# Introduction to the muitoolbox

#### Ian Townend





#### ModelUI 2D diffusion model



ModelUI Managed Realignment Breach Model



ModelUI Vertical Tidal Profile

ModelUI SimpleTide

2: Nov 23

ModelUI Inshore Waves

Contraction of the second



# 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.



# Common interface

#### Use the same underlying class interfaces





COASTALSEA.UK

## ModelUI Structure



COASTALSEA.UK

# Input UIs and Tab display generated automatically

承 Define prop			×
Bed friction coefficient	t		
0.0025			
Nikaurades friction co	efficien	t	
0.0025			
Tidal period (hours)			
12.4			
Roughness length			
0.0008			
Height of velocity mea	asureme	ent abo	ve bed (m)
2			
Velocity at depth z (m	/s)		
1.2			
Water depth (m)			
7			
		NK I	Cancol
			Cancer

Iools Project Setup Run	Plot	 	 	
Project Name:		Date Created:		
enarios Setup Plot Help				
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

承 Select Data for	Plotting			- 0	×	
XY XYZ						
Scenario Plot variab	ole	Test run no: 1 v uBowden v	→ ¥			
Scaling on	otions	Linear			🛃 Fi	igure 3: Results Plot — 🗆 🗙
county op		Lindu			File	Edit View Insert Tools Desktop Window Help
Y-limits		From: 0.8377 To: 1.576				i 🖬 🎯   🗞   🔍 🤍 🦥 🐙 🔏 •   🔂   🔲 🛄 🔲 🛄
X-limits	New	From: 0 To: 7	+	Close		7 6 Test run no: 1; uSoulsby; Linear 7 6 5 6 7 7 8 9 9 1 1 1 1 1 1 1 1 1 1 1 1 1
						0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 Velocity (m/s)



ModelUI 2D diffusion model



ModelUI Managed Realignment Breach Model

# Demonstration models to illustrate a range of applications

ModelUI

Vertical Tidal Profile

ModelUI SimpleTide

8: Nov 23

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
- Download *muitoolbox.mltbx file* from: <u>https://github.com/CoastalSEA/dstoolbox/releases/</u>
- Online Help is available once toolbox is installed
- Manuals with worked examples available for all demonstration models and Apps







#### **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<sup>™</sup> 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<sup>TM</sup>.

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)



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:

TideUI	× +		_	
⊕ New - 👗 🖸	(°) (4) (2) (°)	$\uparrow$ Sort $\checkmark$ $\equiv$ View $\checkmark$		
$\leftarrow$ $\rightarrow$ $\checkmark$ $\uparrow$ $\blacksquare$ $`````````````````````````````````$	NodelUI > Training > TideUI	✓ C Search TideUI		م
A Home	Name	Date modified	Туре	Size
> 📥 Ian - Personal	📒 doc	09/11/2023 12:31	File folder	
	📒 example	09/11/2023 18:26	File folder	
> 🦲 lan - University of Sou	📒 help	09/11/2023 15:30	File folder	
	Working Docs	09/11/2023 12:43	File folder	
🖿 China 🛛 🖈	info	09/11/2023 15:41	XML File	2
📒 Home4Home 🔹 🖈	🖻 mui_logo	09/11/2023 17:03	JPG File	45
📜 MUImodels 🔹 🖈	慉 simple_tide	02/11/2023 09:13	MATLAB Code	3
🚞 Southampton 🔹 🖈	1 TideModel	09/11/2023 17:46	MATLAB Code	5
🗖 Work 🖈	慉 TideParams	09/11/2023 16:55	MATLAB Code	4
🖛 Data (D:) 🛛 🖈	🔝 TideUI	09/11/2023 17:55	MATLAB Code	13
12 items				



# 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.DataUltabs.Plot = {'2D'<mark>,'3D','4D','2DT','3DT','4DT'};</mark>

%Statistics options: 'General', 'Timeseries', 'Taylor', 'Intervals' '

obj.DataUItabs.Stats = {'General'<mark>,'Timeseries','Taylor','Intervals</mark>};

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 (highlighted in blue)
  - 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

Mode	IUI Versio	on: 3.0; Cop	oyright: Ma	y 2021		_	×
File Too	s Project	Setup F	Run Analy	sis Help	Menus		ъ
				<b>T</b> 1			
Cases	Inputs	Q-Plot	Stats	Tat	DS		

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

```
Top level menu name
                                                                                              'Run parameters', not used
                                                                           Sub-menu list and can be deleted
              %% Setup menu ------
                                                                                              + change 3 to 2, delete off.
Level 1
              menu.Setup(1).List = {'Import Data','Input Parameters', 'Run Parameters', 'Model Constants'};
              menu.Setup(1).Callback = [{'gcbo;'},repmat({@obj.setupMenuOptions},[1,3])];
                                                                                      Cell array of callback functions
              %add separators to menu list (optional - default is off)
                                                                                      • {'gcbo'} passes control to next level
              menu.Setup(1).Separator = {'off','off','off','on'}; %separator precedes item
                                                                                       @functions pass control to other
              % submenu for Import Data (if these are changed need to edit
                                                                                        functions in TideUI
              % loadMenuOptions to be match)
Level 2
              menu.Setup(2).List = {'Load','Add','Delete','Quality Control'};
                                                                         This adds menus to load data from a file
              menu.Setup(2).Callback = repmat({@obj.loadMenuOptions},[1,4]);
                   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 = [];
```

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



end

# 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]
% c Edit input properties classnames
props = {... % << Add additional inputs and adjust layout
'Paraminput\_template','Inputs',[0.45,0.48],{180,60},'Input parameters:';...
'Paraminput\_template','Inputs',[0.45,0.48],{180,60},'Run parameters:';:...</pre>

end

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



# setTabActions

#### Defines bespoke behaviour of tabs (if required)

end

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



# setMenuOptions

#### Edit calls for setup menu



- Replace *ParamInput\_template* with *TideParams*
- Delete Run Parameters case (highlighted)



# 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(~,~,~) % << Edit to documentation name if available doc tideui end

• Note that model name is in lower case



# 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





#### 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)',... 'Velocity phase (deg)',... 'N2 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

You can Copy and Paste this text from the file: mui\_training\_input\_text.txt

properties

Duration Tinterval	%duration (days) %time interrval for simulation (mins)
zO	%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)

You can Copy and Paste this text from the file: mui\_training\_input\_text.txt



end 29: Nov 23

# Test the Data Input

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 SAVE YOUR WORK (File>Save as) But it still does not do anything!

承 D  $\times$ Duration (d) 15 Time interval (mins) 15 Mean tide level (mOD) 02 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 OK Cancel

# 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	Velocity (m/s)	m/s
uH	Horizontal velocity	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

# You can Copy and Paste this text from the file: mui\_training\_input\_text.txt

	0/
	% Model code < <insert call="" code="" model="" or="">&gt;</insert>
	<pre>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'; %</pre>
	% % 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
33	: Nov 23

# 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



# 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 substructure 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.

# You can Copy and Paste this text from the file: mui\_training\_input\_text.txt

function dsp = modelDSproperties(~)

'Description',{"},...

'Unit',{"},... 'Label',{''},... 'Format',{''});

%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',{''},...



# 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 Q-Plot tab Use Analysis>Plots menu to produce a range of plots



# View model output on Q-Plot tab





# 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



# 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);
        mulfunctions
end
```

## setDSproperties

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

T	Time h 1	uV uH	
2	'25-Feb-2021	18:48:25'	-1.837334321 0.000540367 1.171610719
3	'25-Feb-2021	19:48:28'	0.461551034 0.00062233 1.303652003
4	'25-Feb-2021	20:48:31'	1.901327587 0.000107269 0.170780697
5	'25-Feb-2021	21:48:34'	1.113827552 -0.000504548 -1.106143433
6	'25-Feb-2021	22:48:37'	-1.120475837 -0.000625049 -1.27996699
7	'25-Feb-2021	23:48:40'	-2.63027956 -0.000137237 -0.166824602
8	'26-Feb-2021	00:48:42'	-1.927763534 0.000496039 1.180937215
9	'26-Feb-2021	01:48:45'	0.374372426 0.000673248 1.476039141
10	'26-Feb-2021	02:48:48'	2.141164923 0.000224011 0.416908491
11	'26-Feb-2021	03:48:51'	1.730517596 -0.000431979 -1.027924702
12	'26-Feb-2021	04:48:54'	-0.466715683 -0.000683858 -1.534115929
13	'26-Feb-2021	05:48:57'	-2.409765832 -0.000305181 -0.663394162
14	'26-Feb-2021	06:49:00'	-2.335701996 0.000338036 0.739100401
15	'26-Feb-2021	07:49:03'	-0.401768487 0.000638966 1.356647671
16	'26-Feb-2021	08:49:05'	1.498951651 0.000323759 0.643711489
17	'26-Feb-2021	09:49:08'	1.55153561 -0.000295575 -0.675337282
18	'26-Feb-2021	10:49:11'	-0.266670605 -0.000623502 -1.311511055
19	'26-Feb-2021	11:49:14'	-2.186209214 -0.000346935 -0.640594555
20	'26-Feb-2021	12:49:17'	-2.339672147 0.000271572 0.708942287
21	'26-Feb-2021	13:49:20'	-0.533789106 0.00064642 1.457050418
22	'26-Feb-2021	14:49:23'	1.571530745 0.000423311 0.88367852
23	26-Feb-2021	15:49:26	2.029950558 -0.000189582 -0.495658657
24	26-Feb-2021	16:49:28	0.430964843 -0.000622101 -1.419574086
25	26-Feb-2021	17:49:31	-1.734399208 $-0.000479395$ $-1.063169577$
26	26-Feb-2021	18:49:34	-2.48810531 9.22378E-05 0.203042362
27	'26-Feb-2021	19:49:37	-1.210929512 0.000551152 1.181384669
28	26-Feb-2021	20:49:40	0.813034747 0.000473432 0.980188919
29	26-Feb-2021	21:49:43	1.640434003 -5.29146E-05 -0.16067596
30	26-Feb-2021	22:49:46	0.506451483 -0.000519703 -1.119158213
22	20-Feb-2021	23:49:49	-1.4/4859/04 -0.000482495 -0.965/10155
2Z 22	27-Feb-2021	00:49:52	1 222250006 0 000E1067 1 211004771
21	27-Feb-2021	02.49.54	-1.525255656 0.00051667 1.211604771
35	27-Feb-2021	02:49:57	1 945620644 6 520238-05 0 076709123
36	27-Feb-2021	04.50.03	1 159520044 0.52025E-05 0.070705125
37	27 Feb 2021	05.50.06	
38	'27-Feb-2021	06:50:09'	-2.261345459 -0.000153082 -0.341431726
39	'27-Feb-2021	07:50:12	-1.816347742 0.000374009 0.806515498
40	'27-Feb-2021	08:50:15'	-0.031833089 0.000528517 1.116103852

#### function dsp = setDSproperties()

'Format',{"'});

%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',{"},...



# View imported data on Q-Plot tab





# 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>



# Converting the model to an App

Matlab provide the tools needed 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

The following slides explain these steps in more detail



# 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
    - o 'doc', 'help' and 'example' folders
    - $\circ$  info.xml
    - $\circ$  mui\_logo.jpg
    - $\circ$  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		– 🗆 X
D:\Matlab Code\MUImodels\muiApps\ModelUI\	Training\TideUI\TideUI.prj	۲
Pick main file	Describe your app	Package into installation file
Main file Main file Control Main file Main file Memove main file Files included through analysis Simple_tide.m TideModel.m TideParams.m Rerun analysis	IdeUI       1.0         Ian Townend       1.0         i.townend@soton.ac.uk       Inversity of Southampton         University of Southampton       Set as default contact         App to demsonstrate the use of ModelUI to construct a bespoke user interface       Image: Construct a bespoke user interface         B       I       Image: Construct a bespoke user interface	Output folder D:\Matlab Code\MUImodels\muiAp Browse Package Packaging complete.
Shared resources and helper files  doc  example	App to demsonstrate the use of ModelUI to construct a bespoke user interface	Open output folder. Package
<ul> <li>help</li> <li>info.xml</li> <li>mui_logo.jpg</li> <li>tideui_import.m</li> <li>Add files/folders</li> </ul>	Products	

# 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 the 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.

45: Nov 23

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.

\*To publish an .m file, go the Publish Tab in Matlab and use the Publish menu option. This creates an html file in the html sub-folder



# Help (2) – files in *htm*/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'):





# 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 are explained at <u>www.coastalsea.uk</u> and can be downloaded at <u>https://github.com/CoastalSEA</u> When dstoolbox and muitoolbox are installed, further documentation can be accessed using the Matlab<sup>TM</sup> Supplemental Software documentation.

Bestimonitation			
CONTENTS Concentration Home  dataCollock (Supplemental Software) datoolbox Classes Function Reference datoolbox Examples Developers web site Examples	Actionability           Instantiant           Actionability           Actionability </td <td>CONTENTS  Contents  Conte</td> <td>nuitoolbox mutoolbox at collection of classes used to create bespoke Uts for models and data analysis. <b>Description Performant Perf</b></td>	CONTENTS  Conte	nuitoolbox mutoolbox at collection of classes used to create bespoke Uts for models and data analysis. <b>Description Performant Perf</b>
	ostable, noice a collection of one or more tratasets 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     deta) which are loaded into drahear and uncleanded inter destable.		<ul> <li>Interruption = on sociation cases are provided in the interruption of colouring and displaying input parameters, minimising the effort to evelup interactive data input.</li> <li>mulDataUI - an abstract class for creating graphic user interface to select data and pass se applications.</li> </ul>

# The dstoolbox

The *dstoolbox* is designed to store and manage multidimensional 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).



### 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



