Unzip its content. Assuming you unzip to the folder:

```
C:\Matlab
```

Then a folder called

```
C:\Matlab\mbin
```

should be created, including a lot files and a couple of subfolders. Note that the `C:\Matlab` folder can be replaced with any other folder, according to user preferences; just replace it in the following.

The `C:\Matlab\mbin` folder should be added to the MATLAB path. Start MATLAB, in the MATLAB command window, issue the command:

```
>> addpath(genpath('C:\Matlab\mbin'));
```

This will add recursively the folder to the MATLAB path for this MATLAB session only. To add the folder permanently to the path, instead add the command to your startup.m file, or use the menu ‘file’ - ‘Set Path’ and add the needed folders there.

You should now be ready to use the DHI MATLAB Toolbox.

2.2.1 Examples

Examples are included with the Toolbox zip, and extracted to:

```
C:\Matlab\Example
```

Change the current directory to this folder, and run any of the `read_xxx.m` or `create_xxx.m` scripts.

There are examples for reading and creating a number of file types. Included are also some small data files.

2.3 Compatibility Issues

From Release 2021 of Mike Products, there is no need to install a Mike products to be able to run the Matlab DHI toolbox.

3 Functionality

Notation: Text from the MATLAB command window is typeset in courier font. Input written at the MATLAB command prompt starts with >>, while remaining lines are output, i.e., looking like the MATLAB command window.

For reading and writing DFS files, check the user guide as described in Section 1.2.

The DHI MATLAB Toolbox consists of a `DFS` class for a fast preview of the content of dfs files.

For handling of Dfs0 files, the compiled binary named MatlabDfsUtil.dll provides more convenient handling of dfs0 files than the pure DHI.Generic.MikeZero.DFS component does. MatlabDfsUtil.dll is compiled based on the DHI.Generic.MikeZero.DFS component, but with a much faster execution.
.

3.1 Examples

With the DHI MATLAB toolbox follows a series of examples that covers the different functionality.

- `read_dfs0` – reading dfs0 files, plotting 4 timeseries.
- `read_dfs1` – reading dfs1 files, animating profile data.
- `read_dfs2` – reading dfs2 files, animating grid data.
- `read_dfs2b` – reading dfs2 files, plotting bathymetry
- `read_dfs3` – reading dfs3 files, making a layered plot.
- `read_dfsu_2D` – examples of how to read and handle triangulated data directly using MATLABs standard plotting routines, and how to use mzPlot,
- `read_dfsu_3D` – examples of how to read and convert 3D input to 2D that can be plotted layer by layer.
- `read_network` – examples of how to read network results, coming from MIKE 11, MOUSE or MIKE 1D.
- `write_dfs1` – example of changing existing profile data.
- `write_dfs2` – example of changing existing grid data.
- `write_dfsu_2D` – example of changing 2D unstructured data.
- `create_dfs0` – how to create a dfs0 time series file.
- `create_dfs1` – how to create a dfs1 profile series file.
- `create_dfs1_noneqspat` – how to create a dfs1 profile series file with non-equidistant spatial axis.
- `create_dfs2` – how to create a dfs2 grid series file.
- `create_dfs3` – how to create a dfs3 3D grid series file.
- `create_dfsu_2D` – how to create a dfsu 2D file from a mesh file.
- `create_dfsu_3Dfrom3D` – how to create a dfsu 3D file from another dfsu 3D file.

3.2 Reading dfs0 files

3.2.1 Making References available to consumption

From Release 2021, the .NET MIKE Powered by DHI (MIKE) binaries are not included in the Windows global assembly cache (GAC). Therefore it is simpler to include any MIKE dll in any Matlab script by including the command NETAssembly(string dll):

```
>> NETaddAssembly('DHI.Generic.MikeZero.DFS.dll');
>> NETaddAssembly('DHI.Generic.MikeZero.EUM.dll');
>> import DHI.Generic.MikeZero.DFS.*;
>> import DHI.Generic.MikeZero.*.*;
```

These lines above are the most necessary lines to work editing dfs files. Now you are ready to start. If you want to enable the Matlab DFSutil, you can do it by typing:

```
>> H = NETaddDfsUtil();
```

3.2.2 Open a file for reading

You open a dfs file by calling the DFSFileFactory Static method DFSGenericOpen method. The file is opened and header information is read. A summary of header information is written to the console, unless hidden with a semicolon at the end of the command:

```
>> dfs0File  = DfsFileFactory.DfsGenericOpen(infile)

 dfs0File =

   DfsFile with properties:

     FileInfo: [1×1 DHI.Generic.MikeZero.DFS.DfsFileInfoWrapper]
     ItemInfo: [1×1
System.Collections.ObjectModel.ReadOnlyCollection<DHI*Generic*Mik
eZero*DFS*IDfsDynamicItemInfo>]
     FileName: [1×1 System.String]
     FileMode: Read
```

To review the header information, you can just enter the object variable at the command prompt

```
>> dfs0File.FileInfo

 ans =

   DfsFileInfoWrapper with properties:

                  FileName: [1×1 System.String]
                 FileTitle: [1×1 System.String]
          ApplicationTitle: [1×1 System.String]
        ApplicationVersion: 0
```

```
                    DataType: 0
    [... remaining output omitted ...]
```

3.2.3 Getting header information / object properties

The get function will return header data. The content will depend on the file being loaded. To view available header data, use the FileInfo property:

```
>> dfs0File.FileInfo
   ans =

   DfsFileInfoWrapper with properties:

                    FileName: [1×1 System.String]
                   FileTitle: [1×1 System.String]
             ApplicationTitle: [1×1 System.String]
          ApplicationVersion: 0
                    DataType: 0
                    FileType: EqtimeFixedspaceAllitems
                   StatsType: NoStat
             DeleteValueFloat: 1.0000e-35
              DeleteValueByte: 0
            DeleteValueDouble: -1.0000e-255
               DeleteValueInt: 2147483647
        DeleteValueUnsignedInt: 2147483647
                  Projection: [1×1
 DHI.Generic.MikeZero.DFS.DfsProjection]
                    TimeAxis: [1×1
 DHI.Generic.MikeZero.DFS.DfsEqCalendarAxis]
                CustomBlocks: [1×1
 System.Collections.Generic.List<DHI*Generic*MikeZero*DFS*IDfsCust
 omBlock>]
             IsFileCompressed: 0
```

If you want a certain part of the header information, it can be put as sub Property (case sensitive).

```
>> dfs0File.FileInfo.Projection.WKTString

  ans =

      NON-UTM
```

3.2.4 Setting header information / object properties

You can update header data using direct assignment into the settable properties.

For instance, if I want to rename the first item of the file you can simply type:

```
>> dfs0File.ItemInfo.Item(0).Name = 'newItem'

dfs0File =

  DfsFile with properties:

    FileInfo: [1×1 DHI.Generic.MikeZero.DFS.DfsFileInfoWrapper]
    ItemInfo: [1×1
System.Collections.ObjectModel.ReadOnlyCollection<DHI*Generic*Mike
Zero*DFS*IDfsDynamicItemInfo>]
    FileName: [1×1 System.String]
    FileMode: Read
```

3.2.5 Show item definitions

You can obtain the number of items by using the object properties:
dfs0File.ItemInfo

ans =

ReadOnlyCollection<DHI*Generic*MikeZero*DFS*IDfsDynamicItemInfo> with properties:

Count: 5

A textual detailed description of each item can be obtained by using its index. For instance if I want to check the last item I would used the index 4 (zero-based):

```
>> dfs0File.ItemInfo.Item(4)

ans =

  DfsDynamicItemInfoWrapper with properties:

    AssociatedStaticItemNumbers: [1×1
System.Collections.ObjectModel.ReadOnlyCollection<System
*Int32>]
                      ValueType: Instantaneous
                       MaxValue: 359.9213
                       MinValue: 0.4069
                     ItemNumber: 5
                           Name: [1×1 System.String]
                       Quantity: [1×1
DHI.Generic.MikeZero.eumQuantity]
                       DataType: Float

[... remaining output omitted ...]
```

The number following the EUM type and unit is the unique integer identifying the type/unit.

3.2.6 Building a dfs0 file from scratch

To create a new dfs (dfs0) file, you need two factors, a Factory and a build objects, which are objects that can create items, and spatial and temporal axis. The builder is used to create items

First you create the factory and the builder:

```
>> factory = DfsFactory();
>> builder = DfsBuilder.Create('Matlab dfs0 file','Matlab DFS',0);
```

We need a map projection to relate the data and a time axis. These are created by the factory object:

```
>> proj = factory.CreateProjectionGeoOrigin('UTM-33',12,54,2.6);
>> date = System.DateTime(2002,2,25,13,45,32);
>> unit = eumUnit.eumUsec;
>> tAxis = factory.CreateTemporalNonEqCalendarAxis(unit, date);
```

Then, you need to specify the type of data you want to use in the file you are creating:

```
>> builder.SetDataType(0);
>> builder.SetGeographicalProjection(proj);
>> builder.SetTemporalAxis(tAxis);
```

The DataType is a required step, for used in complex projects. For our task here is not relevant what argument we give, but it is mandatory call that method. The other two lines set the projection and time axis to the recently created ones.

Then we need to create the quantities to be stores in the time series file:

```
>> eumWl = eumItem.eumIWaterLevel;
>> eumMt = eumUnit.eumUmeter;
>> quantity = DHI.Generic.MikeZero.eumQuantity(eumWl, eumMt);
```

This quantity, which is a water level measured in meters, will be used in the created file. Then we need to create an item and include this quantity in the builder:

```
>> item1 = builder.CreateDynamicItemBuilder();
>> item1.Set('WaterLevel item', quantity, dfsdataType);
>> item1.SetValueType(DataValueType.Instantaneous);
>> item1.SetAxis(factory.CreateAxisEqD0());
```

In the lines above we create the item and set the basic item information. Once we have the item ready, we have to add the item to the file builder:

```
>> builder.AddDynamicItem(item1.GetDynamicItemInfo());
```

Then, we create the file with a single item on it, and get a handle to the file stream:

```
>> builder.CreateFile(filename);
>> dfs = builder.GetFile();
```

Now we have created the file and we could put data on it.

3.2.7 Writing item data

Writing item data follows the reading functionality. We start by creating a file to write on it:

```
% Create the file - make it ready for data
>> builder.CreateFile(filename);
>> dfs = builder.GetFile();
>> % Create time vector: constant time step of 60 seconds here
>> t(:) = 60*(0:numTimes-1)';
```

We proceed to write data on it:

```
% Write to file using the raw .NET API (very slow)
>> dfs.WriteItemTimeStepNext(t(14),NET.convertArray(data(14,2)));
```

The data to write must have the same format as the data returned when reading.

If you are writing a large amount of data into a file, this method will be called inside a double loop (through all items and through all time steps) which is highly time consuming. The alternative is to use the DfsUtil, which will make the writing operations much faster, and in a single line of code (i.e., no need to make a loop):

```
>> netTime = NET.convertArray(times);
>> nbTimes = size(data,1); %number of time steps;
>> nItems = size(data,2);
>> nbType = 'System.Double';
>> netData = NET.convertArray(data, nbType, nbTimes, nItems);
>> mu = MatlabDfsUtil;
>> mu.DfsUtil.WriteDfs0DataDouble(dfs, netTime, netData);
```

This could be written in one line If you type:

```
>> MatlabDfsUtil.DfsUtil.WriteDfs0DataDouble(dfs,
NET.convertArray(times), NET.convertArray(data, 'System.Double',
size(data,1), 1));
```

You can choose which version to use in your scrips, depending on your scripting preferences.

3.2.8 Reading item data

To read Data: Use the DfsUtil for bulk-reading all data and time steps in one line. You need to import it first from the dfs0 library:

```
>> import DHI.Generic.MikeZero.DFS.dfs0.*;
>> dd   = double(Dfs0Util.ReadDfs0DataDouble(dfs0File));
>> t    = dd(:,1);
>> data = dd(:,2:end);
```

So if you want to read a single value, you just get it from the data array. For example, to obtain the value from item 2, timestep 14, use:

```
>> val = data(14, 2)

  val =

    -0.3700
```

You may retrieve more than one timestep at a time as standard Matlab vector access, e.g.,

```
>> val  = data(14:17,2);      % read timestep 14 to 17
>> val2 =  data([4 6 8], 2);  % read timestep 4,6 and 8
>> val3 = data(:, 2);         % read all timesteps
```

You can use the `end` keyword to read the last item or the last timestep, i.e.

```
>> val4 = data(end,end);
```

will read the last timestep of the last item. For more tricks, use the Matlab documentation.

3.2.9 Closing a file

When done working with a file, and you wish to free memory associated with the objects, use the close function.

```
>> dfs0File.Close();
```

Setting `dfs=0` will free associated leftover memory.

3.2.10 Saving file

File data are saved when the `Close` function is issued.

```
>> dfs.Close();
```

3.2.11 Read time information

To read time information for the timesteps of the file, use

```
>> dfs.FileInfo.TimeAxis

   ans =

   DfsNonEqCalendarAxis with properties:

          StartDateTime: [1×1 System.DateTime]
               TimeSpan: 599940
           TimeAxisType: CalendarNonEquidistant
               TimeUnit: eumUsec
```

```
        StartTimeOffset: 0
      NumberOfTimeSteps: 10000
    FirstTimeStepIndex: 0
```

Time steps can be read for items in the following ways:

```
>> t = dfs0.ReadItemTimeStep(3,13)
              % reads item 3, timestep 14
```

Or Using the DfsUtil as shouwn in section 3.2.7.
Note that timestep indices start at 0, i.e., the first timestep has index 0.

3.2.12 Adding timesteps and editing time

To add a timestep, use:

```
>> dfs.WriteItemTimeStepNext(0, NET.convertArray(data1(:)));
```

which will add 1 timestep after the last timestep, incremented with the default timestep interval. Each item will have delete values added to the added timestep.

Equidistant time axis types

You can set the desired calendar time axis type when creating the file, through the builder object, something like:

```
>> factory = DfsFactory();
>> builder = Dfs1Builder.Create('Matlab dfs1 file','Matlab
DFS',0);
>> builder.SetDataType(0);

% Create a temporal definition
>> date = System.DateTime(2002,2,25,13,45,32);
>> timeAxis =
factory.CreateTemporalEqCalendarAxis(eumUnit.eumUsec,date,0,60);
>> builder.SetTemporalAxis(timeAxis);
```

The time axis can be created with one of the factory methods, which are summarized in the following table:

Factory Method	Axis Type created
CreateTemporalEqCalendarAxis	'Equidistant_Calendar'
CreateTemporalEqTimeAxis	'Equidistant_Relative'
CreateTemporalNonEqCalendarAxis	Non_Equidistant_Calendar'
CreateTemporalNonEqTimeAxis	Non_Equidistant_Relative'

To set the time of each timestep, the procedure depends on the time axis type.

Note that changing the time definition affects all items of the file. Item data values for removed timesteps are automatically deleted.

Non-equidistant time axis types

Each timestep time can be set individually after creating the file and having items added:

```
>> tAxis = factory.CreateTemporalNonEqCalendarAxis(unit,D);
>> builder.SetTemporalAxis(timeAxis);
>> builder.CreateFile(filename);
>> item1 = builder.CreateDynamicItemBuilder();
>> eumWl = eumItem.eumIWaterLevel;
>> eumMt = eumUnit.eumUmeter;
```

```
>> quantity = DHI.Generic.MikeZero.eumQuantity(eumWl, eumMt);
>> item1.Set('WaterLevel item', quantity, dfsdataType);
>> item1.SetValueType(DataValueType.Instantaneous);
>> item1.SetAxis(factory.CreateAxisEqD0());
>> builder.AddDynamicItem(item1)
>> dfs = builder.GetFile();
>> dat = NET.convertArray(data(1,1));
>> dfs.WriteItemTimeStepNext(10, dat));
```

The lines above will create a single item and time step in a non equidistant dfs0 file. Check the installed example to see more details on the edition of dfs files.

When setting times for new timesteps, setting a time value bigger than or equal to the next Timestep time value or smaller than or equal to the previous Timestep time value is not possible.

4 Other Tools

As a part of the DHI MATLAB toolbox a number of tools have been implemented in order to help the processing of data. They include:

- Reading and writing mesh files. Reordering and refining meshes.
- Analysing and modify mesh files
- Reading and writing xyz files

Other tools are available for plotting dfsu data, calculating gradients and more. The tools are located in the mbin folder (see the installation section), and have an mz prefix.

4.1 Mesh Files

A mesh file is a flexible mesh, consisting of elements and node data. It also contains information on the coordinate system and the projection used for the node coordinates.

```
>> [Elmts,Nodes,proj] = mzReadMesh(filename)
```

Elmts is the element-node-connectivity table. Nodes consist of 4 columns, the first 3 are x, y, and z coordinates for each node, and the last column is a boundary code, telling which boundary this node belongs to. proj is a text string representing the projection which the coordinates use.

Similar you can write a mesh file to disc using:

```
>> mzWriteMesh(filename,Elmts,Nodes,proj)
```

See help mzReadMesh, help mzWriteMesh for details.

Furthermore, there are tools to reorder and refine meshes, see help `mzReorderMesh` and help `mzRefineMesh` for details.

4.2 Mesh Analyse Tool

The Mesh analyse tool, `mzMeshAnalyse`, analyses a mesh and highlights elements which gives the mesh a "bad" quality. The user can zoom in on the worse element or toggle between the 20 worse elements in the mesh. And the user can edit and modify the mesh in order to improve on the quality.

The quality of the mesh is measured in three different ways:

1. Simulation timestep (dt) based on a CFL condition for the shallow water equations (sqrt(g*h)). Using this measure, the user can decrease simulation runtime. By only editing a few elements, often a significant amount of time is saved.
2. Smallest angle of elements. Elements with small angles give more inaccurate results and should be avoided.
3. Smallest area of elements. Small elements close to larger elements can give more inaccurate results

You can zoom in to the "bad" elements and manually edit the mesh.

1. Collapse a face: Select a face, and the two end-nodes of the face is collapsed to one node at the center of the face.
2. Collapse an element: Select an element, and the element nodes are collapsed to one node at the center of the element.
3. Create a quad from two neighbouring triangles, by deleting the face in between the two.
4. Delete a node: Mesh is updated accordingly.
5. Mode a node
6. Add a node

Type `help mzMeshAnalyse` for further details.

The tool is based on the concepts and initial analysis code presented in Lambkin, D.O. (2007), '*Optimising mesh design in MIKE FM*', DHI website and in: Dix, J.K., Lambkin, D.O. and Cazenave, P.W. (2008) '*Development of a Regional Sediment Mobility Model for Submerged Archaeological Sites*', English Heritage ALSF project 5224.

This report considers many important aspects of mesh creation, and is worth reading before creating your next mesh.

4.3 XYZ Files

An xyz file is a text file with coordinates of a number of points, and optionally including a text annotation. Each line has the form

```
x1 y1 z1 [text]
x2 y2 z2 [text]
```

where the text is optional. There are two functions provided, for reading and writing xyz files:

```
>> [x,y,z,ta] = mzReadxyz(filename)
>> [x,y,z,ta] = mzWritexyz(filename,x,y,z,ta)
```

See `help mzReadxyz` and `help mzWritexyz` for details on arguments and usage.

4.4 Plotting 2D Flexible Mesh Triangular Data

Mesh data for mesh files or 2D dfsu files being based purely on triangles are compatible with the standard MATLAB triangle plotting routines. To plot a dfsu file mesh please try the following:

```
>> infile = 'data/data_oresund_2D.dfsu';
>> dfsu2 = DfsFileFactory.DfsuFileOpen(infile);
>> xn = double(dfsu2.X);
>> yn = double(dfsu2.Y);
>> zn = double(dfsu2.Z);
>> tn = mzNetFromElmtArray(dfsu2.ElementTable);
>> trimesh(tn,xn,yn,zn); view(2); axis equal; colorbar
```

Plotting a mesh file is very similar:

```
>> [t,Nodes] = mzReadMesh('data/bathy_oresund.mesh');
>> trimesh(t,Nodes(:,1),Nodes(:,2),Nodes(:,3));
>> view(2); axis equal; colorbar
```

For a 2D file consisting of mixed triangles/quadrilaterals, trimesh will not work, instead use:

```
>> mzPlotMesh(t,Nodes);
```

Plotting dfsu item data cannot be done directly in MATLAB. The reason is that standard MATLAB triangular plotting routines are based on node values (finite element data), while most items in the dfsu files contain element centre values (finite volume data). There are several ways to plot dfsu data in MATLAB:

- Create a triangular mesh based on element centre nodes. Then the raw data will be plotted, but not on the original mesh
- Interpolate element centre values to node positions. Data values are slightly modified, but are plotted on the original mesh
- Use the supplied `mzPlot` routine.

There are routines included supporting the first two options. See the `read_dfsu_2D.m` example for details.