DeSIO_GUI Documentation

Guided by

David maertins
Wissenschaftliche Mitarbeiterinnen und Mitarbeiter
EMAIL - d.maertinsisd.uni-hannover.de
501, Appelstraße 9a
30167 Hannover

Daniel schulter
Wissenschaftliche Mitarbeiterinnen und Mitarbeiter
EMAIL - d.schuster@isd.uni-hannover.de
501, Appelstraße 9a
30167 Hannover

Developed by
Kishan paladiya
Jamal bhatti

Contents

1. **Beam** 2
Class Beam (): 2
Class stand (): 4
Class Node (): 4
Class Element (): 5
Class Segment (): 5
2. **Desio** 6
3. **Gui_fun_file** 6
Class MonopilePage: 6
Class TowerPage: 7
4. **Io** 9
5. **Structure** 9
6. **Utils** 10
Class Compartment (): 10
Class CrossSectionProperty (): 10
Class Geometry (): 11
Class Utilities (): 11

For this development, the required libraries are listed below.

- os, sys, pathlib, numpy, PyQt5.QtWidgets, PyQt5, traceback, math, matplotlib.pyplot, time, subprocess

1. Beam

There was total 5 classes in the side beam folder, each with its own functionality, which is listed below.

```
class Beam ():
class Element ():
class Node ():
class Segment ():
class Stand ():
class Utilities ():
```

Class Beam ():

There are numerous functions in this class that actually function to iterate the values provided by the user and store them in various lists. This class takes values such as beam_id, beam_type, n_segments, and n_elements from the instance.

There are four main classes from which value is extracted.

```
from beam.segment import Segment
from beam.node import Node
from beam.element import Element
from Utils.geometry import Geometry
```

- Different functions inside beam class shown below

```
def getBeamAsStr(self):
```

Three different variables that store value in the form of list

- segmentsStr = "[]"
- nodesStr = "[]"
- elementsStr = "[]"

generate 3 different variable that content string values in the List and return text value using input data which is shown below.

```
Beam ID: 1, Beam Type: Stand, Beam Class: S, Beam Length: 1.000, No. of
segments: 2, No. of elements: 3, Segments: ………, Nodes: ………, Elements: ……
```

- This function uses static method to iterate data

```
@staticmethod
def getBeamsAsStr(beams):
```

This function increase base on the beam value. For instance, if the beam size is more like 3, and 4 beams. Base on that value beam string data increase and store.

```
return beamsStr
```

- This function takes a single argument in the form of dictionary.

```
def setSegmentbyDic(self,dics):
```

Return the segment value as a tuple after extracting the elements from the dictionary.

```
self.segments = segments
```

- This function takes the segment value from beam class. After that checks the segment value are true or not and return Boolean output.

```
def setSegments(self, segments):
```

- This function uses instance value of start point and end_point to determine the length of a beam. If the value is empty, return false; otherwise, use the Geometry class inside findPointsDistance function to calculate the distance between start_point and end_point.

```
def getBeamLength(self):
```

```
return self.beam_length
```

- This function takes value from instance and return class of beam

```
def getBeamClass(self):
```

```
return self.beam_class
```

- Take variable of the starting and end points of beams return start_point and end_point.

```
def setStartAndEndPoints(self, start_point, end_point):
    self.start_point = start_point
    self.end_point = end_point
```

- This function takes nodes as n input variable and return nodes.

```
def setNodes(self, nodes):
    self.nodes = nodes
```

- This function takes a single argument which is elements and return as an element.

```
def setElements(self, elements):
    self.elements = elements
```

- This function takes a single beam class argument, and return beam_class

```
def setBeamClass(self, beam_class):
    self.beam_class = beam_class
```

- This method returns a value based on the value of a segment. Define the start point and end point variables. Based on the change, construct dir_vector as 3*1 matrices and store in segment, then continue till you reach the end point.

```
def generateSegmentsStartAndEndPoints(self):
```

- This function takes a single argument of n_comps and return beam classes and their respective descriptions.

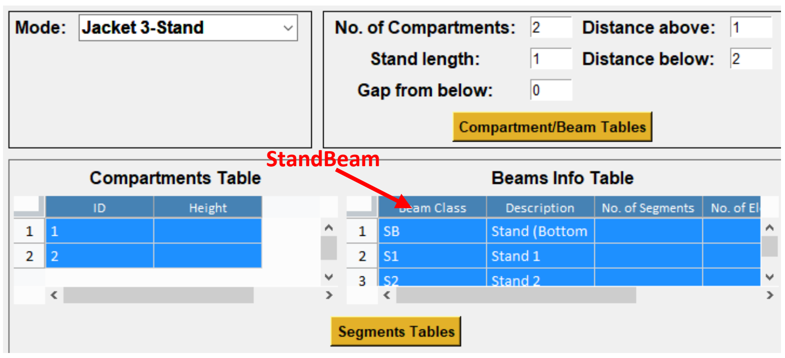
```
@staticmethod
def getBeamClasses(n_comps):
```

- Because they have a large number of stands and a beam class based on n_compartment, these two functions are used for Jacket 3 and Jacket 4 beams. The values of stand beam classes and comp beam classes are returned.

```
@staticmethod
def getStandBeamClasses(n_comps):
```

```
@staticmethod
def getCompBeamClasses(n_comps):
```

Which is look like this in the figure shows below.



Mode: Jacket 3-Stand

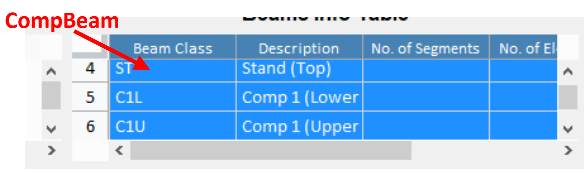
No. of Compartments: 2
Stand length: 1
Gap from below: 0
Distance above: 1
Distance below: 2

Compartments Table

	ID	Height
1	1	
2	2	

Beams Info Table

	Beam Class	Description	No. of Segments	No. of El
1	SB	Stand (Bottom		
2	S1	Stand 1		
3	S2	Stand 2		



	Beam Class	Description	No. of Segments	No. of El
4	ST	Stand (Top)		
5	C1L	Comp 1 (Lower		
6	C1U	Comp 1 (Upper		

Class stand ():

This class takes 2 arguments like stand_id and height. Return the value which is shown below.

```
STAND:
[Stand ID: 1, Height: 1.000, Beams: []]
```

Class Node ():

This class takes 3 different arguments and return those three arguments base on how many nodes inside beam. This list increases base on the length of nodes.

```
Nodes: [
[Local Num: 1, Global Num: 1, Coordinates: [……….]],
```

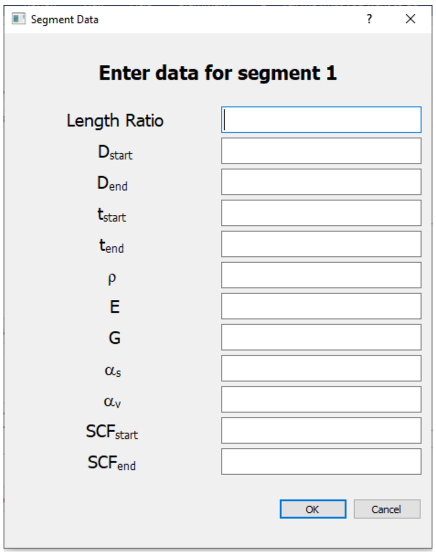
Class Element ():

This describes inside code including comments. Return this kind of element list and also increase base on elements which is shown below.

```
Elements: [
[Local Num: 1, Global Num: 1,
Start Node: [Local Num: 1, Global Num: 1, Coordinates: [………]],
End Node: [Local Num: 2, Global Num: 2, Coordinates: […………]],
Cross Section Property: […………]]
```

Class Segment ():

Takes all variable from the input value of the segment table. Which is shown below.



Length Ratio	
D_{start}	
D_{end}	
t_{start}	
t_{end}	
ρ	
E	
G	
α_s	
α_v	
SCF_{start}	
SCF_{end}	

class function user can see inside code and gives the output like this shown below.

```
Segments: [
[Segment ID: 0 (Start Point: ['0.000000', '0.000000', '-1.000000'], End
Point: ['0.000000', '0.000000', '-1.000000']),
Length Ratio: 0.000, Diameter Start: 0.000, Diameter End: 1.000, Thickness
Start: 0.000, Thickness End: 1.000, Density: 1.000, E: 1.000, G: 1.000,
alpha_s: 1.000, alpha_v: 1.000, SCF Start: 1.000, SCF End: 3.000]
```

2. Desio

The monopile, tower, jacket-3, and jacket-4 beam types are all displayed in this folder. Also included is a video illustration of how all of the GUI looked suspiciously.

3. Gui_fun_file

Monopile data is generated using this class. Use the various classes listed below to do this.

Class MonopilePage:

```
from Utils.utilities import Utilities as util
from structure.monopile.monopile import Monopile
from structure.jacket.jacket import Jacket
from beam.beam import Beam
from beam.segment import Segment
from beam.stand import Stand
from Utils.segments_userInterface import segments_ui
from Utils.geometry import Geometry
import traceback
from PyQt5.QtWidgets import QDialog
from segment_table import *
```

- This grabs all the variables from the combo table in the form of a set, such as stand_legth, no_segments, and no_elements. Also use for J3 and J4 to grab there text values which user can see inside code.

```
def getValuesFromParent(self):
```

```
self.mono_page_fields = {}
self.j3_page_fields = {}
self.j4_page_field = {}
```

- These three functions are used to examine values that come from the combo table, such as grater then 0 or equal to 0 based on requirements. Also convert text values in float or int.

```
def checkMonoPageInputs(self):
```

```
def checkJ4Pageinpurs(self):
```

```
def checkJ3Pageinputs(self):
```

- This function reads a text value from the segment dialog table and checks whether the data is acceptable or not. This use for a single input segment. If there are more input segments then use different function. Which is shown below.

```
def checkDialogInput(self, input_fields):
```

```
def checkAllSegments(self):
```

- Sets the Beams and segment using different class called stand.py and beam.py

```
def generate_input_files(self):
        self.monopile_stand.setBeams(self.beams)
        self.monopile_stand.beams[0].setSegments(self.segments)
```

create monopile object using Monopile class, which need single argument.

```
self.monopile = Monopile(self.monopile_stand)
```

Check lengthRatiosValidation of monopile data if not valid gives the error message. Generate Beam Input Data using functions like writeBeamInput(), writeLogFile(), (Beam, Segments, Nodes, Elements, etc.) of Monopile.

```
beamInputGenerated = self.monopile.generateBeamInputData()
if beamInputGenerated:
    self.monopile.writeBeamInput()
    self.monopile.writeLogFile()
```

- Get the data from Beamclass, standBeamClass and CompBeamClass. Also initialize Stand and beam which can clearly see in code.

```
def load_monopile(self):
```

Append monopile beam to the list of beams (single item list). taking values from segment table.

```
self.beams = list()
self.beams.append(self.monopile_beam)

self.segments = []
self.segment_btns = segments_ui(self.ui.tabSupportStructure,
self.mono_page_fields["no_segments"])
```

Class TowerPage:

All of these functions work in the same way as the MonopilePage class function.

```
def getValuesFromParent(self):
def checkTowerPageInputs(self):
def checkDialogInput(self,input_fields):
def checkAllSegments(self):
def dispDialogSegmentTable(self,id=None):
```

- This function also works like MonopilePage class. Set intermediate parameters of bema and segment.

```
def writeBeamFile(self):
        self.tower.setBeams(self.beams)
        self.tower.beams[0].setSegments(self.segments)
```

check the raise exception if the length ratios not valid. Generate Beam Input Data (Beam, Segments, Nodes, Elements, etc.) of Tower using Tower class. Write beam input perameter using function writeBeamInput(), and write Log file using writeLogFile() function.

```
beamInputGenerated = self.tower.generateBeamInputData()
if beamInputGenerated:
    self.tower.writeBeamInput()
    self.tower.writeLogFile()
```

Append tower beam to the list of beams (single item list), taking values from segment table.

```
self.beams = list()
self.beams.append(self.tower_beam)

self.segments = []
self.segment_btns = segments_ui(self.ui.tabTower,
self.tower_page_fields["no_segments"])
```

4. Io

The contents of this folder store structure log data as well as output of beam input data or tower input data, which user can see in the io folder for clear understanding.

5. Structure

This has classes such as Monopile, Jacket, and Tower that are used to generate beam input data, generate cross section properties, obtain cross section properties, write Beam input, and write log file.

Describe each step in the code using comments so that the user may refer back to it.

```
class Jacket():
class Monopile():
class Tower():
```

6. Utils

There are various classes in this folder, as indicated below.

```
class Compartment():
class CrossSectionProperty():
class Geometry():
class Utilities():
```

These two classes are used to produce 2 different tables. A beaminfo_ui class used to utilized to create Beam info for J3 and J4 beams, while another class segments are used to generate a segment info table for all beams for example Monopile, J3, J4 and Tower as well.

```
class beamsinfo_ui(QtWidgets.QWidget):
class segments_ui(QtWidgets.QWidget):
```

Class Compartment ():

This particular class used for J3 and J4 Beam to set and get the compartment string values. For that they need 2 different arguments compartment_id , height.

- This function grabs the value of compartment from Beam class.

```
getCompartmentAsStr(self):
```

```
return f"[Compartment ID: {int(float(self.compartment_id))}, Height:
{float(self.height):.3f},\nBeams: {beamsStr}\n]"
```

Result:

```
COMPARTMENTS:

[[Compartment ID: 1, Height: 2.000,Beams: []]]
```

Class CrossSectionProperty ():

This class contains a variety of functions, which are listed below.

```
def setProperties(self, properties):
def getPropertyAsStr(self):
def getPropertiesAsLine(self):
def getCrossSectionProperties(prop_id, beam_length, n_elems, segments):
def deriveCrossSectionProperty(elem_prop):
```

- To derive cross section properties, use this class to declare all elements first. For example curr_elem_property = [elem_length, elem_diameter, elem_diameter, elem_thickness, elem_thickness, current_segment.density, current_segment.e, current_segment.g, current_segment.alpha_s, current_segment.alpha_v, current_segment.scf_start, current_segment.scf_end]
- This function was used for this, and it is extensively described in the code using comments.
- Return cross section properties

```
@staticmethod
def getCrossSectionProperties(prop_id, beam_length, n_elems, segments):
```

```
return cross_section_properties
```

- All mathematic formulas in the form of matrices that determine cross-section characteristics are stored in this function. Inside the code, fully describe using comments.

```
def deriveCrossSectionProperty(elem_prop):
```

Class Geometry ():

Mathematical formulas for geometry basis functions such as magnitude, normalize vector, Orthonormal bases, Radius, Diameter, Distance, Direction vector, three-dimensional intersection, and so on are also included in this class. All are used staticmethod decorator.

```
def getMagnitude(vector):
```

- The normalized vector of 1×3 metric may be created by using magnitude values.

```
def normalizeVector(vector):
```

```
def findOrthonormalBases(dir_vector):
def findRadius(n_points, dist_bw_points):
def findStandRadiusAtPoint(stand, point):
def findDiameterAtSegmentPosition(diameter_start, diameter_end, pos_ratio):
def findPointsDistance(start_point, end_point):
def findPointOnVectorFromRatio(start_point, dir_vector, dist_ratio):
def findPointOnVector(start_point, dir_vector, dist):
def findDirectionVector(start_point, end_point):
def findThreeDimIntersection(prev_lower, prev_upper, next_lower,
next_upper):
def findAdjacentCirclePoints(n_points, center, radius):
def getCoordinates(start_point, end_point, n_elements):
```

Class Utilities ():

All functions use staticmethod decorator

- Checks if a string is an integer or not
- Checks if a string is a float or not
- Checks if a string is a positive number or not
- Checks if a string is a positive number and greater than 0
- Checks if a string is a positive int and greater than 0
- Checks if a string is a positive float and greater than 0
- Checks if a string is a positive float and greater or equal to 0
- Convert empty values to zero
- Convert dictionary to float
- To show Information message box
- To show warning message box
- To show error message box