MATLAB DFS

Interface Library

User Guide

DHI **Water • Environment • Health**
Agern Allé 5
DK-2970 Hørsholm
Denmark

Tel: +45 4516 9200
Fax: +45 4516 9292
E-mail: dhi@dhigroup.com
Web: www.dhigroup.com

CONTENTS

MATLAB DFS Interface Library User Guide

1 INTRODUCTION

This document constitutes the user guide and documentation for the Matlab DFS Interface Library.

1.1 Why a Matlab Interface?

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 binary file format named DFS (Data File System). These data cannot be loaded directly into the Matlab workspace. To overcome this problem we have developed a DFS interface to Matlab including functions for reading, writing and creating various types of DFS files.

1.2 Supported Data File Types

The Matlab DFS interface provides different levels of support for different file types.

- Read support means the file can be opened and data can be read from the file
- Write support means that an existing file can be opened, and data can be modified and written back to the file
- Create support means that a new file can be created from scratch, defining temporal and spatial information and writing data to the file

Version 2007 supports:

dfs0:	Full support (read, write, create)
dfs1+2+3:	Read support only
dfsu 2D and 3D:	Read and write support.
mesh and xyz files:	Read and create support

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

Please note that the previous DHI Matlab DFS interface provides create-support for dfs2 files. For all other file types this new Matlab DFS interface provides more functionality than the previous version. It is the intention in time to provide write support also for dfs1+2+3 files.

1.3 User Support

The Matlab DFS interface software 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, and requests for new features please contact software@dhigroup.com and add 'Matlab DFS' in the email subject line.

1.4 Disclaimer

A disclaimer is included in the installation[2]

[2] Matlab DFS interface Disclaimer.pdf document

2 INSTALLATION

2.1 Requirements

Matlab version 7.2 or later is required for all features to work.

Matlab version 7.0 or later lacks support for a few features. These are related to setting date/time for calendar type time axis, e.g., when adding or editing time/date for timesteps while creating/modifying dfs0 files. For all other features, version 7.0 will work well.

This current Matlab DFS interface is based on one of two other components:

- MIKE Objects Timeseries Package, version 2007, which is a COM interface for accessing (reading, writing, creating) dfs0 files
- MIKE Zero version 2007[3]

It is required that either the MIKE Object Timeseries package version 2007 or MIKE Zero version 2007 or a later version is installed on the computer.

2.1.1 When MIKE Zero is installed on the computer

MIKE Zero version 2007 by default installs all components necessary to install and run the Matlab DFS interface. Earlier versions of MIKE Zero are not compatible with the Matlab DFS interface.

2.1.2 When MIKE Zero is not installed on the computer

Then you need first to download and install the MIKE Object Timeseries Package. MIKE Objects is available from the DHI download web server (per February 2007):

http://www.dhisoftware.com/MIKEobjects/

Download and follow the install instructions for installing the MIKE Object Timeseries Package.

[3] A demo version can be downloaded here http://www.dhigroup.com/Software/Download/DHISoftware2007.aspx (eg. MIKE 21)

2.2 Installing the Matlab DFS Interface

Download the Matlab DFS interface from the DHI download web server (per February 2007):

http://www.dhisoftware.com/mike21/Download/Tools/MatLabDFS2007.htm

Make sure to download the version that matches your version of MIKE Zero or MIKE Object Timeseries package.

Get the DFSMatlab_mbin_vXXXX.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 DFSManager interface. In the Matlab command prompt you may try to issue the command

```
>> dfsManager()
```

and if installed correctly, it should return a message in the form

```
>> dfsManager()
ans =
   Empty dfsManager object. Not initialized
```

Now the Matlab DFS interface is installed correctly.

2.2.1 Installing examples

Get the DFSMatlab_Example.zip. Unzip the file to a folder of your choice. Change the current directory to this folder, and run any of the read_xxx.m or write_xxx.m scripts.

There are examples for reading all file types, and writing and creating for those that support this. 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 Matlab DFS interface of same version. The Matlab DFS interface starts at version 2007, i.e., MIKE Zero version 2005 and prior cannot work with this Matlab DFS interface.

This is true for handling dfs1+2+3 and dfsu files, however, for dfs0 files also older versions of MIKE Zero will be compatible.

For mesh and xyz file support, there are no requirements.

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.

The Matlab DFS interface library consists of two Matlab classes: One for handling dfs1+2+3+u, the `dfsManager` class, and one for handling dfs0 files, the `dfsTSO` class. However, the interface is very similar for the two classes, and from a users point of view, it is not necessary to know both. Using the `dfsManager` on dfs0 files will utilize `dfsTSO` functionality.

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 Common Functionality for all dfs Files

This section describes the functionality common for the two dfs objects.

3.1.1 Open a file

You open a file by creating a new `dfsManager` 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:

```
>> dfs = dfsManager('data_oresund_2D.dfsu')
dfs =
   filename            : data_oresund_2D.dfsu
   dimensions          : 2
   number of nodes     : 2057
   number of elmts     : 3636
```

```
   number of items     : 4
             item     1     :  Surface  elevation     (elmt
values)
             item   2   : U velocity        (elmt values)
             item   3   : V velocity        (elmt values)
             item   4   : Current speed     (elmt values)
   time axis type      : Calendar time axis, equidistant
   startdate           : 1993-12-02 00:00:00.000
   enddate             : 1993-12-04 00:00:00.000
   timestep interval   :  14400.000 (seconds)
   number of timesteps : 13
   projection          : UTM-33
```

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

```
>> dfs
dfs =
   filename            : data_oresund_2D.dfsu
   ...[remaining output omitted]
```

Header information will vary depending on the file type. It will always show information on items, time, dimension and size.

If opening a dfs0 file, the `dfsManager` will automatically return a `dfsTSO` object, and the user will not note any difference.

3.1.2 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)
```

Remember to save the file if changes have been made. Otherwise changes are discarded.

For dfs0 files, setting `dfs=0` will free associated memory and have the same effect as `close(dfs)`, however this is not the case for the remaining dfs files.

If you set `dfs=0` for all but dfs0 files, the handle to the open file is lost, but memory is not freed. The only way to free this memory is to call

```
>> closeAll(dfsManager())
```

Which will close all `dfsManager` files and free associated memory, also of those files where an object is still available.

3.1.3 Saving file

File data are only saved when the save function is issued:

```
>> save(dfs)
```

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

```
>> save(dfs,1)
```

which will save and overwrite without a warning dialog.

3.1.4 Getting header information / object properties

The get function will return header data. To view available header data, use the get function only with the dfsManager object as argument:

```
>> get(dfs)
        FileName: 'data_oresund_2D.dfsu'
      Dimensions: 2
        NumItems: 4
       ItemNames: {4x1 cell}
           Items: {4x7 cell}
    TimeAxisType: 'Equidistant_Calendar'
       StartDate: [1993 12 2 0 0 0]
         EndDate: [1993 12 4 0 0 0]
     TimeStepSec: 14400
    NumtimeSteps: 13
      Projection: 'UTM-33'
        NumElmts: 3636
        NumNodes: 2057
     DeleteValue: 1.0000e-035
```

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

```
>> get(dfs,'filename')
ans =
data_oresund_2D.dfsu

>> get(dfs,'itemnames')
ans =
    'Surface elevation'
    'U velocity'
    'V velocity'
    'Current speed'
```

3.1.5 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 set object properties is only relevant for files with write or create support. Not all obtainable properties are settable. To view which

header data are settable, use the set function only with the dfsManager object as argument.

```
>> set(dfs)
    FileName: 'mynewfilename.dfsu'
```

To set a property

```
>> set(dfs,'filename','mynewfilename.dfsu')
```

See `help dfsManager/set` for full information on syntax.

3.1.6 Reading item data

Data exist either as node based or element based, depending on the file type. When loading file data, the raw data are returned, so the user must know whether the data are node or element based. To load data from item 2, timestep 14, use:

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

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

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

For dfs0 files you may retrieve more than one timestep at a time, e.g.,

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

For remaining dfs files, only one timestep at a time can be retrieved.

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

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

will read the last timestep of the last item.

For flexible mesh data (dfsu), a vector with element data is returned. For 3D dfsu data, a matrix is returned with the number of elements in 2D as rows, and each layer as columns.

For structured mesh data, a scalar, a vector, a matrix or a 3D matrix is returned for a dfs1+2+3 file respectively.

3.1.7 Writing item data

Data exist either as node based or element based, depending on the file type. When loading file data, the raw data are returned, so the user must know whether the data are node or element based. To write data to item 2, timestep 14, use:

```
>> writeItemTimestep(dfs,2,14,data);
```

You may write data using subscript indexing, i.e., the same data can be written using:

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

The data to write should have the same format as the data read. For flexible mesh files, the data argument must be a vector with element data. For structured mesh files, data must be a scalar, a vector, a matrix, or a 3D matrix for a dfs0+1+2+3 file respectively.

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

Note: It is not currently supported to write data to dfs1+2+3 files.

3.1.8 Read time information

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

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

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

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

Note: For dfs1+2+3+u files, the non-equidistant time axis types (both calendar and relative) are not supported (they are quite rare). For dfs0 files 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.

3.1.9 Show item definitions

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

```
>> showItemDefs(dfs)
   #items   : 2
item   1
   Name     : Surface elevation
   EUMType  : Surface Elevation (100078)
   EUMUnit  : meter (1000)
item   2
   Name     : U velocity
   EUMType  : u-velocity component (100269)
   EUMUnit  : m/s (2000)
```

3.1.10 Loading mesh data

Data exist either as data on nodes or data in elements. It is possible to load nodes as well as element coordinates. Therefore, different functionalities are provided, and the user must know which to use, depending on the file type. You can load the coordinates of nodes and element centres using.

```
xyz       = readElmts(Hdfs);    % reads coordinates to
matrix
[x,y,z]  =  readElmts(Hdfs);    % reads coordinates to
vector

xyz       = readNodes(Hdfs);    % reads coordinates to
matrix
[x,y,z]  =  readNodes(Hdfs);    % reads coordinates to
vector
```

For flexible mesh data, you can load the connectivity of each element, i.e., a table describing for each element which nodes are used.

```
>> t = readElmtNodeConnectivity(Hdfs);
```

For triangular meshes, you can use the build in triangular plotting routines, e.g.:

```
>> [x,y,z] = readNodes(Hdfs);
>> t = readElmtNodeConnectivity(Hdfs);
>> trimesh(t,x,y,z)
```

The trimesh function is a build in Matlab plot-routine. For mixed triangular/quadrilateral meshes, to plot use:

```
>> t = mzPlot(t,x,y,z,c);  % NOT YET PROVIDED!!!
```

For structured mesh data readElmts and readNodes return a vector, a matrix or a 3D matrix for a dfs1+2+3 file respectively. Equidistant data only return a start and end coordinate and a grid spacing.

3.2 Reading Spatial Information

For regular mesh files, dfs1+2+3, spatial information exists in a grid, where each element has a coordinate. Use the `readElmt(dfs)` to get element coordinates.

For flexible mesh files, dfsu, spatial information exists as nodes and elements. Each node has a coordinate, and for an element is specified a list of nodes that defines a polygon (2D) or polyhedron (3D) encompassing the element, called the element-node-connectivity table.

Use the `readNodes(dfs)` to get node coordinates, `readElmt(dfs)` to get coordinate of the centre of the element, and `readElmtNodeConnectivity(dfs)` to get the element-node-connectivity table.

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

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

3.2.1 Reading node coordinates

Read the nodes coordinates of a dfsu files using:

```
>> [x,y,z] = readNodes(dfs)
```

3.2.2 Reading element coordinates

Read the element coordinates using:

```
>> [x,y,z] = readElmts(dfs)
```

For a dfs1+2+3 file, this will produce a coordinate of each of the data points.

For a dfsu file, this will produce the coordinate of the centre of the element.

3.2.3 Reading element-node-connectivity table

Read the element-node-connectivity table for a dfsu file using:

```
>> t = readElmtNodeConnectivity(dfs)
```

For 2D dfsu files, the result `t` will have 3 or 4 columns, depending on being pure triangular or mixed triangular/quadrilateral.

For 2D dfsu files only consisting of triangles, the result will be a connectivity table, which is compatible with the Matlab triangle plotting routines. For details see the section on plotting flexible mesh data.

For 3D dfsu files, the result `t` will have 6 or 8 columns, depending on being pure triangular or mixed triangular/quadrilateral.

3.3 Creating a New File

It is currently only supported from within Matlab to create new dfs0 files.

If you wish to create new dfsu files, you can use the Data Manager (a part of MIKE Zero), which will guide you through the creation of the spatial definition, the temporal definition, and adding the necessary items. When this is done, you can use Matlab to update the item data in the file.

For dfs1+2+3 files there is currently no support for creating or modifying item data.

The steps for creating a new file are as follows:

- Create a new dfs object
- Add items
- Add timesteps

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

3.3.1 Creating the object

To create a new dfs object, the create argument must be set

```
>> dfs = dfsManager('my_new_file.dfs0',1);
```

This object is empty and should not be saved.

3.3.2 Adding / editing / removing items

To add items, use:

```
>> addItem(dfs,'Speed average');
```

which will add a new item to the dfs0 object, with the name specified. The type and unit of the item is undefined. To also set the type and unit, use:

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

where the third argument is a valid EUM type string, and the fourth argument is 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 DFSTSO/listEumTypes` and `help DFSTSO/listEumUnits`.

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

```
>> setItemEum(dfs,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 using:

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

which will remove the second item from the file.

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

3.3.3 Adding / removing timesteps and editing time

To add timesteps, use:

```
>> addTimesteps(dfs,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(dfs,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(dfs,'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(dfs,'startdate',[2005 6 25 15 30 0]);
>> set(dfs,'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(dfs,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.

3.4 Other Tools

As a part of the Matlab DFS interface a number of tools have been implemented in order to help the processing of data. They include:

- Reading and writing mesh files. Reorder and refine meshes
- Reading and writing xyz files

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

3.4.1 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,coordsys] = 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. coordsys is a text string representing the coordinate system and the projection.

Similar you can write a mesh file to disc using:

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

See help mzReadMesh, help mzWriteMesh for details.

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

3.4.2 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.

3.4.3 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 please try the following:

```
>> dfs = DFSManager('data/data_oresund_2D.dfsu')
>> [x,y,z] = readNodes(dfs);
>> t = readElmtNodeConnectivity(dfs);
>> trimesh(t,x,y,z); view(2);
```

Plotting a mesh file is very similar:

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

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

```
>> mzPlotMesh(t,[x,y,z]);
```

Plotting item dfsu 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 two 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

There are routines included supporting both options.

To create a triangular mesh based on element centre nodes, try:

```
>> dfsu2      = dfsManager('data/data_oresund_2D.dfsu')
>> tn         = readElmtNodeConnectivity(dfsu2);
>> [xe,ye,ze] = readElmts(dfsu2);      % Element center
coordinates
>> [xn,yn,zn] = readNodes(dfsu2);     % Node coordinates
>> te = mzTriangulateElmtCenters(xe,ye,tn);
```

```
>> se = dfsu2(4,1);                    % Load data
>> trisurf(te,xe,ye,se);
```

To interpolate element center values to node positions, try:

```
>> dfsu2      = dfsManager('data/data_oresund_2D.dfsu')
>> tn         = readElmtNodeConnectivity(dfsu2);
>> [xe,ye,ze] = readElmts(dfsu2);      % Element center
coordinates
>> [xn,yn,zn] = readNodes(dfsu2);     % Node coordinates
>> s  = dfsu2(4,1);                   % Load data
>> sn = mzCalcNodeValues(tn,xe,ye,s,xn,yn);
>> trisurf(tn,xn,yn,sn);
```

For details, consult the `read_dfsu_2d.m` file provided in the example zip archive. The example zip archive contains examples of reading all types of dfs data files.