Protocol Matlab programming style

Project **BwN DM 1.1**, Deliverable **2009**

Printing: If you want to print this document select tools -> export to pdf from the top right corner

Matlab coding guidelines

Help block

All Openearth matlab routines should contain a proper help block to be made before you type any code. There are 4 essential requirements for any matlab function. These requirements are the same as adhered to by the Mathworks themselves. Preferable one cannot distinguish Openearth from official matlab functions.

- Valid **H1 line**: a ONE line description that appears when you cal command line help on a directory. This info can be superseded by any content.m file in a directory, this content.m can be distilled from all H1 lines. The H1 line should contain the function name in capitals (these appear as clickable links when calling command line help on a directory) + a concise yet effective description.
- **See also** links: enclose links to related functions at the end of the help block. These appear as clickable links when calling command line help on a function.
- Contain a description of all **input** and **output** variables.
- Contain an **example** that can be pasted onto the **matlab command line**.

When you start a new function by calling *oetnewfun* rather than just *edit*, these 4 requirements appear in a template. The resulting help block (the first commented section following the function identifier) can look something like this:

```
%FUNCTIONNAME one line comment on the basic functionality of this function (will appear in MATLAB help request)
%
% In this block important background information should be listed.
%
% syntax:
% [output] = functionName(input)
%
% input:
% inputargument = each of the input arguments should be carefully described
%
% output:
% outputargument = each of the output arguments should be carefully described
%
% example:
% An example of the application can be given here. Preferably in a form
% that can be copied in the Matlab commandline to work.
%
% See also: relatedFunction1, relatedFunction2
```

Open source copyright statements.

OpenEarth is meant to be open source. Please specify this explicitly for every new function you add. When you start a new function by calling *oetnewfun* rather than just *edit*, a GNU LGLP copyright statement with personalized whereabouts information is automatically generated. Various open source license definitions are:

- GNU Recommended!
- MIT
- creativecommons

Test function *_test.m

Enclose a working test for function before releasing any code. This code should execute the quintessential test case. In the future such functions are foreseen to adhere to specific requirements to allow for automatic testing of the OpenEarth toolbox.

Keyword value pairs.

- Pass optional arguments in the <'PropertyName', PropertyValue> style or as a struct where the fieldnames are the keywords. Use the function *set property*, as in the following example:

```
function result = your_function(varargin)
% YOUR_FUNCTION this is an example
%
%% default properties (property values can be all kinds of variables)
%
OPT = struct(...
'PropertyName1', <PropertyValue1>,...
'PropertyName2', <PropertyValue2>,...
'PropertyName3', <PropertyValue3>,...
...
'PropertyNamen', <PropertyValuen>);
%
%% overrule default properties with those available in varargin
%
OPT = setproperty(OPT, varargin{:});
%
result = OPT.PropertyName1 + 10; % just as an example
```

Proper alignment

- Align with spaces, never with tabs.
- Enclose with semi-colon.
- Add units.

Use

```
x     = 3; %[m]
width = 3; %[m]
```

instead of

```
x=3
width=3
```

Apply namespace principle for function names

Group similar functions using identical initial letters.

- Handy for tab-completion to find a function.
- Prevents double usage of function names (such a plot_test)
- Makes it possible to migrate to object-oriented programming in future.
- namespace is mandatory in case the input arguments are structs with special fields.
- Examples are:
    - nc_* for all netCDF related io
    - delft3d_* for all function related to to Delft3D
    - jarkus_* for all functions related to jarkus profiles
    - swan_* for all functions related to to swan
    - etc.

Adhere to the following conventions

- Function names should be as much as possible self explanatory.
- Use the oetnewfun command to create a new function

```
%% open a new function in the matlab editor
oetnewfun('functionname')
```

- Prevent all GUI elements (graphical objects) when not strictly necessary. This allows for batch running in terminal mode only (-nojvm) in linux machines.
- After the namespace, the function names should start whenever possible with a lower cap verb followed by other words that start with a capital (ucit_getCrossSection, jarkus_findCrossings, etc.)
- If possible applications should contain default settings to allow running without input

- Crucial applications should be accompanied by benchmarking tests which is named the same as the application including '_test' (e.g. kml_line_test)
- Vectors (e.g. x and z values) should be stored in columns
- Follow any logics imposed by similar functions, e.g.:
    - For dune erosion routines: coastal profiles should have the positive x in seaward direction.
    - For vertical profiles z should be upward, to have a right-handed coordinate system.
    - Adding a number at the end of the function name means the number of dimensions it applies to, e.g. interp1, bin2, arrow3.
    - For functions operating on NEFIS files: the function should start with vs_
- ylabels of figures should be rotated as follows: ylabel('Label text', 'Rotation', 270, 'VerticalAlignment', 'top'); This prevents axislabels to be upside-down when figure is used as landscape in a portrait report.
- Before creating a new routine an OpenEarth developer should first check the toolbox to see if the new routines filename is unique or not.
- Use fullfile instead of a combined string to create a file or pathname. This much more flexible because: 1) it is platform independent 2) it doesn't matter whether the pathname ends with a filesep or not (''fullfile('p:', 'mctools', 'ucit.demo')'' gives the same result as ''fullfile('p:\mctools\', 'ucit.demo')'' and even the same result as ''fullfile('p:/mctools/', 'ucit.demo')'')

Functions should be placed in the existing repository structure as much as possible.

However, the existing OpenEarth is NOT considered ultimate and divine, adjust it when necessary. Generic functions should be stored under 'general', application specific functions under 'applications'.