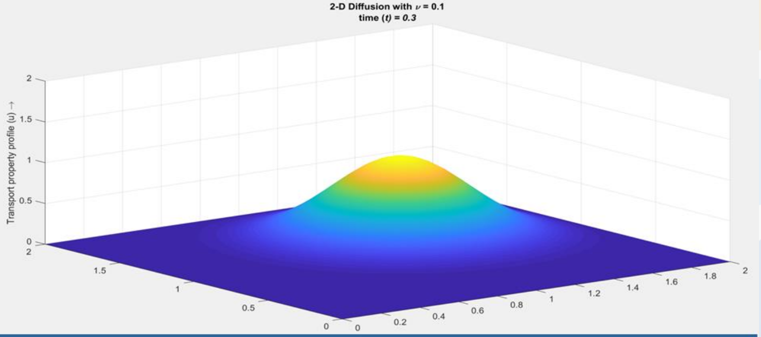
Introduction to the muitoolbox

Ian Townend



ModelUI

2D diffusion model

ModelUI

Managed Realignment Breach Model

ModelUI

Vertical Tidal Profile

ModelUI

SimpleTide

ModelUI

Inshore Waves

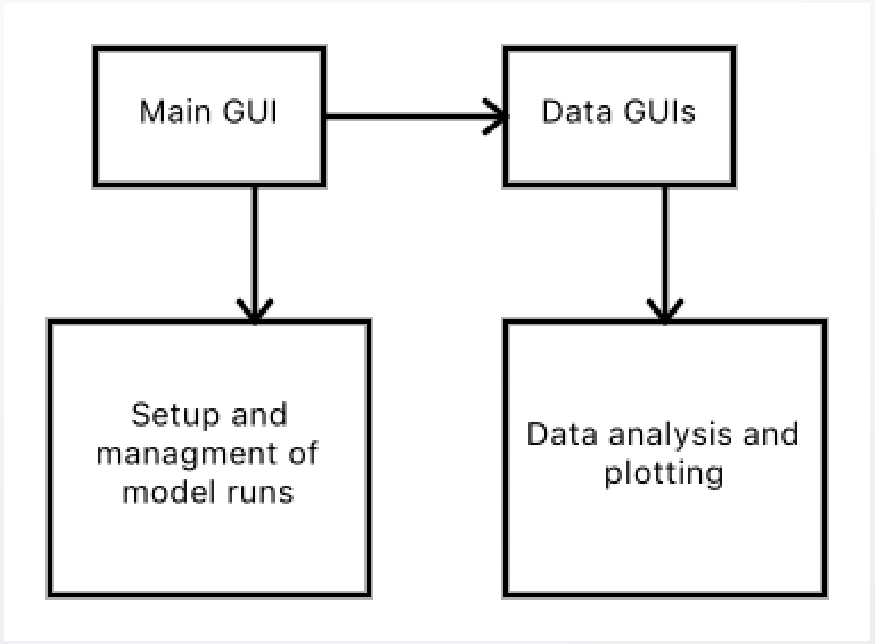
Concept

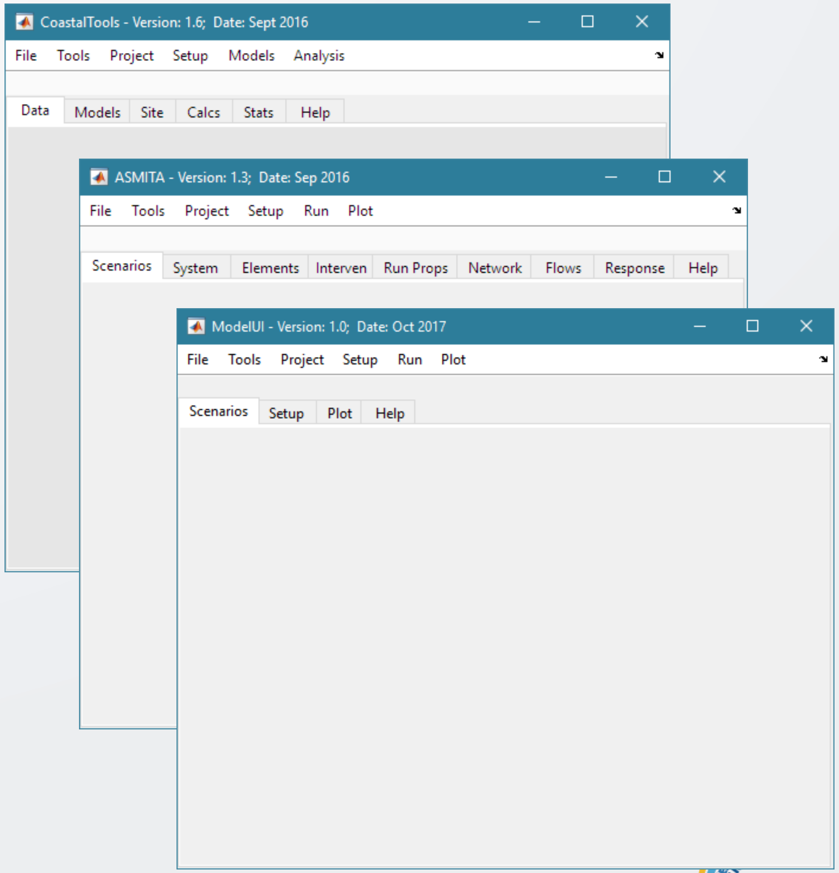
- A standard modelling interface to enable rapid model prototyping.
- Provides a User Interface (UI), management of model runs and a range of utilities, including plotting.
- Can easily be adapted. Depending on complexity of model may only need to write input definition and some code to call the new model.
- Can also extend GUI to add tabs and additional menu options.

COASTALSEA.UK

Common interface

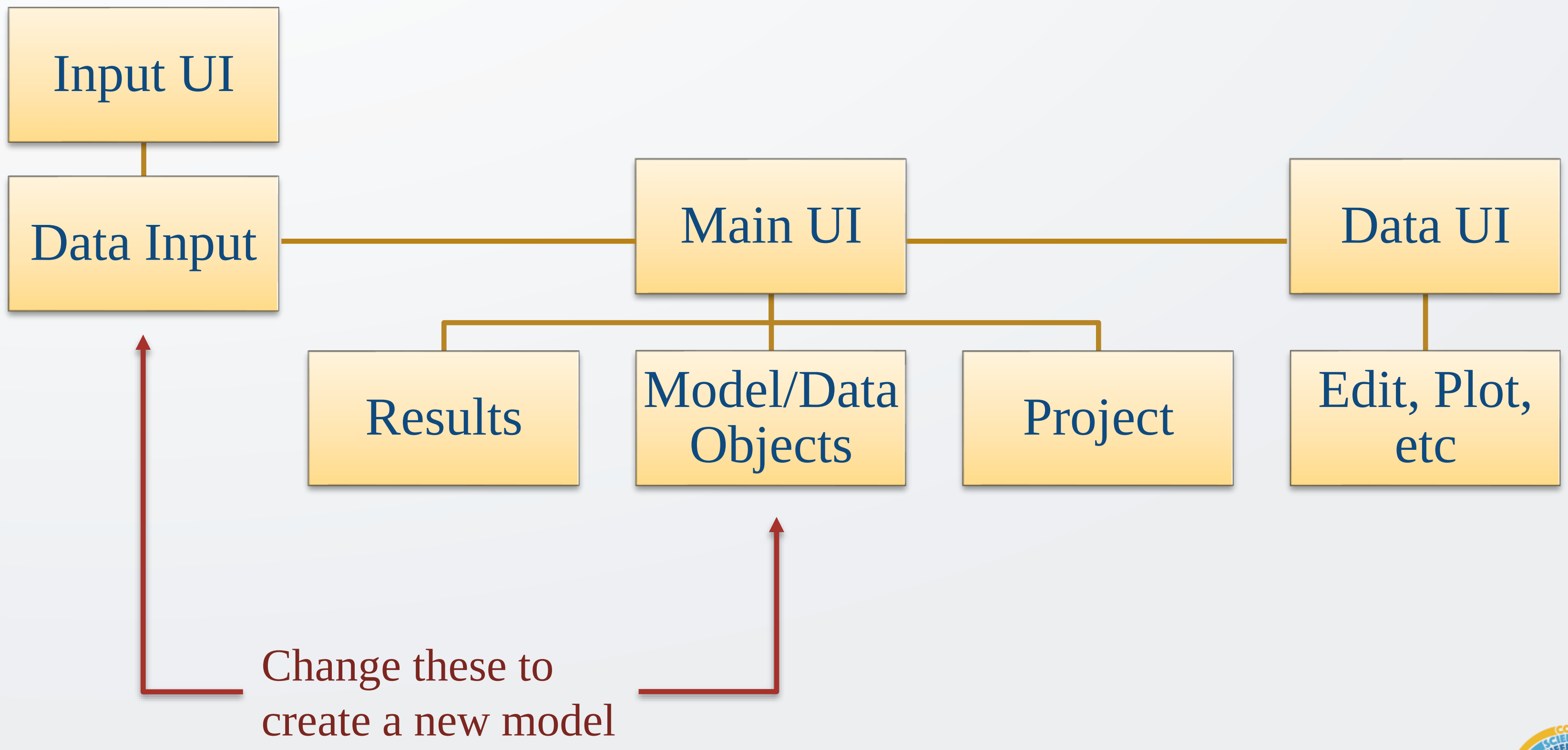
Use the same underlying class interfaces




CoastalTools - Version: 1.6; Date: Sept 2016
File Tools Project Setup Models Analysis
Data Models Site Calcs Stats Help
ASMITA - Version: 1.3; Date: Sep 2016
File Tools Project Setup Run Plot
Scenarios System Elements Interven Run Props Network Flows Response Help
ModelUI - Version: 1.0; Date: Oct 2017
File Tools Project Setup Run Plot
Scenarios Setup Plot Help

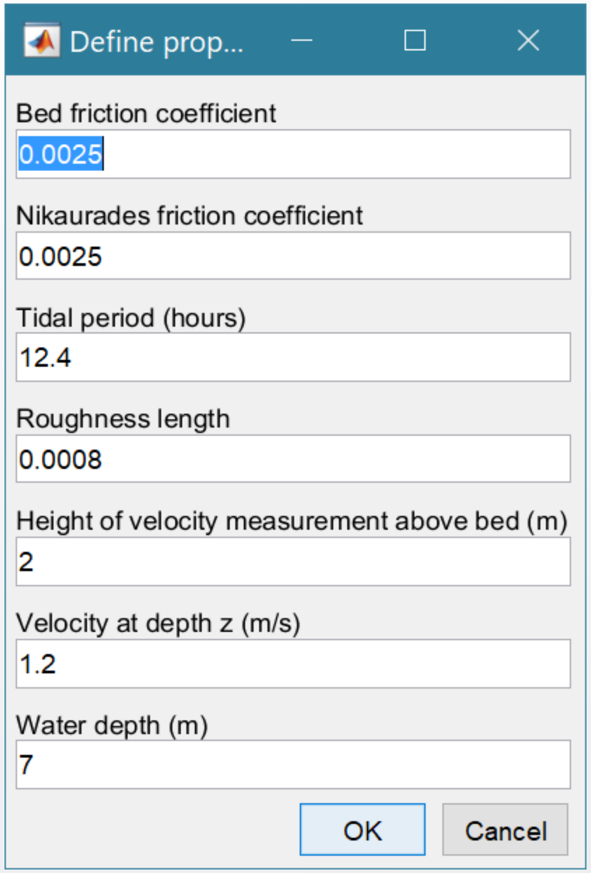
COASTALSEA.UK

ModelUI Structure



COASTALSEA.UK

Input UIs and Tab display generated automatically



Define prop...

Bed friction coefficient
0.0025

Nikaurades friction coefficient
0.0025

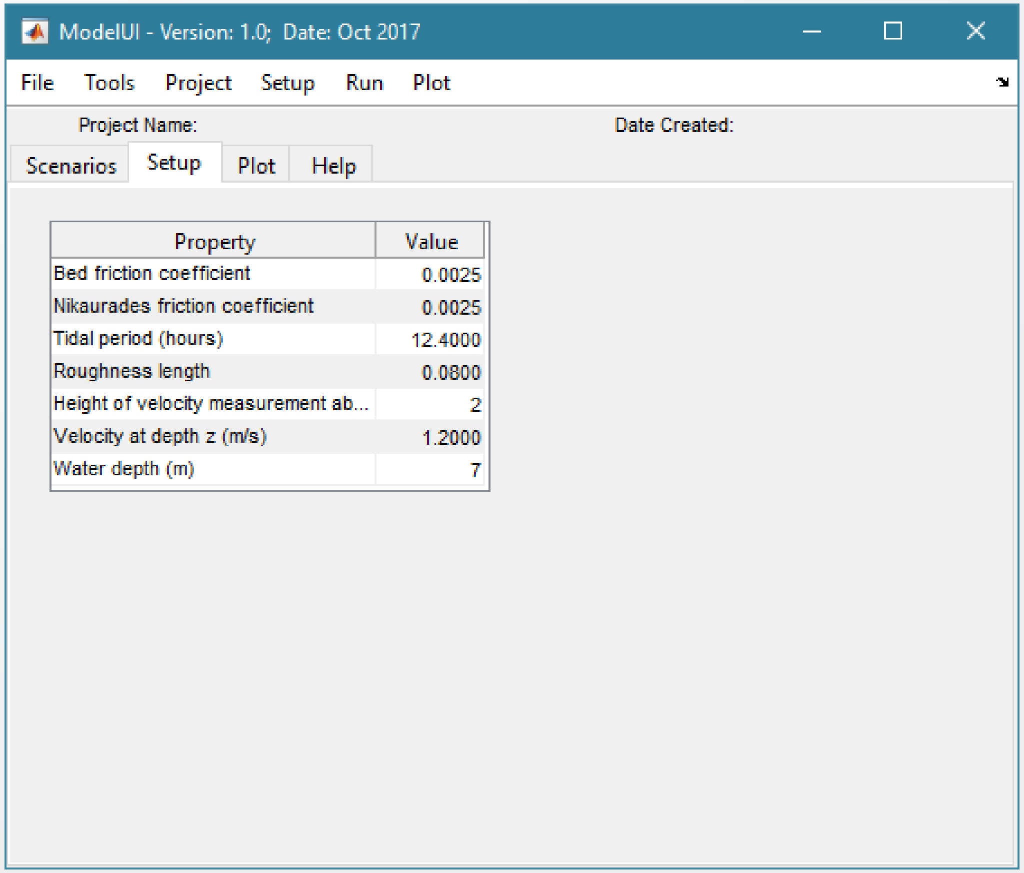
Tidal period (hours)
12.4

Roughness length
0.0008

Height of velocity measurement above bed (m)
2

Velocity at depth z (m/s)
1.2

Water depth (m)
7

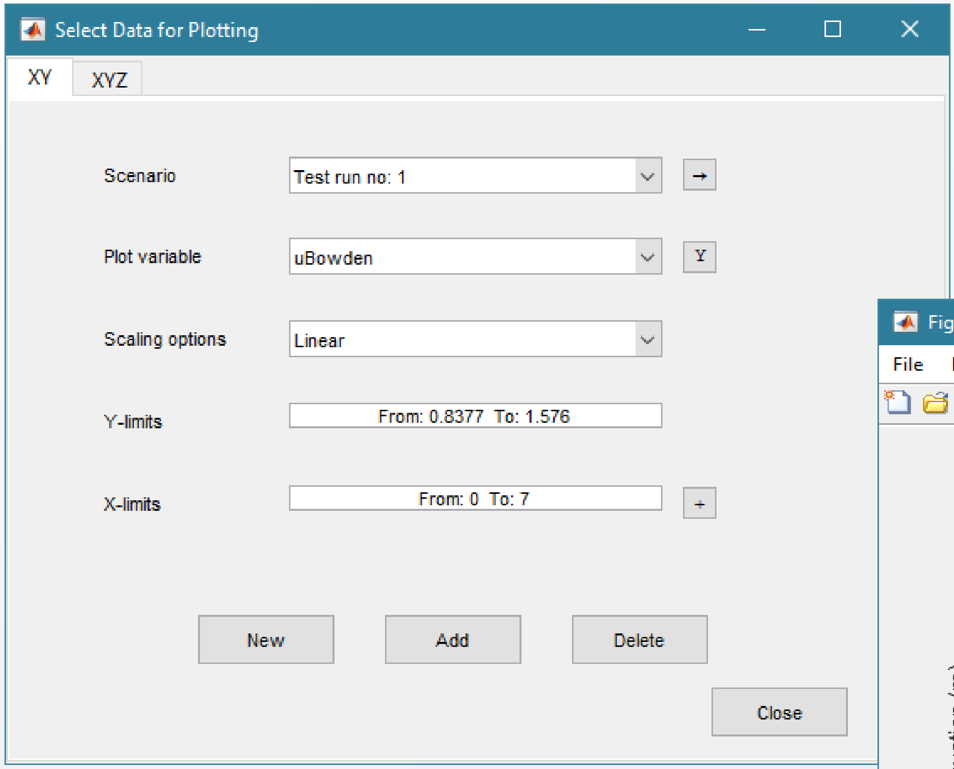


ModelUI - Version: 1.0; Date: Oct 2017

Project Name:
Date Created:

Property	Value
Bed friction coefficient	0.0025
Nikaurades friction coefficient	0.0025
Tidal period (hours)	12.4000
Roughness length	0.0800
Height of velocity measurement ab...	2
Velocity at depth z (m/s)	1.2000
Water depth (m)	7

Plotting, Editing and Derived output as part of UI



Select Data for Plotting

Scenario	Test run no: 1
Plot variable	uBowden
Scaling options	Linear
Y-limits	From: 0.8377 To: 1.576
X-limits	From: 0 To: 7

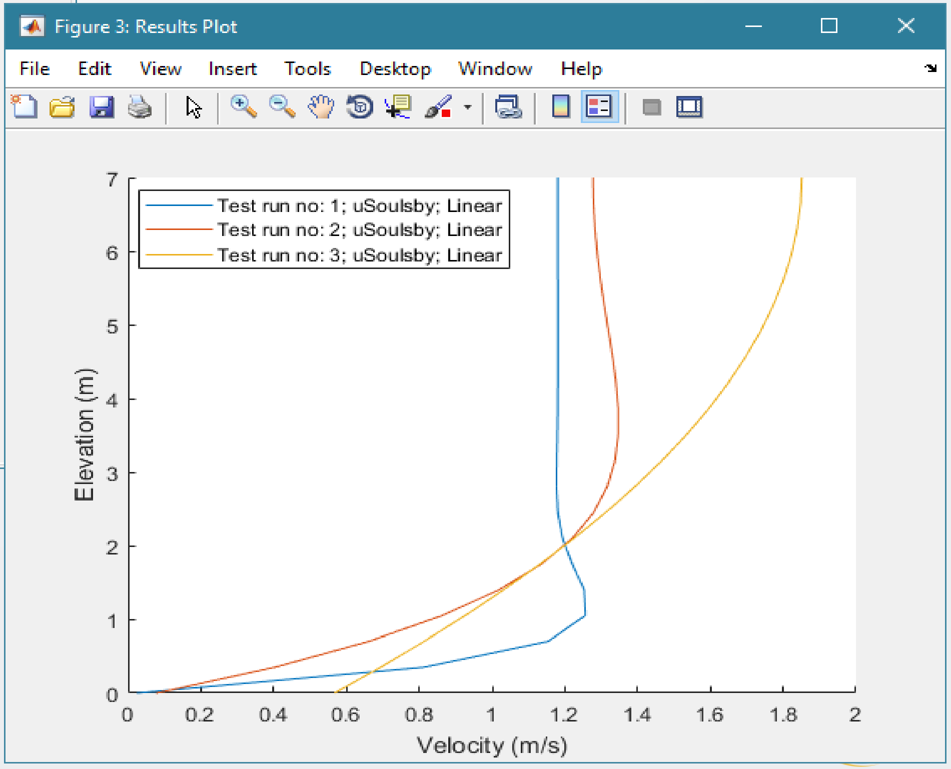
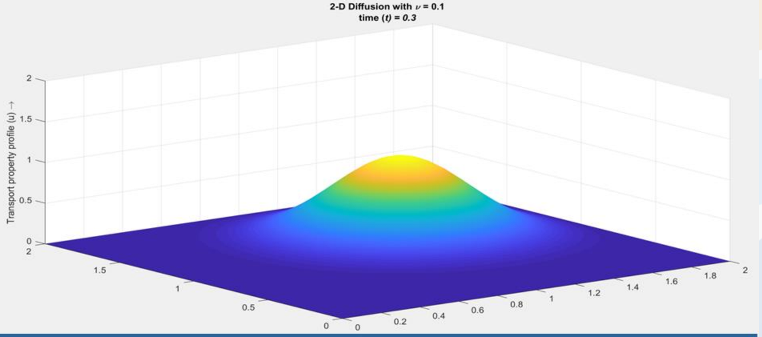


Figure 3: Results Plot

Demonstration models to illustrate a range of applications



ModelUI

2D diffusion model

ModelUI

Managed Realignment Breach Model

ModelUI

Vertical Tidal Profile

ModelUI

SimpleTide

ModelUI

Inshore Waves

Demonstration models in the ModelUI App

ModelUI handles model outputs that vary in any combination of variable, v, and dimensions x, y, z and t.

Models provided in the ModelUI App illustrate different types of application:

- Vertical Tidal Profile: simple x-v data set
- SimpleTide: time data (t-v and no x, y, z)
- 2D diffusion: variable in x, y, t (can also include z for 3D case)

Other Apps illustrate a wide range of applications

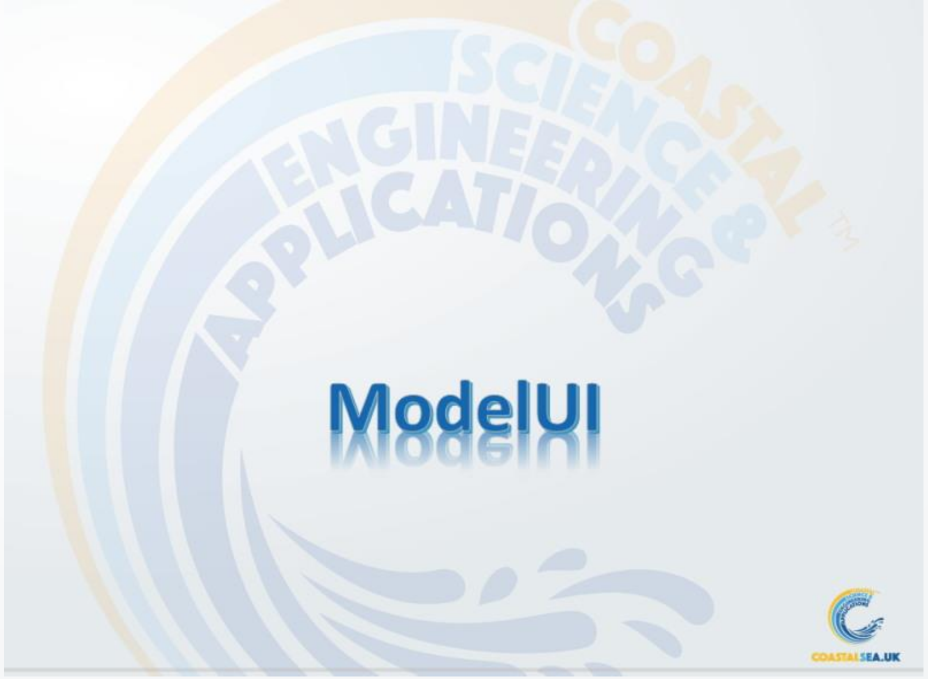
- Data analysis – CoastalTools, SedTools
- Models – Asmita, ChannelForm, WaveRayModel, CSTmodel
- Analysis – ModelSkill
- Design – MRBreach

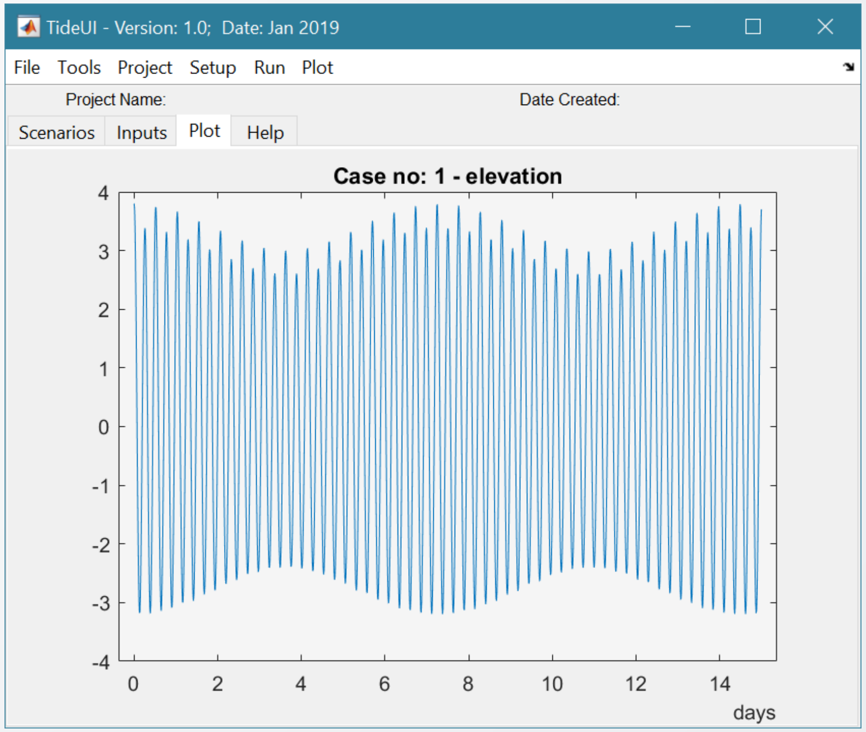
Summary

- The *muitoolbox* can handle multiple variables, with multiple dimensions: e.g. time and x, y, z.
- It is available as Open Source
- Same site as ASMITA and CoastalTools

  www.coastalsea.uk
- Manuals with worked examples available for all demonstration models and Apps

COASTALSEA.UK





Getting Started

Model to generate a simple tidal time-series

Simple tidal signal model

Existing function to compute a tidal water level time series using main constituents scaled to required tidal amplitude.

Function name: *simple_tide.m*

Input is a structure for:

- duration, time interval, mean tidel level, tidal amplitude, elevation phase, velocity amplitude, velocity phase, and amplitudes for M2, S2 and O1 tidal constituents

Output is a structure for

- time, elevation, vertical velocity and horizontal velocity

Install toolboxes and copy templates

Download and install *dstoolbox* and *muitoolbox* (see next slide)
Create a new working folder
Copy *~BlankAppFolders.zip* file from …/muitoolbox/muitemplates
Unzip the contents into the working folder
Copy following templates from …/muitoolbox/muitemplates folder:

Source	Rename as
UseUI_template.m	TideUI.m
ParamInput_template.m	TideParams.m
Model_template.m	TideModel.m

Note:
UseUI_template.m allows the UI to be customized
UseModelUI_template.m uses ModelUI as the user interface and requires ModelUI App to be installed

Installing the Toolboxes

The toolboxes can be installed using the

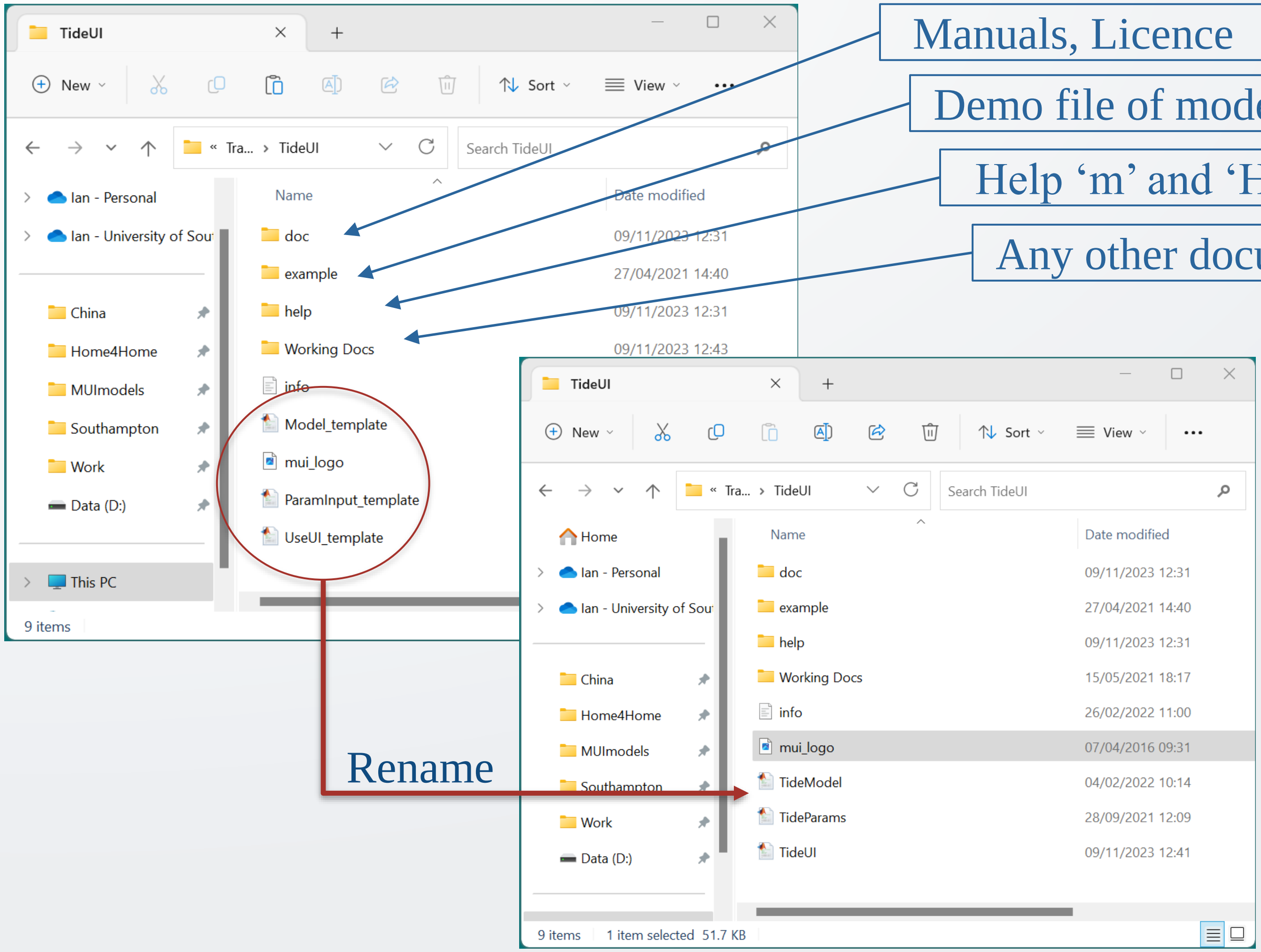
Add-Ons>Manage Add-Ons option on Home tab of Matlab™

Alternatively, right-click the mouse on the ‘mltbx’ files and select install.

All the folder paths are initialised upon installation and the location of the code is also handled by Matlab™.

The location of the code can be accessed using the options in the Manage Add-Ons UI (option on Home tab)

App folder structure (using *~BlankAppFolders.zip* and template files)



TideUI

Name	Date modified
doc	09/11/2023 12:31
example	27/04/2021 14:40
help	09/11/2023 12:31
Working Docs	09/11/2023 12:43
info	
Model_template	
mui_logo	
ParamInput_template	
UseUI_template	

9 items

TideUI

Name	Date modified
doc	09/11/2023 12:31
example	27/04/2021 14:40
help	09/11/2023 12:31
Working Docs	15/05/2021 18:17
info	26/02/2022 11:00
mui_logo	07/04/2016 09:31
TideModel	04/02/2022 10:14
TideParams	28/09/2021 12:09
TideUI	09/11/2023 12:41

9 items 1 item selected 51.7 KB

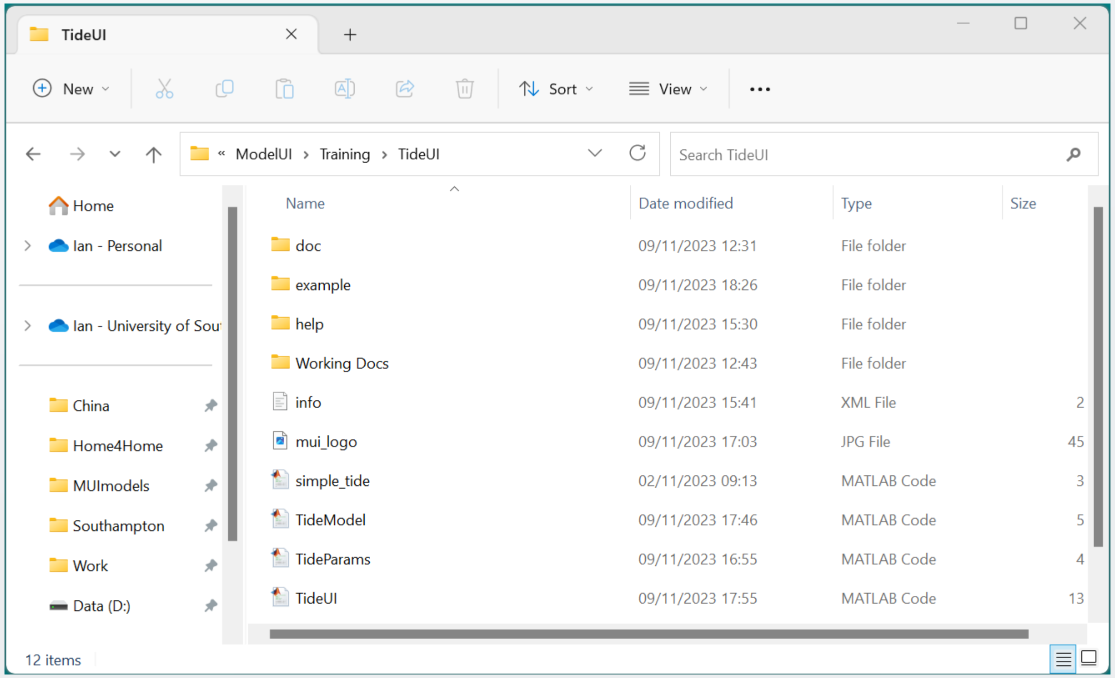
COASTALSEA.UK

Add model to Working Folder

The model file can be found in the Working Docs folder extracted from the *~BlankAppFolders.zip* file.

Move '*simple_tide.m*' up to the working folder.

The working folder should now look like this:



Name	Date modified	Type	Size
doc	09/11/2023 12:31	File folder	
example	09/11/2023 18:26	File folder	
help	09/11/2023 15:30	File folder	
Working Docs	09/11/2023 12:43	File folder	
info	09/11/2023 15:41	XML File	2
mui_logo	09/11/2023 17:03	JPG File	45
simple_tide	02/11/2023 09:13	MATLAB Code	3
TideModel	09/11/2023 17:46	MATLAB Code	5
TideParams	09/11/2023 16:55	MATLAB Code	4
TideUI	09/11/2023 17:55	MATLAB Code	13

UseUI_template.m >> TideUI.m

To change the Class name throughout the file:

In *classdef* select *UseUI_template*

Editor tab>Find

‘replace with’: *TideUI* > Replace All

Sections of code that need editing are indicated by comments such as: % << Edit to classname

For this application the following functions need to be edited:

- setMUI
- setMenus and setTab
- setTabProperties and setTabActions
- setupMenuOptions
- runMenuOptions

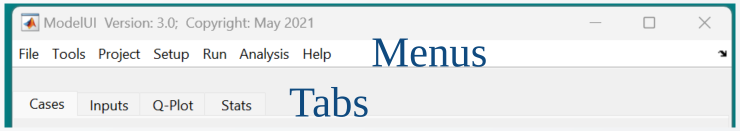
setMUI

```
function obj = setMUI(obj)
        %initialise standard figure and menus
        % classes required to run model, format:
        % obj.ModelInputs.<model classname> = {'Param_class1',Param_class2',etc}
        %                                         % << Edit to model and input parameters classnames
        obj.ModelInputs.Model_template = {'ParamInput_template'};
        %tabs to include in DataUIs for plotting and statistical analysis
        %select which of the options are needed and delete the rest
        %Plot options: '2D','3D','4D','2DT','3DT','4DT'
        obj.DataUItabs.Plot = {'2D','3D','4D','2DT','3DT','4DT'};
        %Statistics options: 'General','Timeseries','Taylor','Intervals'
        obj.DataUItabs.Stats = {'General','Timeseries','Taylor','Intervals'};
        modelLogo = 'mui_logo.jpg'; %default splash figure - edit to alternative
        initialiseUI(obj,modelLogo); %initialise menus and tabs
end
```

- Replace *Model_template* with *TideModel*
- Replace *ParamInput_template* with *TideParams*
- Delete unwanted Plot and Stats UIs
  - For TideUI only {'*2D*'} and {'*General*'} are needed

setMenus

The setMenus function defines the menu structure in the UI. These are in order from left to right on the UI



Sample code for Setup menu to define menu lists, callback functions and separators (if required):

Top level menu name

Sub-menu list

Level 1

```
%% Setup menu -------------------------------------------------
menu.Setup(1).List = {'Import Data','Input Parameters','Run Parameters','Model Constants'};
menu.Setup(1).Callback = [{'gcbo;'},repmat({@obj.setupMenuOptions},[1,3])];
%add separators to menu list (optional - default is off)
menu.Setup(1).Separator = {'off','off','off','on'};     %separator precedes item
```

Cell array of callback functions

- {'gcbo'} passes control to next menu level
- functions pass control to other functions in TideUI

Level 2

```
% submenu for Import Data (if these are changed need to edit
% loadMenuOptions to be match)
menu.Setup(2).List = {'Load','Add','Delete','Quality Control'};
menu.Setup(2).Callback = repmat({@obj.loadMenuOptions},[1,4]);
```

This adds menus to load data from a file

Note that at each level List, Callback and Separator cell arrays must be the same size

setTabs

Default code shows the definition of 4 tabs and two sub-tabs on the Stats tab.

```
function [tabs,subtabs] = setTabs(obj)
          %define main tabs and any subtabs required. struct field is
          %used to set the uitab Tag (prefixed with sub for subtabs).
          %Order of assignment to struct determines order of tabs in
          figure.
          %format for tabs:
          % tabs.<tagname> = {<tab label>,<callback function>};
          %format for subtabs:
          % subtabs.<tagname>(i,:) = {<subtab label>,<callback
          function>};
          %where <tagname> is the struct fieldname for the top level
          tab.
          tabs.Cases = {' Cases ',@obj.refresh}; % << Edit tabs to suit
          model
          tabs.Inputs = {' Inputs ',@obj.InputTabSummary};
          tabs.Plot = {' Q-Plot ',@obj.getTabData};
          tabs.Stats = {' Stats ',''};
          subtabs.Stats(1,:) = {' Descriptive ',@obj.setTabAction};
          subtabs.Stats(2,:) = {' Extremes ',@obj.setTabAction};
          %if subtabs are not required eg for Stats
          % tabs.Stats = {' Stats ',@obj.setTabAction};
          % subtabs = [];
end
```

For the TideUI application the Stats sub-tabs are not required and can be deleted (highlighted text). The text below then needs to be *Uncommented* to include just the top-level Stats tab

COASTALSEA.UK

setTabProperties

Defines where model input parameters are displayed.

```
function props = setTabProperties(~)
        %define the tab and position to display class data tables
        %props format: {class name, tab tag name, position, column width, table title}
        % position and column widths vary with number of parameters
        % (rows) and width of input text and values. Indicative positions:
        % top left [0.95,0.48]; top right [0.95,0.97]
        % bottom left [0.45, 0.48]; bottom right [0.45,0.97]
        %                                                  << Edit input properties classnames
        props = {... % << Add additional inputs and adjust layout
        'ParamInput_template','Inputs',[0.95,0.48],{180,60},'Input parameters:';...
        'ParamInput_template','Inputs',[0.45,0.48],{180,60},'Run parameters:'};
end
```

- Replace *ParamInput_template* with *TideParams*
- Delete second line (highlighted) but not cell end bracket

setTabActions

Defines bespoke behaviour of tabs (if required)

```
function setTabAction(~,src,cobj)
        %function required by muiModelUI and sets action for selected tab (src)
        msg = 'No results to display';
        switch src.Tag % << Edit match tab requirements
                case 'Plot’
                        tabPlot(cobj,src);
                case {'Descriptive','Extremes’}          %or 'Stats' if no subtabs
                        cobj = getClassObj(obj,'mUI','Stats',msg);
                        if isempty(cobj), return; end
                        tabStats(cobj,src);
        end
end
```

For the TideUI application the Stats sub-tabs are not required (highlighted text) and need to be replaced with ‘*Stats*’

COASTALSEA.UK

setMenuOptions

Edit calls for setup menu

```
function setupMenuOptions(obj,src,~)
%callback functions for data input
switch src.Text
        case 'Input Parameters' % << Edit to call Parameter Input class
                ParamInput_template.setInput(obj);
                %update tab display with input data
                tabsrc = findobj(obj.mUI.Tabs,'Tag','Inputs’);
                InputTabSummary(obj,tabsrc);
        case 'Run Parameters' % << Edit to call Data Import class
                ParamInput_template.setInput(obj);
                %update tab display with input data
                tabsrc = findobj(obj.mUI.Tabs,'Tag','Inputs’);
                InputTabSummary(obj,tabsrc);
        case 'Model Constants’
                obj.Constants = setInput(obj.Constants);
        end
end
```

- Replace *ParamInput_template* with *TideParams*
- Delete *Run Parameters* case (highlighted)

COASTALSEA.UK

runMenuOptions and Help

Edit calls for run menu

```
function runMenuOptions(obj,src,~)
        %callback functions to run model
        switch src.Text
                case 'Run Model' % << Edit to call Model class
                        Model_template.runModel(obj);
                case 'Derive Output’
                        obj.mUI.Manip = muiManipUI.getManipUI(obj);
        end
end
```

- Replace *Model_template* with *TideModel*

Help

```
%% Help menu ------------------------------------------------------
function Help(~,~,~)
        doc tideui                      % << Edit to documentation name if available
end
```

- Note that model name is in lower case

COASTALSEA.UK

Test the UI

In the Command Window type:

```
>>TideUI;
```

You should now have a working GUI!

It just does not do anything yet!!

ParamInput_template >> TideParams.m

To change the Class name throughout file:

In *classdef* select *ParamInput_template*

Editor tab>Find

‘replace with’: *TideParams* > Replace All

Sections of code that need editing are indicated by comments such as: `% << Edit to classname`

As a minimum the following needs to be edited:

- Class properties (next slides)

TideParams.m – setting the class properties

```
properties (Hidden)
        %abstract properties in muiPropertyUI to define input parameters
        PropertyLabels = {'Parameter 1 short description',... % << Edit to property descriptions
                          'Parameter 2 short description',...
                          'etc'};
        %abstract properties in muiPropertyUI for tab display
        TabDisplay %structure defines how the property table is displayed
end

properties
        Parameter_1      %definition of parameter 1 % << Edit to property names
        Parameter_2      %definition of parameter 2
        etc              %there should be as many properties as labels
end
```

Edit highlighted text to define model variables

Model Input Parameters

Variable	Description
Duration	duration (days)
Tinterval	time interval for simulation (mins)
z0	mean tide level to ordnance datum (mOD)
TidalAmp	tidal elevation amplitude (m)
ElevPhase	phase of elevation (ie k.x) (rads)
VelocityAmp	tidal velocity amplitude (m/s)
VelocityPhase	phase of velocity (ie k.x+phi) (rads)
aM2	M2 tidal amplitude (m)
aS2	S2 tidal amplitude (m)
aO1	O1 tidal amplitude (m)

TideParams.m - properties

```
properties (Hidden)
        %abstract properties in muiPropertyUI to define input parameters
        PropertyLabels = {'Duration (d)','Time interval (mins)'...
                          'Mean tide level (mOD)',...
                          'Tidal amplitude (m)',...
                          'Elevation phase (deg)',...
                          'Velocity amplitude (m/s)',...
                          'Velocity phase (deg)',...
                          'M2 tidal amplitude (m)',...
                          'S2 tidal amplitude (m)',...
                          'O1 tidal amplitude (m)'};
        %abstract properties in muiPropertyUI for tab display
        TabDisplay      %structure defines how the property table is displayed
end

properties
        Duration        %duration (days)
        Tinterval       %time interrval for simulation (mins)
        z0              %mean tidel level to ordnance datum (mOD)
        TidalAmp        %tidal elevation amplitude (m)
        ElevPhase       %phase of elevation (ie k.x) (rads)
        VelocityAmp     %tidal velocity amplitude (m/s)
        VelocityPhase   %phase of velocity (ie k.x+phi) (rads)
        aM2             %M2 tidal amplitude (m)
        aS2             %S2 tidal amplitude (m)
        aO1             %O1 tidal amplitude (m)
end
```

Test the Data Input

In the Command Window type:

```
>>TideUI;
```

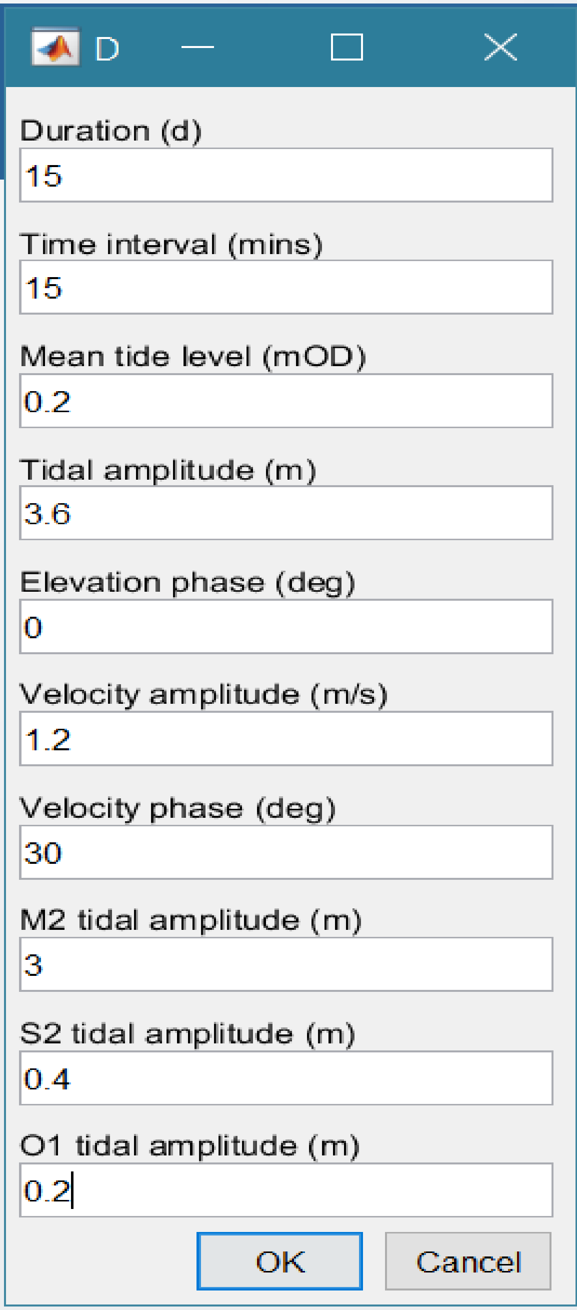
In the model UI select

Setup>Input Data>Model Data

Enter some data

See the input data on the *Inputs* tab

But it still does not do anything!



Duration (d)	15
Time interval (mins)	15
Mean tide level (mOD)	0.2
Tidal amplitude (m)	3.6
Elevation phase (deg)	0
Velocity amplitude (m/s)	1.2
Velocity phase (deg)	30
M2 tidal amplitude (m)	3
S2 tidal amplitude (m)	0.4
O1 tidal amplitude (m)	0.2

Model_template >> TideModel.m

To change the Class name throughout file:

In *classdef* select *Model_template*

Editor tab>Find

‘replace with’: *TideModel* > Replace All

Sections of code that need editing are indicated by comments such as: % << Edit to classname

As a minimum the following functions need to be edited:

- runModel
- modelDSproperties

Model Output Parameters

Name	Description	Label	Units
h	Elevation	Elevation (mOD)	mOD
uV	Vertical velocity	Vertical velocity (m/s)	m/s
uH	Horizontal velocity	Horizontal velocity (m/s)	m/s

simple_tide returns a struct with fields:

results.t – time (s)

results.z – elevation relative to datum (mAD)

results.dz – vertical velocity (m/s)

results.u – horizontal velocity (m/s)

TideModel - runModel function

Modify template as follows:

- handle to input parameters class
- call to simple_tide model
- format model time data
- convert model output to a cell array of variables (first cell is time so exclude in assignment)
- assign to dstable
- add additional meta data
- assign results to catalogue

```
%--------------------------------------------------------------------------
% Model code <<INSERT MODEL CODE or CALL MODEL>>
%--------------------------------------------------------------------------
inp = mobj.Inputs.TideParams;
res = simple_tide(inp);
%now assign results to object properties
modeltime = seconds(res.t); %durataion data for rows
modeltime.Format = 'd';
%--------------------------------------------------------------------------
% Assign model output to a dstable using the defined dsproperties meta-data
%--------------------------------------------------------------------------
%each variable should be an array in the 'results' cell array
%if model returns single variable as array of doubles, use {results}
res = struct2cell(res);
dst = dstable(res{2:end},'RowNames',modeltime,'DSproperties',dsp);
%--------------------------------------------------------------------------
% Save results
%--------------------------------------------------------------------------
%assign metadata about model
dst.Source = metaclass(obj).Name;
dst.MetaData = sprintf('Model run for %d days',inp.Duration);
%save results
setDataSetRecord(obj,mobj.Cases,dst,'model');
getdialog('Run complete');
end
```

TideModel - modelDSproperties function

Modify template as shown:

A ‘dstable’ requires the properties being saved to be defined by a set of ‘dsproperties’. The *dsproperty* structure has fields for the **Variables**, the **Row** (usually time) and any other **Dimensions**. Each of these has a sub-structure with fields for the Name, Description, Unit, Label, and Format. The cell arrays assigned to this set of fields must be the same size for each of the top-level fields. In the example, Variables has a length of 3, whereas Row has a length of 1 and Dimensions is empty.

```
function dsp = modelDSproperties(~)
        %define a dsproperties struct and add the model metadata
        dsp = struct('Variables',[],'Row',[],'Dimensions',[]);
        %define each variable to be included in the data table and any
        %information about the dimensions. dstable Row and Dimensions can
        %accept most data types but the values in each vector must be unique
        %struct entries are cell arrays and can be column or row vectors
        dsp.Variables = struct(...
                'Name',{'h','uV','uH'},...
                'Description',{'Elevation','Vertical velocity',...
                'Horizontal velocity'},...
                'Unit',{'mAD','m/s','m/s'},...
                'Label',{'Elevation (mOD)','Velocity (m/s)','Velocity (m/s)'},...
                'QCflag',{'model','model','model'});
        dsp.Row = struct(...
                'Name',{'Time'},...
                'Description',{'Time'},...
                'Unit',{'d'},...
                'Label',{'Time (d)'},...
                'Format',{'d'});
        dsp.Dimensions = struct(...
                'Name',{''},...
                'Description',{''},...
                'Unit',{''},...
                'Label',{''},...
                'Format',{''});
end
```

COASTALSEA.UK

Test the model

In the Command Window type:

```
>>TideUI;
```

In the model UI select

Setup>Input Parameters

- Enter some data
- See the input data on the *Inputs* tab

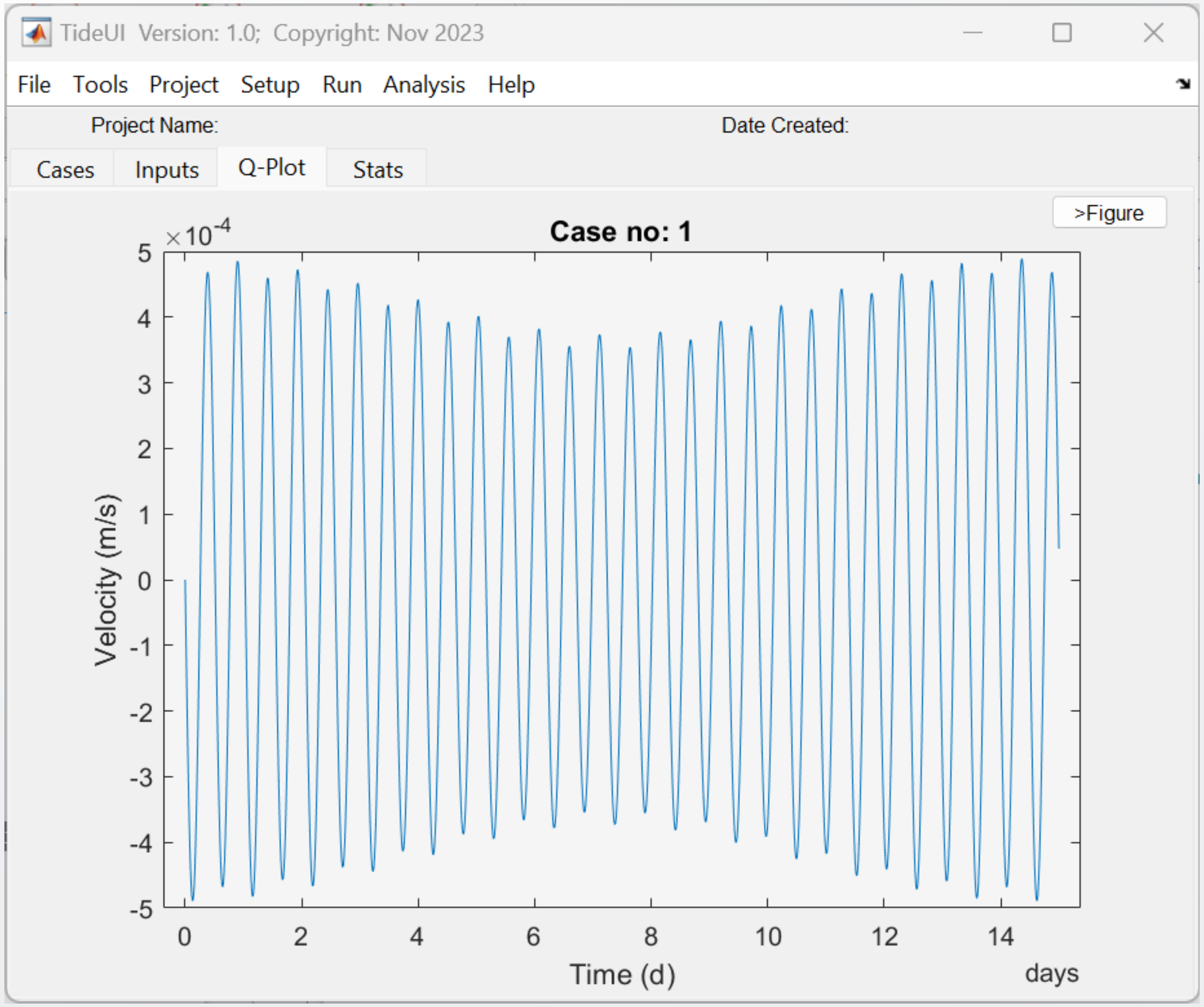
Run>Run Model

- When prompted, give the run a name

View the elevation results on the Plot tab

Use Analysis>Plots menu to produce a range of plots

View model output on Q-Plot tab


TideUI Version: 1.0; Copyright: Nov 2023
File Tools Project Setup Run Analysis Help
Project Name:
Date Created:
Cases Inputs Q-Plot Stats
>Figure
Case no: 1
×10^-4
5
4
3
2
1
0
-1
-2
-3
-4
-5
Velocity (m/s)
0
2
4
6
8
10
12
14
Time (d)
days

COASTALSEA.UK

Extension to load data

Copy *dataimport_format_template* from muitemplates folder

Rename as *tideui_import.m*

To change the function name throughout the file:

- In function definition select *dataimport_format_template*
- Editor tab>Find
  - ‘replace with’: *tideui_import* > Replace All

For this application, the following functions need to be edited:

- getData
- readInputData
- setDSproperties

COASTALSEA.UK

getData and readInputData

Edit the default template

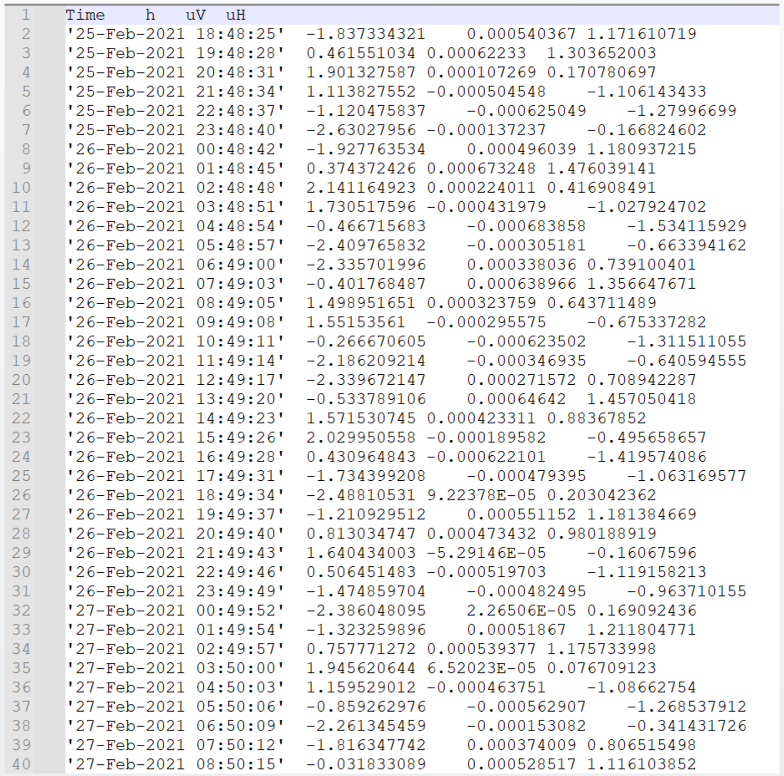
- see next slide
- depends on format of data in source file
- format specification to read each line of data

```
function dst = getData(~,filename)
        %read and load a data set from a file
        [data,~] = readInputData(filename);
        if isempty(data), dst = []; return; end

        %set metadata
        dsp = setDSproperties;

        %code to parse input data and assign to vardata
        mdat = datetime(strip(data{1},'"'));
        mtim = datetime(strip(data{2},'"'));
        myDatetime = mdat + timeofday(mtim);

        % concatenate date and time
        myDatetime = myDatetime-myDatetime(1);
        myDatetime.Format = dsp.Row.Format;
        vardata = data(3:end); %sorted data to be loaded

        %load the results into a dstable
        dst = dstable(vardata{:},'RowNames',myDatetime,'DSproperties',dsp);
end
%%
function [data,header] = readInputData(filename)
        %read tidal elevation data.
        dataSpec = '%s %s %f %f %f';
        nhead = 1; %number of header lines
        [data,header] = readinputfile(filename,nhead,dataSpec);    %see muifunctions
end
```

COASTALSEA.UK

setDSproperties

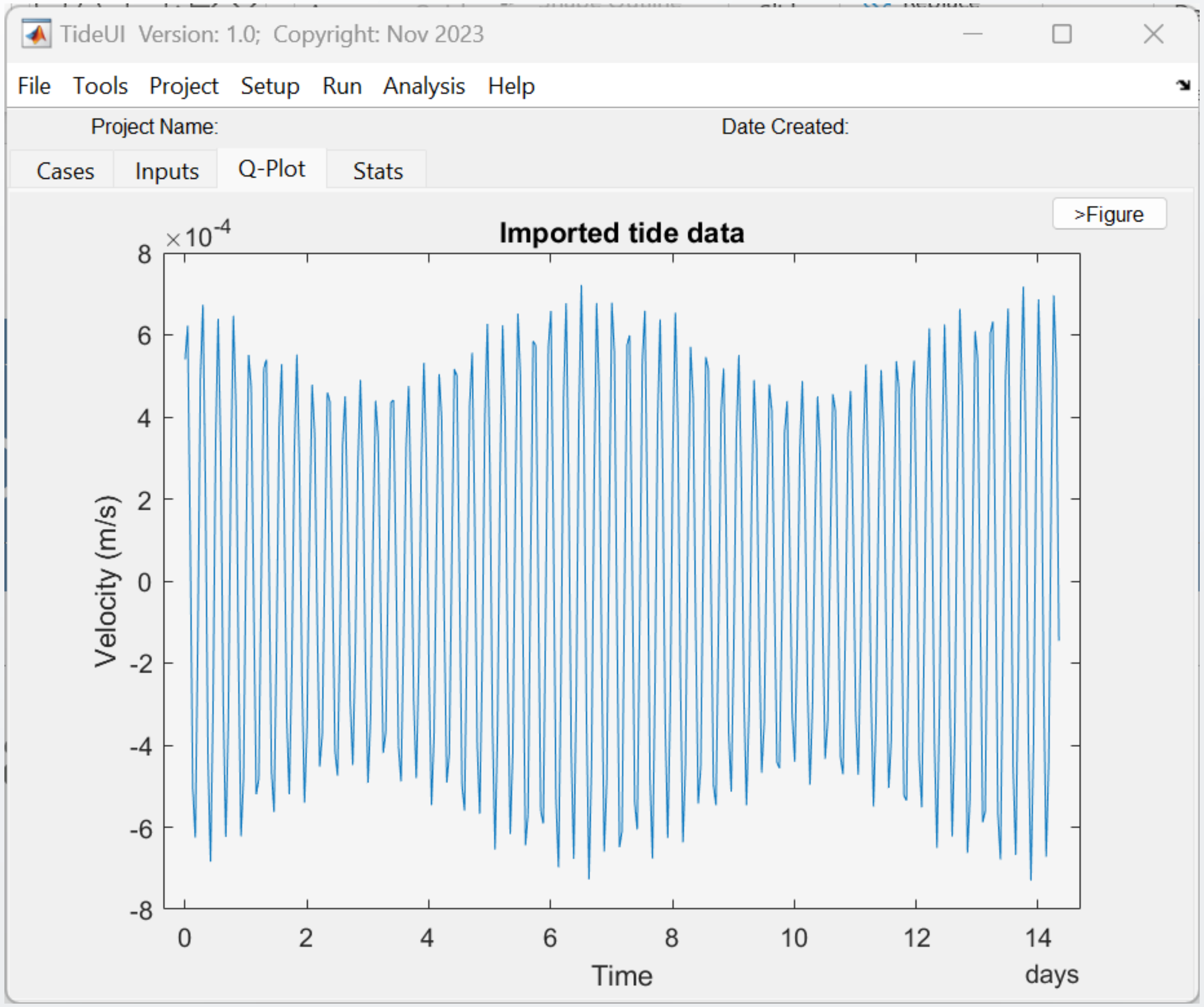
File with data/time, elevation, vertical and horizontal velocity



```
Time     h    uV   uH
'25-Feb-2021 18:48:25'  -1.837334321    0.000540367 1.171610719
'25-Feb-2021 19:48:28'  0.461551034 0.00062233  1.303652003
'25-Feb-2021 20:48:31'  1.901327587 0.000107269 0.170780697
'25-Feb-2021 21:48:34'  1.113827552 -0.000504548    -1.106143433
'25-Feb-2021 22:48:37'  -1.120475837    -0.000625049    -1.27996699
'25-Feb-2021 23:48:40'  -2.63027956 -0.000137237    -0.166824602
'26-Feb-2021 00:48:42'  -1.927763534    0.000496039 1.180937215
'26-Feb-2021 01:48:45'  0.374372426 0.000673248 1.476039141
'26-Feb-2021 02:48:48'  2.141164923 0.000224011 0.416908491
'26-Feb-2021 03:48:51'  1.730517596 -0.000431979    -1.027924702
'26-Feb-2021 04:48:54'  -0.466715683    -0.000683858    -1.534115929
'26-Feb-2021 05:48:57'  -2.409765832    -0.000305181    -0.663394162
'26-Feb-2021 06:49:00'  -2.335701996    0.000338036 0.739100401
'26-Feb-2021 07:49:03'  -0.401768487    0.000638966 1.356647671
'26-Feb-2021 08:49:05'  1.498951651 0.000323759 0.643711489
'26-Feb-2021 09:49:08'  1.55153561  -0.000295575    -0.675337282
'26-Feb-2021 10:49:11'  -0.266670605    -0.000623502    -1.311511055
'26-Feb-2021 11:49:14'  -2.186209214    -0.000346935    -0.640594555
'26-Feb-2021 12:49:17'  -2.339672147    0.000271572 0.708942287
'26-Feb-2021 13:49:20'  -0.533789106    0.00064642  1.457050418
'26-Feb-2021 14:49:23'  1.571530745 0.000423311 0.88367852
'26-Feb-2021 15:49:26'  2.029950558 -0.000189582    -0.495658657
'26-Feb-2021 16:49:28'  0.430964843 -0.000622101    -1.419574086
'26-Feb-2021 17:49:31'  -1.734399208    -0.000479395    -1.063169577
'26-Feb-2021 18:49:34'  -2.48810531 9.22378E-05 0.203042362
'26-Feb-2021 19:49:37'  -1.210929512    0.000551152 1.181384669
'26-Feb-2021 20:49:40'  0.813034747 0.000473432 0.980188919
'26-Feb-2021 21:49:43'  1.640434003 -5.29146E-05    -0.16067596
'26-Feb-2021 22:49:46'  0.506451483 -0.000519703    -1.119158213
'26-Feb-2021 23:49:49'  -1.474859704    -0.000482495    -0.963710155
'27-Feb-2021 00:49:52'  -2.386048095    2.26506E-05 0.169092436
'27-Feb-2021 01:49:54'  -1.323259896    0.00051867  1.211804771
'27-Feb-2021 02:49:57'  0.757771272 0.000539377 1.175733998
'27-Feb-2021 03:50:00'  1.945620644 6.52023E-05 0.076709123
'27-Feb-2021 04:50:03'  1.159529012 -0.000463751    -1.08662754
'27-Feb-2021 05:50:06'  -0.859262976    -0.000562907    -1.268537912
'27-Feb-2021 06:50:09'  -2.261345459    -0.000153082    -0.341431726
'27-Feb-2021 07:50:12'  -1.816347742    0.000374009 0.806515498
'27-Feb-2021 08:50:15'  -0.031833089    0.000528517 1.116103852
```

```matlab
function dsp = setDSproperties()
        %define the variables in the dataset
        %define the metadata properties for the demo data set
        dsp = struct('Variables',[],'Row',[],'Dimensions',[]);
        %define each variable to be included in the data table and any
        %information about the dimensions. dstable Row and Dimensions can
        %accept most data types but the values in each vector must be unique

        %struct entries are cell arrays and can be column or row vectors
        dsp.Variables = struct(...
                'Name',{'h','u','v'},...
                'Description',{'Elevation','Vertical velocity','Horizontal velocity'},...
                'Unit',{'mAD','m/s','m/s'},...
                'Label',{'Elevation (m)','Velocity (m/s)','Velocity (m/s)'},...
                'QCflag',{'data','data','data'});
        dsp.Row = struct(...
                'Name',{'Time'},...
                'Description',{'Time'},...
                'Unit',{'d'},...
                'Label',{'Time'},...
                'Format',{'d'});
        dsp.Dimensions = struct(...
                'Name',{''},...
                'Description',{''},...
                'Unit',{''},...
                'Label',{''},...
                'Format',{''});
end
```

View imported data on Q-Plot tab



TideUI Version: 1.0; Copyright: Nov 2023
File Tools Project Setup Run Analysis Help
Project Name:
Date Created:
Cases Inputs Q-Plot Stats
>Figure
Imported tide data
×10-4
8
6
4
2
0
-2
-4
-6
-8
Velocity (m/s)
0
2
4
6
8
10
12
14
Time
days

Alternative is to set up a new data class

The *DataImport_formatfiles_template* is a class that uses the *dataimport_format_template* and allows the user to select from multiple format files that handle different file formats. Simply rename the file and classdef and edit the file list to match the new *import_format_file* names.

OR

The *DataImport_template* is a template for a new data import class. To use this option, rename the file, classdef and object calls where indicated (% << Edit to classname). Then edit the readInputData and setDSproperties functions, as shown for the *input_format_file* option above.

In both cases, edit call in the main UI to the new class name

```
function loadMenuOptions(obj,src,~)
        %callback functions to import data
        classname = 'muiUserData';
```

e.g. in TideUI edit highlighted text to <data import classname>

COASTALSEA.UK

Converting the model to an App

Matlab provide the tools need to package models as an App. The default template structure is designed to make this easy to do. The main steps are as follows:

1. Edit the info.xml and create project file (*.prj)
2. Create the project help files and related utilities
3. Draft a manual - pdf format (if required)
4. Add documentation index
5. Package App

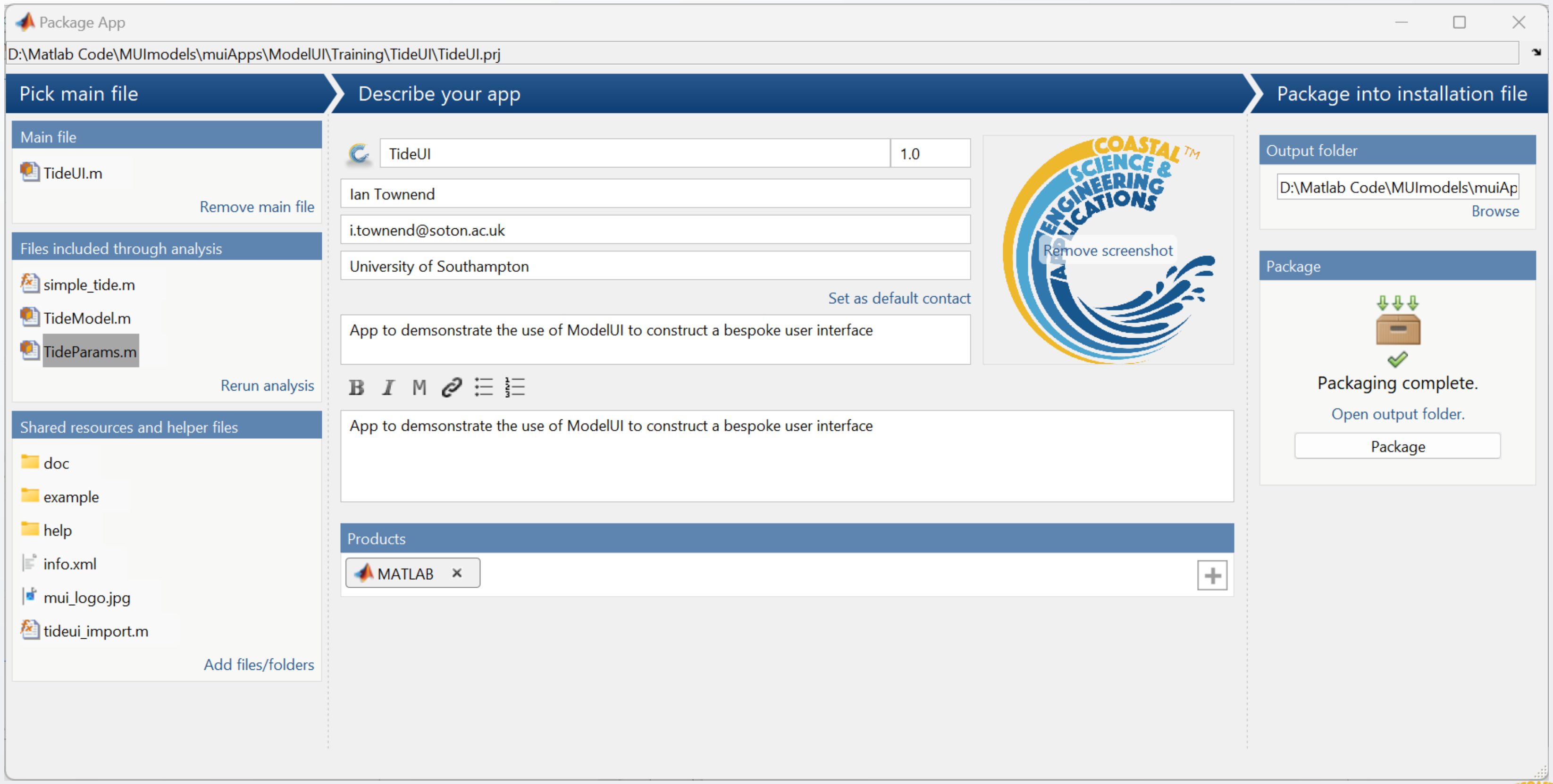
Creating an App Package (step 1)

From the working folder

- In ‘info.xml’ file change ‘ModelUI’ to App name e.g., ‘TideUI’
- Add project file:
  - On Matlab Apps tab select Package App.
  - Complete the form to describe the App.
  - Select the main file (‘TideUI’). This will populate the Files included list.
  - Scroll down to shared resources and add
    - ‘doc’, ‘help’ and ‘example’ folders
    - info.xml
    - mui_logo.jpg
    - tideui_import.m
  - From the Files included list delete everything except the App specific classes and functions (in this case just keep TideModel.m, TideParams.m and simple_tide.m). To do this select files and left-click mouse and select remove.

Creating an App Package (step 1)

The package UI should now contain the following files:



Package App

D:\Matlab Code\MUImodels\muiApps\ModelUI\Training\TideUI\TideUI.prj

Pick main file | Describe your app | Package into installation file

Main file
- TideUI.m

Remove main file

Files included through analysis
- simple_tide.m
- TideModel.m
- TideParams.m

Rerun analysis

Shared resources and helper files
- doc
- example
- help
- info.xml
- mui_logo.jpg
- tideui_import.m

Add files/folders

TideUI 1.0

Ian Townend

i.townend@soton.ac.uk

University of Southampton

Set as default contact

App to demsonstrate the use of ModelUI to construct a bespoke user interface

Remove screenshot

App to demsonstrate the use of ModelUI to construct a bespoke user interface

Products

MATLAB ×

Output folder

D:\Matlab Code\MUImodels\muiAp

Browse

Package

Packaging complete.

Open output folder.

Package

Help (1) – files in *help* folder (step 2)

Online documentation is defined in the ‘m’ files to be found in the ‘help’ folder. When *published*, html files with same name are created in the ‘html’ sub-folder.

- Edit the name of the file model_template.m to the model class name in lower case (tideui.m in this case).
- Edit the other file names to replace ‘model’ with some model specific name (e.g., ‘tui’ in this case).
- Edit the content of each file to document the model. The text and layout in the templates provide some indicative content.
  - for details of the markup format see: Matlab Help > Publishing Markup
  - for details of how the online help is configured see: Matlab Help > Display Custom Documentation
- In each help file edit references to the manual and example folder, replacing ‘xxx’ with the help prefix (e.g., ‘tui’ in this case).
- From the Matlab Publish tab, publish the files to the ‘html’ sub-folder.

To add a pdf manual, create a pdf with the same name as the App model class (*TideUI* in this case) and place it in the App ‘doc’ folder.

Help (2) – files in *html* folder (step 2)

In …/help/html folder

- Change file names of 'xxx_example_folder.m' and xxx_open_manual.m', replacing xxx with help prefix (e.g. 'tui' in this case)
- In these files:
  - update the function name to match the file name.
  - change the *appname = 'ModelName'* to the name of the model/App ('TideUI' in this case).
  - In the xxx_open_manual.m file model modify *fpath* to the correct file name for the manual (e.g., 'TideUI manual.pdf').
- In 'helptoc.xml' file change all *.html file names to the help file names (e.g., 'tui_menus') and the labels to the App name (e.g., 'TidalUI'):

```
<tocitem target="tidalui.html">TidalUI
```

Help (3) - related utilities (steps 2 and 3)

If a manual is to be included (e.g.; to explain use cases and document the basis of the model, etc), this needs to be placed as pdf file in the *doc* sub-folder. The file name must be as defined in the xxx_open_manual.m, e.g.

'TideUI manual.pdf'

Add a demo project mat file to the 'example' folder.

- Note: if the folder is empty, it is not installed as part of the App

Draft a manual to document how the App works and the background information on the model implementation

Create the App (steps 4 and 5)

Once help files have been setup and edited

Set Matlab environment path to include the working folder.

- Add documentation index using:

```
>> builddocsearchdb('<full path name>\help\html')
```

- Build app by opening project file and selecting the '*Package*' button.
- Install App.

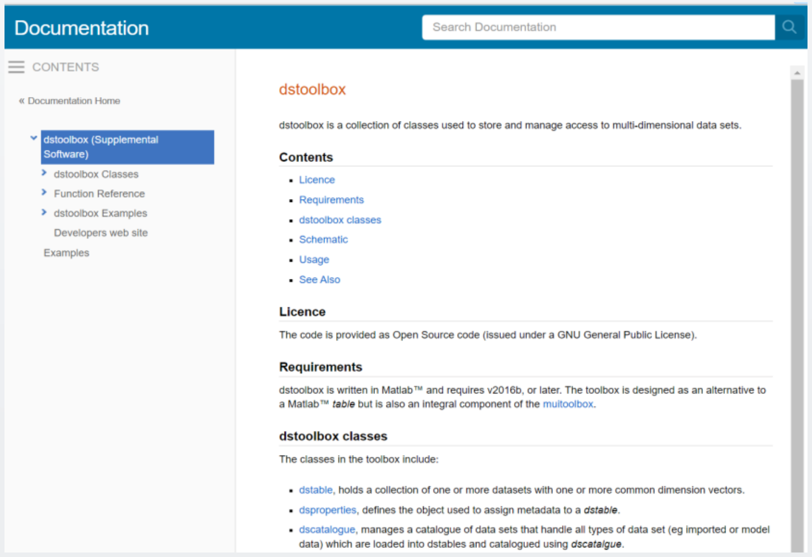
The App should now be available to run from the Matlab APPS tab

Further information

Toolboxes and Apps can be downloaded from

www.coastalsea.uk

When dstoolbox and muitoolbox are installed, further documentation can be accessed using the Matlab^{TM} Supplemental Software documentation.



dstoolbox

dstoolbox is a collection of classes used to store and manage access to multi-dimensional data sets.

Contents

- Licence
- Requirements
- dstoolbox classes
- Schematic
- Usage
- See Also

Licence

The code is provided as Open Source code (issued under a GNU General Public License).

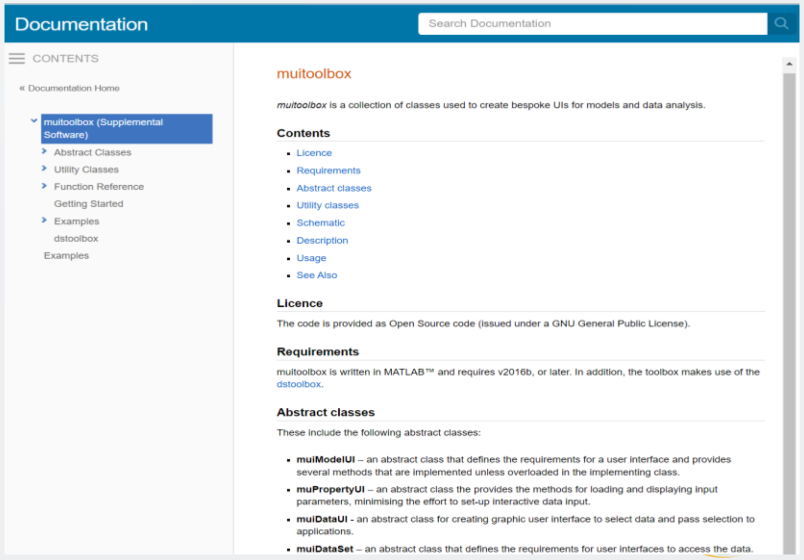
Requirements

dstoolbox is written in Matlab™ and requires v2016b, or later. The toolbox is designed as an alternative to a Matlab™ *table* but is also an integral component of the muitoolbox.

dstoolbox classes

The classes in the toolbox include:

- dstable, holds a collection of one or more datasets with one or more common dimension vectors.
- dsproperties, defines the object used to assign metadata to a *dstable*.
- dscatalogue, manages a catalogue of data sets that handle all types of data set (eg imported or model data) which are loaded into dstables and catalogued using *dscatalgue*.



muitoolbox

muitoolbox is a collection of classes used to create bespoke UIs for models and data analysis.

Contents

- Licence
- Requirements
- Abstract classes
- Utility classes
- Schematic
- Description
- Usage
- See Also

Licence

The code is provided as Open Source code (issued under a GNU General Public License).

Requirements

muitoolbox is written in MATLAB™ and requires v2016b, or later. In addition, the toolbox makes use of the dstoolbox.

Abstract classes

These include the following abstract classes:

- **muiModelUI** – an abstract class that defines the requirements for a user interface and provides several methods that are implemented unless overloaded in the implementing class.
- **muPropertyUI** – an abstract class the provides the methods for loading and displaying input parameters, minimising the effort to set-up interactive data input.
- **muiDataUI** - an abstract class for creating graphic user interface to select data and pass selection to applications.
- **muiDataSet** – an abstract class that defines the requirements for user interfaces to access the data.

COASTALSEA.UK

The dstoolbox

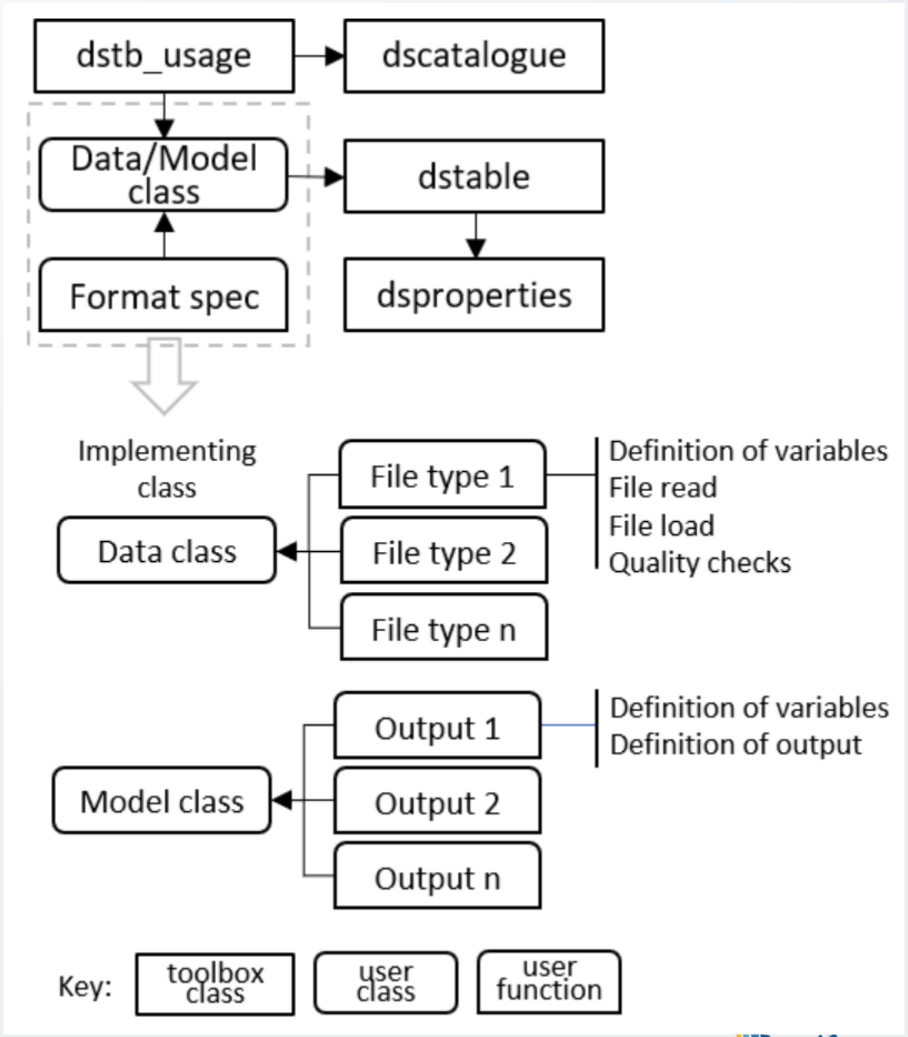
The *dstoolbox* is designed to store and manage multi-dimensional data sets, including meta-data of the variables and all dimensions and manage access to a collection of classes that hold data sets using a catalogue.

In the schematic outline *dstb_usage* is a class to illustrate how *dstable*, *dsproperties* and *dscatalogue* are used.

Data are loaded into a *dstable* with relevant metadata added to the table and made accessible using *dsproperties*. Each time a class adds data a record is added using *dscatalogue*.

The 'Format Spec' user functions, shown in the upper part of the figure, are implemented with functions, indicated by 'File Type' and 'Output Type', shown in the lower part of the figure.

These define the meta-data of the data set being saved (and any input parameters, or details needed to read and load data from a file, depending on the application).



COASTALSEA.UK

The muitoolbox

The purpose of the *muitoolbox* is to minimise the effort in creating or prototyping an interface for a model or data analysis tool.

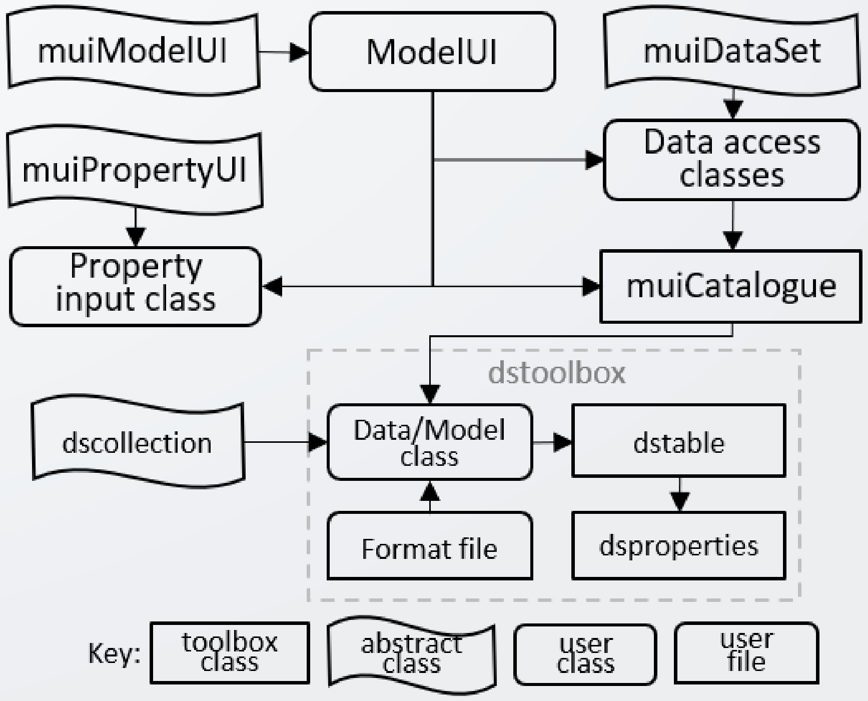
Creating a new model requires 3 components to be defined, namely the interface (ModelUI in the above illustration), one or more classes to manage the input of model parameters (if required) and the classes to hold imported data, or running a model and storing the output.

Central to this is the holding of input data in the Inputs property and accessing the data via the Cases property. In this context, Cases comprise a record of each Case and a dataset. The records are held in the Catalogue property and the dataset (an instance of the data or model class) in the DataSets property of *muiCatalogue*.

Each data or model class stores the dataset in the Data property, with additional information held in the RunData property (e.g. holding input parameters of a model run).

Any type of dataset can be stored in the Data property but when using the dstoolbox multidimensional data sets can be stored using *dstable* and a full set of meta-data attached using *dsproperties*.

The overall architecture and the properties that provide the links between one class and another are shown in the flow chart below



COASTALSEA.UK