DHI MATLAB Toolbox

User Guide for Version 2011, 2012 & 2014

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 2011, 2012 & 2014

1 Introduction

This document constitutes the user guide and documentation for the DHI MATLAB Toolbox, and especially the `dfsTSO` class 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 (Release 2014; for earlier releases it can be found in the standard installation).

This version of the toolbox only works together with a MIKE by DHI product version 2011 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. Also, the toolbox contains a class for handling dfs0 files, the `dfsTSO` class, which can be used when working with dfs0 files.

DFS files are handled by the `DHI.Generic.MikeZero.DFS` component, which is a .NET component installed with MIKE Zero, MIKE URBAN 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.

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 MIKE by DHI product version 2011 or later must be installed in order to work with DFS files. MIKE Zero (any sub-product), MIKE URBAN and MIKE SDK will do the job.

The remaining tools, like, e.g., the `mzMeshAnalyser`, does not require any other DHI software, and will work in any fairly recent version of MATLAB.

There is a version 2007 of the MATLAB DFS interface which works with release 2007 of MIKE Zero, and a 2008 version which works with release 2008 of MIKE Zero. From release 2011 and onwards, the MATLAB toolbox is very different from previous versions. The current version will NOT work with older versions of MIKE by DHI (MIKE Zero)

2.2 Installing the DHI MATLAB Toolbox

Download the DHI MATLAB Toolbox (per February 2014):

http://www.mikebydhi.com/Download/DocumentsAndTools/Tools/DHIMatLabToolbox.aspx

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('C:\Matlab\mbin');
```

This will add 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 folder 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

On computers already having MIKE Zero installed, you can only install and use the DHI MATLAB Toolbox of same version. The DHI MATLAB Toolbox exists in special versions for version 2007 and 2008. From release 2011 and later of MIKE Zero, the same toolbox can be used.

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 two MATLAB classes for handling files, the `dfsTSO` class for handling dfs0 files and the `DFS` class for a fast preview of the content of dfs files.

The `dfsTSO` class provides more convenient handling of dfs0 files than the DHI.Generic.MikeZero.DFS component does. The `dfsTSO` class that handles dfs0 files is based on the MIKE Objects Timeseries Package, which is a COM interface. MATLAB supports COM interfaces natively; hence if you seek more advanced functionality than described here, you can use the COM interface directly.

```
>> TSO = get(dfs,'TSObject')
TSO =
       COM.TimeSeries_TSObject
```

Through the COM interface you have direct access to all properties of the file, and can perform more advanced operations than through the dfsTSO class, as e.g., merging of two items from two files, interpolation to new timestep times. Please consult the documentation for the Timeseries Package for further information.

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. There are 2 different versions, one using `dfsTSO`. And one using the `DHI.Generic.MikeZero.DFS` library.
- `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_dfs0_dfsTSO` – how to create a dfs0 time series file using the `dfsTSO`.
- `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 using dfsTSO Object

3.2.1 Open a file

You open a file by creating a new `dfsTSO` object. 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:

```
>> dfs0  = dfsTSO('data/data_ndr_roese.dfs0')
dfs0 =
   filename            : data/data_ndr_roese.dfs0
   dimensions          : 1
   number of items     : 5
            item   1   : Point 1: Surface elevation
            item   2   : Point 1: U velocity
            item   3   : Point 1: V velocity
            item   4   : Point 1: Current speed
            item   5   : Point 1: Current direction
   time axis type      : Calendar time axis, equidistant
   startdate           : 1993-12-02 00:00:00.000
   enddate             : 1993-12-13 00:00:00.000
   timestep interval   :    120.000 (seconds)
   number of timesteps : 7921
```

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

```
>> dfs0
dfs0 =
   filename            : data/data_ndr_roese.dfs0
   dimensions          : 1
   [... remaining output omitted ...]
```

3.2.2 Closing a file

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

```
>> close(dfs)
```

Setting `dfs=0` will free associated memory and have the same effect as `close(dfs)`.

3.2.3 Saving file

File data are only saved when the `saveAndClose` function is issued:

```
>> saveAndClose(dfs)
```

If the file already exists, a dialogue will appear and ask if you wish to overwrite the file. To suppress the warning, use instead

```
>> saveAndClose(dfs,1)
```

which will save and overwrite without a warning dialogue.

3.2.4 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 get function only with the dfs object as argument:

```
get(dfs0)
            FileName: 'data/data_ndr_roese.dfs0'
           FileTitle: 'Ndr Roese'
          Dimensions: 1
            NumItems: 5
           ItemNames: {5x1 cell}
               Items: {5x6 cell}
      ItemValueTypes: {1x5 cell}
    ItemCoordinates: [5x3 double]
       TimeAxisType: 'Equidistant_Calendar'
          StartDate: [1993 12 2 0 0 0]
            EndDate: [1993 12 13 0 0 0]
        TimestepSec: 120
           Timestep: [0 0 0 0 2 0]
       NumTimeSteps: 7921
         Projection: ''
        DeleteValue: 1.0000e-035
           TSObject: [1x1 COM.TimeSeries_TSObject]
```

If you want a certain part of the header information, it can be put as the second argument (not case sensitive).

```
>> get(dfs0,'FileName')
ans =
data/data_ndr_roese.dfs0

>> get(dfs0,'ItemNames')
ans =
    'Point 1: Surface elevation'
    'Point 1: U velocity'
    'Point 1: V velocity'
    'Point 1: Current speed'
    'Point 1: Current direction'
```

3.2.5 Setting header information / object properties

The set function can update header data, and works as the built-in set function on general MATLAB objects.

To view which header data are settable, use the set function only with the dfs object as argument.

```
>> set(dfs0);
           FileName: 'data/data_ndr_roese.dfs0'
          FileTitle: 'Ndr Roese'
          ItemNames: {1x5 cell}
     ItemValueTypes: {1x5 cell}
    ItemCoordinates: [5x3 double]
       TimeAxisType: 'Equidistant_Calendar'
          StartDate: [1993 12 2 0 0 0]
           TimeStep: [0 0 0 0 2 0]
        DeleteValue: 1.0000e-035
```

To set a property

```
>> set(dfs0,'filename','mynewfilename.dfs0')
```

See, e.g., `help dfsTSO/set` for full information on syntax.

3.2.6 Show item definitions

A textual detailed description of each item can be obtained by using:

```
>> showItemDefs(dfs0)
   #items    : 5
item   1
   Name      : Point 1: Surface elevation
   EUMType   : Surface Elevation
   EUMUnit   : m
item   2
   [... remaining output omitted ...]
```

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

3.2.7 Reading item data

To load data from item 2, timestep 14, use:

```
>> data = readItemTimestep(dfs0,2,14);
```

You may read data using subscript indexing, i.e., the same data can be loaded using:

```
>> data = dfs0(2,14);
```

You may retrieve more than one timestep at a time, e.g.,

```
>> data = dfs0(2,14:17);    % read timestep 14 to 17
>> data = dfs0(2,[4 6 8]); % read timestep 4,6 and 8
>> data = dfs0(2);          % read all timesteps
```

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

```
>> data = dfs0(end,end);
```

will read the last timestep of the last item.

See `help readItemTimestep` for details.

3.2.8 Writing item data

Writing item data follows the readItemTimestep functionality:

```
>> writeItemTimestep(dfs0,2,14,data);
>> dfs0(2,14) = data;
```

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

Data are updated in the memory but not saved to the file. To save to file, use the `saveAndClose` function.

3.2.9 Read time information

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

```
>> t = readTimes(dfs0,5);
```

which will the read time for timestep 5. Times can be read in the following ways:

```
>> t = readTimes(dfs0)          % read all timesteps
>> t = readTimes(dfs0,5)        % read timestep 5
>> t = readTimes(dfs0,5:10)     % read timesteps 5 to 10
>> t = readTimes(dfs0,[5,7,10]) % read timesteps 5,7,10
```

For calendar type time axis the result will return one row with 6 columns for each timestep requested. Each row will contain year, month, day, hour, minute and decimal seconds, as returned by the MATLAB function `datevec`. For a relative type time axis, only on number per timestep will be returned.

Note that timestep indices start at 0, i.e., the first timestep has index 0.

There is a performance issue with the non-equidistant calendar time axis type. Thus, it is not advisable to read more than 1000 timesteps at a time.

3.2.10 Adding / removing timesteps and editing time

To add timesteps, use:

```
>> addTimesteps(dfs0,10);
```

which will add 10 timesteps after the last timestep, incremented with the default timestep interval. Each item will have delete values added to the added timesteps.

To remove existing timesteps, use

```
>> removeTimesteps(dfs0,9);
```

which will remove the timestep at index 9 (the 10th timestep) from the file. For equidistant time axis types, only timesteps at the beginning and the end can be removed. For non-equidistant time axis types, any timestep can be removed. You can remove several timesteps in one call, see `help DFSTSO/removeTimesteps` for details.

By default a new dfs object gets the equidistant calendar time axis type. If you need another time axis type, use:

```
>> set(dfs0,'timeaxistype','non_equidistant_calendar')
```

The time axis type must be one off:

```
'undefined'
'Equidistant_Relative'
'Non_Equidistant_Relative'
'Equidistant_Calendar'
'Non_Equidistant_Calendar'
```

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.

Equidistant time axis types

The timestep times are updated by setting the start date/time and the timestep interval:

```
>> set(dfs0,'startdate',[2005 6 25 15 30 0]);
>> set(dfs0,'timestep',[0 0 0 0 1 0]);
```

which will set the start date to June 25th 2005 15:30:00 and the time step interval to one minute.

Note that the start date format depends on whether the time axis type is a calendar or relative. Use `set(dfs,'startdate')` to see the format.

Non-equidistant time axis types

Each timestep time can be set individually by using:

```
>> writeTimes(dfs0,5,[2005 6 25 15 35 00])
```

which will set the date June 25.th 2005 15:35:00 to timestep number 6 (remember that timestep indices start at 0, i.e., the first timestep has index 0 and the 6 timestep index 5). It is possible to update times for one, all or a list of timesteps in one call to `writeTimes`, See `help DFSTSO/writeTimes` for a list of valid arguments.

Note that the date format depends on whether the time axis type is a calendar or relative. Use `readTimes(dfs)` to see the format.

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.

3.2.11 Editing / removing items

For existing items you can alter the EUM type and unit, using:

```
>> setItemEum(dfs0,2,'Current Speed','m/s');
```

to set the EUM type and unit for item 2 in this case.

See `help DFSTSO/setItemEUM` for details of arguments.

You can remove an item (`dfsTSO` only) using:

```
>> removeItem(dfs0,1);
```

which will remove the first item from the file.

Note that item numbers start at one (as opposed to timestep numbers which start at 0).

3.2.12 Handling spatial information

For dfs0 files, you can retrieve and set item coordinates using the set function, try:

```
>> set(dfs0,'itemcoordinates')
```

3.3 Creating a New File

The steps for creating a new file are as follows:

1. Make a new dfs object
2. Set a temporal definition
3. Set a spatial definition
4. Add items
5. Create the file
6. Insert data

Then you are ready to update the data for items and timesteps as described previously.

When adding items, the EUM type and the EUM unit must also be specified:

```
>> addItem(dfs,'Surface','Surface Elevation','m');
```

which will add a new item to the dfs object, with the name, item type and item unit specified. The third argument must be a valid EUM type string, and the fourth argument must be a valid EUM unit string that matches the EUM type specified.

Note that you cannot set a unit that is not valid for the type, i.e., for the item type `'Surface Elevation'`, the unit must be either `'m'` or `'ft'`, `'m/s'` is not allowed. To get a list of valid EUM type and unit strings, see `help listEumTypes` and `help listEumUnits`.

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.