DHI MATLAB TOOLBOX

Including DFS Interface Library

User Guide for Version 2008

Agern Allé 5	Tel:	+45 4516 9200
DK-2970 Hørsholm	Support:	+45 4516 9333
Denmark	Fax:	+45 4516 9292

E-mail: software@dhigroup.com
Web: www.dhigroup.com

WATER • ENVIRONMENT • HEALTH

CONTENTS

DHI MATLAB Toolbox and DFS Interface Library User Guide

1 INTRODUCTION

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

This version of the toolbox only works with MIKEZero version 2008 or MIKE Objects Timeseries Package, version 2008.

1.1 Why a Matlab Toolbox and DFS 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.

Furthermore, DHI have developed a number of minor tools in Matlab that many find useful, and they are made available here.

1.2 Supported Data File Types

The DFS (Data File System) is a very flexible format, and is used in many different ways throughout the DHI product line. This interface supports the most common ways of using DFS, hence not all possible variations of the DFS file format is fully supported.

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

- R: Read support means the file can be opened and data can be read from the file
- W: Write support means that an existing file can be opened, and data can be modified and written back to the file
- P: Property support means that all properties of an existing file can be modified and written back to the file.

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

- C: Create support means that a new file can be created from scratch, defining temporal and spatial information and writing data to the file

Version 2008 supports:

dfsu 2D and 3D	RWC
dfs0	RWPC
dfs1+2+3 equidistant spatial axis	RWC
dfs1 non-equidistant spatial axis	RWC, minor limits in R
Mesh files	RC
dfsu vertical profiles	Not supported
Non-equidistant temporal axis	Not supported, only for dfs0
Static item access	Not supported
Custom blocks access	Not supported

1.3 User Support

The DHI Matlab Toolbox and 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 prior) 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 should work as well.

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

- MIKE Objects Timeseries Package, version 2008, which is a COM interface for accessing (reading, writing, creating) dfs0 files
- MIKE Zero version 2008[3]

It is required that either the MIKE Object Timeseries package version 2008 or MIKE Zero version 2008 or a later version is installed on the computer for the DFS interface to work.

The remaining tools, like, e.g., the `mzMeshAnalyser`, does not require any other DHI software.

There is a version 2007 of the Matlab DFS interface which works with version 2007 of MIKE Zero. This 2008 version is greatly enhanced compared to the 2007 version, so do find the user guide for version 2007, do not read on here.

2.1.1 When MIKE Zero is installed on the computer

MIKE Zero version 2008 by default installs all components necessary to install and run the Matlab DFS interface. Earlier versions of MIKE Zero are not compatible with this version of 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 December 2007):

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

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

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

2.2 Installing the Matlab DFS Interface

Download the DHI Matlab Toolbox and DFS interface from the DHI download web server (per December 2007):

http://www.dhigroup.com/Software/Download/DocumentsAndTools/FreeTools/Marine.aspx

Locate the DHI Matlab Toolbox and DFS interface version 2008. 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, write_xxx.m or create_xxx.m scripts.

There are examples for reading, writing and creating all 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 Matlab DFS interface of same version. The Matlab DFS interface starts at version 2007, most current is version 2008. Hence 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, and for the other tools in the toolbox, 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. The interface is very similar for the two classes, they provide very similar functionality. Using the `dfsManager` on dfs0 files will not presently work.

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 Matlab DFS interface 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.
- `write_dfs1` – example of changing existing profile data.
- `write_dfs2` – example of changing existing grid data.

- `write_dfs3` – example of changing 3D grid data.
- `write_dfsu_2D` – example of changing 2D unstructured data.
- `write_dfsu_3D` – example of changing 3D 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_3Dfrom2D` – how to create a dfsu 3D file from a dfsu 2D file containing bathymetry and surface elevation data.
- `create_dfsu_3Dfrom3D` – how to create a dfsu 3D file from another dfsu 3D file.

3.2 General Functionality for all dfs Files

3.2.1 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.

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 (`dfsManager`).

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

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

For the `dfsTSO` object (dfs0 files), setting `dfs=0` will free associated memory and have the same effect as `close(dfs)`.

If you set `dfs=0` for a `dfsManager` object, 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 open `dfsManager` files and free associated memory.

3.2.3 Saving file

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

```
>> saveAndClose(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

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

which will save and overwrite without a warning dialog.

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(dfs)
           FileName: 'data_oresund_2D.dfsu'
          FileTitle: 'Area Series'
         Dimensions: 2
           NumItems: 4
          ItemNames: {4x1 cell}
              Items: {4x8 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 (not case sensitive).

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

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

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 set object properties is only relevant for files with write or create support. Not all gettable properties are settable. To view which header data are settable, use the set function only with the dfs object as argument.

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

To set a property

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

See, e.g., `help dfsManager/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(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)
```

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

3.2.7 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);    % read timestep 14 to 17
>> data = dfs(2,[4 6 8]); % read timestep 4,6 and 8
>> data = dfs(2);          % read 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. For dfs2 and dfs3 grid series files, the data in file is stored in row major order, while data in Matlab are stored in column major order. The `readItemTimestep` automatically transposes the data such that the layout is correct. If this is not desired, you can call `readItemTimestep` with a `transpose = 0` argument

```
>> data = readItemTimestep(dfs,2,14,0);
```

See `help readItemTimestep` for details.

3.2.8 Writing item data

Writing item data follows the readItemTimestep functionality:

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

The data to write must have the same format as the data returned when reading. 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 (temporary files) but not saved to the file. To save to file, use the `saveAndClose` function

3.2.9 Reading/writing spatial item data

Dfsu 3D files has one spatial item varying in time, the Z coordinate. You can read and write the spatial varying item for every timestep using:

```
Z = readSpatialItemTimestep(dfs,1,i);
writeSpatialItemTimestep(dfs,1,i,Z);
```

which reads the spatial item at timestep `i`.

3.2.10 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, 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 yet 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.

3.2.11 *Adding / removing timesteps and editing time (dfsTSO only)*

To add timesteps (`dfsTSO` only), 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 (`dfsTSO` only), 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 (`dfsTSO` only):

```
>> 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 (`dfsTSO` only):

```
>> 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 (`dfsTSO` only):

```
>> 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.

3.2.12 Editing / removing items (dfsTSO only)

For existing items you can alter the EUM type and unit (`dfsTSO` only) 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 (`dfsTSO` only) using:

```
>> removeItem(dfs,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).

3.3 Handling Spatial Information

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 for the different file types, and the user must know which to use, depending on the file type. You can load the coordinates of nodes and element centres using.

For regular mesh files, dfs1+2+3, spatial information exists in a grid, where each element has a center coordinate. Use the `readElmt(dfs)` to get element center 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. See also section 0 on how to plot flexible mesh data.

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')
```

3.3.1 Reading node coordinates

Read the nodes coordinates of a dfsu files using:

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

3.3.2 Reading element coordinates

Read the element coordinates using:

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

For a dfs1+2+3 file, this will produce a set of coordinates for each of the data axis.

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

3.3.3 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.

3.4 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. This section will cover part 1-5.

3.4.1 Creating the object

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

```
>> dfs = dfsManager('my_new_file.dfs1',1);
```

This object is empty and should not be saved.

3.4.2 Creating temporal definition

To add timesteps to a `dfsTSO` object, see section 3.4.5.

For the `dfsManager` a temporal definition is set using the `createTemporalDef` function:

```
createTemporalDef(dfs,'Equidistant_Calendar',...
       [2002,2,25,13,45,32],25,3600);
```

Currently only the two equidistant time axis's are supported. See `help createTemporalDef` for details.

3.4.3 Creating spatial definition

For dfs1+2+3 equidistant files the spatial axis is created by specifying the origin coordinate, the grid spacing and the number of points on the axis, for each dimension. For a dfs2 file, e.g.:

```
setSpatialDef(dfs,'UTM-33',[12,54],2.6,  ...
               [0,100,11;…
                0,200,6]);
```

Here x and y axis both start at 0, having a gridspacing of 100 (x) and 200 (y) respectively and has 11 (x) and 6 (y) points in their respective direction.

For a dfs1 file with non-equidistant spatial axis, you may give the x axis directly:

```
X = [1 5 9 10 12 19];
setSpatialDef(dfs,'UTM-33',[12,54],2.6,x);
```

For a dfsu 2D file, you must enter the element table and the node coordinates and code:

```
setSpatialDef(dfs,proj,Elmts,Nodes);
```

For a 3D dfsu file, you must enter the 3D element table, the node coordinates and code, and also the number of layers.

```
setSpatialDef(dfs,proj,Elmts_3D,Nodes_3D,numlayers);
```

You can also create a 3D spatial definition from a 2D definition, a layer specification and the surface elevation:

```
layers  = [0.1 0.2 0.4 0.2 0.1];
s       = % surface elevation, constant or ...
          % at node coordinates
setSpatialDef(dfs,proj,Elmts_2D,Nodes_2D,layers,s);
```

Consult the examples on details of creating the spatial definition.

3.4.4 Adding items

To add items, use:

```
>> 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`.

3.4.5 Creating the file (dfsManager only)

When the temporal definition, the spatial definition and the items have been added, you need to call the create function to prepare the object for data:

```
>> create(dfs);
```

Adding data before create has been called are not well defined. This is only necessary for the `dfsManager` object. The `dfsTSO` is ready to receive data immediately.

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 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,Nodes);
```

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 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.

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,xn,yn,s);
>> 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.