

```

1 # -*- coding: utf-8 -*-
2
3 """
4 interface.py: A Tkinter application for creating configuration files to
5 run simulation with csenergy.py
6 @author: pacomunuera
7 2020
8 """
9
10
11 import sys
12 sys.path.append('./libs')
13 import os.path
14 import csenergy as cs
15 import CoolProp.CoolProp as CP
16 import pvlib as pvlib
17 import tkinter as tk
18 from tkinter.filedialog import askopenfilename, asksaveasfile
19 import tkinter.ttk as ttk
20 from tkinter import messagebox
21 # recipe-580746-1.py from
22 # http://code.activestate.com/recipes/
23 # 580746-t kinter-treeview-like-a-table-or-multicolumn-listbox/
24 import recipe5807461 as table
25 import json
26 from decimal import Decimal
27 from datetime import datetime
28 import pandas as pd
29
30
31 class Interface(object):
32
33     _COOLPROP_FLUIDS = ['Water', 'INCOMP::TVP1', 'INCOMP::S800']
34
35     _MODELS = ['Barbero4thOrder', 'Barbero1stOrder', 'BarberoSimplified']
36
37     _DIR = {
38         'saved_configurations': './saved_configurations/',
39         'site_files': './site_files/',
40         'fluid_files': './fluid_files',
41         'hce_files': './hce_files',
42         'sca_files': './sca_files',
43         'fielddata_files': './fielddata_files',
44         'weather_files': './weather_files'}
45
46     cfg_settings = {
47         'simulation': {},
48         'solar_plant': {},
49         'site': {},
50         'weather': {},
51         'hce': {},
52         'SCA': {},
53         'hot_hce': {},
54         'cold_hce': {},
55         'cycle': {},
56         'hce_model_settings': {},
57         'hce_scattered_params': {},
58         'sca_scattered_params': {}}
59
60     def __init__(self):

```

```

61
62     # Main window
63     self.root = tk.Tk()
64     self.root.attributes('-fullscreen', False)
65     self.varroottitle = tk.StringVar(self.root)
66     self.root.title('Solar Field Configurator ')
67     w, h = self.root.winfo_screenwidth(), self.root.winfo_screenheight()
68     self.root.geometry('%dx%d+0+0' % (w-200, h-200))
69     self.root.geometry('%dx%d+0+0' % (800, 800))
70
71     # Menu
72     self.menubar = tk.Menu(self.root)
73
74     self.simulation_menu = tk.Menu(self.menubar, tearoff=0)
75     self.simulation_menu.add_command(
76         label='New', command=self.simulation_new)
77     self.simulation_menu.add_command(
78         label='Open', command=self.simulation_open)
79     self.simulation_menu.add_command(
80         label='Save', command=self.simulation_save)
81     self.simulation_menu.add_command(
82         label='Save as...', command=self.simulation_save_as)
83     self.simulation_menu.add_separator()
84     self.simulation_menu.add_command(
85         label='Exit', command=self.simulation_exit)
86     self.menubar.add_cascade(
87         label='Simulation', menu=self.simulation_menu)
88     self.run_menu = tk.Menu(self.menubar, tearoff=0)
89     self.run_menu.add_command(label='Run', command=self.run_simulation)
90     self.menubar.add_cascade(label='Run', menu=self.run_menu)
91     self.help_menu = tk.Menu(self.menubar, tearoff=0)
92     self.help_menu.add_command(label='Help', command=self.help_help)
93     self.help_menu.add_command(label='About', command=self.help_about)
94     self.menubar.add_cascade(label='Help', menu=self.help_menu)
95     self.root.config(menu=self.menubar)
96
97     # Notebook (tabs)
98     self.nb = ttk.Notebook(self.root)
99
100    self.fr_simulation = tk.Frame(self.nb)
101    self.fr_solarfield = tk.Frame(self.nb)
102    self.fr_fluid = tk.Frame(self.nb)
103    self.fr_hce = tk.Frame(self.nb)
104    self.fr_sca = tk.Frame(self.nb)
105
106    self.buildNotebook()
107    self.buildSolarFieldFrame()
108    self.buildSimulationFrame()
109    self.buildFluidFrame()
110    self.buildHCEFrame()
111    self.buildSCAFrame()
112
113    self.simulation_open(self._DIR['saved_configurations']+ 'template.json')
114
115    def simulation_new(self):
116
117        # Simulation configuration
118        self.varsimID.set('')
119        self.varsimdatatype.set(1)
120        self.varsimulation.set(False)

```

```

121     self.varbenchmark.set(False)
122     self.varfastmode.set(False)
123     self.vardatafilename.set('')
124     self.vardatafilepath.set('')
125     self.vardatafileurl.set('')
126     self.varfirstdate.set(pd.to_datetime('2000/01/01 1:00').strftime(
127         '%Y/%m/%d %H:%M'))
128     self.varlastdate.set(pd.to_datetime('2000/12/31 1:00').strftime(
129         '%Y/%m/%d %H:%M'))
130     self.cmbmodelname.set('')
131     self.varmodelmaxerrt.set(1.0)
132     self.varmodelmaxerrtro.set(0.1)
133     self.varmodelmaxerrpr.set(0.01)
134
135     self.varsitename.set(0)
136     self.varsitelat.set(0)
137     self.varsitelong.set(0)
138     self.varsitealt.set(0)
139     self.checkoptions()
140     self.checkfastmode()
141
142     # Solarfield configuration
143     self.solarfield_table.table_data = []
144     self.columns_table.table_data = []
145
146     self.vartin.set(0)
147     self.vartout.set(0)
148     self.varpin.set(0)
149     self.varpout.set(0)
150     self.vartmin.set(0)
151     self.vartmax.set(0)
152     self.varratedmassflow.set(0)
153     self.varrecirculation.set(0)
154     self.varscas.set(0)
155     self.varhces.set(0)
156     self.varrowspacing.set(0)
157     self.varscatrackingtype.set(1)
158     self.varfluidtable.set(1)
159     self.varfluidname.set('')
160     self.varfluidtmax.set(0)
161     self.varfluidtmin.set(0)
162     self.fluid_table.table_data = []
163     self.cmbcoolpropID.set('')
164     self.checkfluid()
165
166     # SCA configuration
167     self.varscaname.set('')
168     self.varscalength.set(0)
169     self.varscaperture.set(0)
170     self.varscafocallen.set(0)
171     self.varscIAMF0.set(0)
172     self.varscIAMF1.set(0)
173     self.varscIAMF2.set(0)
174     self.varscareflectance.set(0)
175     self.varscageoaccuracy.set(0)
176     self.varscatracktwist.set(0)
177     self.varscacleanliness.set(0)
178     self.varscafactor.set(0)
179     self.varscavailability.set(0)
180

```

```

181     # HCE configuration
182     self.varhcename.set('')
183     self.varhcedri.set(0)
184     self.varhcedro.set(0)
185     self.varhcedgi.set(0)
186     self.varhcedgo.set(0)
187     self.varhcelength.set(0)
188     self.varhceemittanceA0.set(0)
189     self.varhceemittanceA1.set(0)
190     self.varhceabsorptance.set(0)
191     self.varhcetransmittance.set(0)
192     self.varhceinnerroughness.set(0)
193     self.varhceminreynolds.set(0)
194     self.varhcebrackets.set(0)
195     self.updateHCEperSCA()
196
197     self.fr_fluid.update()
198     self.fr_hce.update()
199     self.fr_solarfield.update()
200     self.fr_sca.update()
201     self.fr_simulation.update()
202
203 def simulation_open(self, path=None):
204
205     if path == None:
206         path = askopenfilename(initialdir=self._DIR['saved_configurations'],
207                               title='choose your file',
208                               filetypes=[('JSON files', '*.json')])
209
210     head, tail = os.path.split(path)
211     self.filename = path
212     self.root.title('Solar Field Configurator: ' + tail)
213     with open(path) as cfg_file:
214         cfg = json.load(cfg_file,
215                         parse_float= float,
216                         parse_int= int)
217
218     # Simulation configuration
219     self.varsimID.set(cfg['simulation']['ID'])
220     self.varsimdatatype.set(cfg['simulation']['datatype'])
221     self.varsimulation.set(cfg['simulation']['simulation'])
222     self.varbenchmark.set(cfg['simulation']['benchmark'])
223     self.varfastmode.set(cfg['simulation']['fastmode'])
224     self.vardatafilename.set(cfg['simulation']['filename'])
225     self.vardatafilepath.set(cfg['simulation']['filepath'])
226     self.vardatafileurl.set(cfg['simulation']['filepath'] +
227                             cfg['simulation']['filename'])
228
229     self.varfirstdate.set(pd.to_datetime(
230         cfg['simulation']['first_date']))
231     self.varlastdate.set(pd.to_datetime(
232         cfg['simulation']['last_date']))
233
234     self.cmbmodelname.set(cfg['model']['name'])
235     self.varmodelmaxerrt.set(cfg['model']['max_err_t'])
236     self.varmodelmaxerrtro.set(cfg['model']['max_err_tro'])
237     self.varmodelmaxerrpr.set(cfg['model']['max_err_pr'])
238
239     self.varsitename.set(cfg['site']['name'])
240     self.varsitelat.set(cfg['site']['latitude'])

```

```

241     self.varsitelong.set(cfg['site']['longitude'])
242     self.varsitealt.set(cfg['site']['altitude'])
243     self.checkoptions()
244     self.checkfastmode()
245
246     # Solarfield configuration
247     list_subfields = []
248     for r in cfg['subfields']:
249         list_subfields.append(list(r.values()))
250
251     self.solarfield_table.table_data = list_subfields
252
253     if 'tags' in cfg.keys():
254         list_tags = []
255         for r in cfg['tags']:
256             list_tags.append([r, ' ', cfg['tags'][r]])
257         self.columns_table.table_data = list_tags
258
259         self.vartin.set(cfg['loop']['rated_tin'])
260         self.vartout.set(cfg['loop']['rated_tout'])
261         self.varpin.set(cfg['loop']['rated_pin'])
262         self.varpout.set(cfg['loop']['rated_pout'])
263         self.vartmin.set(cfg['loop']['tmin'])
264         self.vartmax.set(cfg['loop']['tmax'])
265         self.varratedmassflow.set(cfg['loop']['rated_massflow'])
266         self.varrecirculation.set(cfg['loop']['min_massflow'])
267         self.varscas.set(cfg['loop']['scas'])
268         self.varhces.set(cfg['loop']['hces'])
269         self.varrowspacing.set(cfg['loop']['row_spacing'])
270         self.varscatrackingtype.set(cfg['loop']['Tracking Type'])
271
272     # HTF configuration
273     fluid_table = []
274
275     if cfg['HTF']['source'] == 'table':
276         self.varfluidtable.set(1)
277         self.varfluidname.set(cfg['HTF']['name'])
278         self.varfluidtmax.set(cfg['HTF']['tmax'])
279         self.varfluidtmin.set(cfg['HTF']['tmin'])
280         fluid_table.append(['mu'] + cfg['HTF']['mu'])
281         fluid_table.append(['cp'] + cfg['HTF']['cp'])
282         fluid_table.append(['rho'] + cfg['HTF']['rho'])
283         fluid_table.append(['kt'] + cfg['HTF']['kt'])
284         fluid_table.append(['h'] + cfg['HTF']['h'])
285         fluid_table.append(['t'] + cfg['HTF']['t'])
286         self.fluid_table.table_data = fluid_table
287
288     elif cfg['HTF']['source'] == 'CoolProp':
289         self.varfluidtable.set(2)
290         self.cmbcoolpropID.set(cfg['HTF']['CoolPropID'])
291         self.varfluidtmax.set(cfg['HTF']['tmax'])
292         self.varfluidtmin.set(cfg['HTF']['tmin'])
293
294     self.checkfluid()
295
296     # SCA configuration
297     self.varscaname.set(cfg['SCA']['Name'])
298     self.varscalength.set(cfg['SCA']['SCA Length'])
299     self.varscaperture.set(cfg['SCA']['Aperture'])
300     self.varscafocallen.set(cfg['SCA']['Focal Len'])

```

```

301     self.varscaIAMF0.set(cfg['SCA']['IAM Coefficient F0'])
302     self.varscaIAMF1.set(cfg['SCA']['IAM Coefficient F1'])
303     self.varscaIAMF2.set(cfg['SCA']['IAM Coefficient F2'])
304     self.varscareflectance.set(cfg['SCA']['Reflectance'])
305     self.varscageoaccuracy.set(cfg['SCA']['Geom.Accuracy'])
306     self.varscatracktwist.set(cfg['SCA']['Track Twist'])
307     self.varscacleanliness.set(cfg['SCA']['Cleanliness'])
308     self.varscafactor.set(cfg['SCA']['Factor'])
309     self.varscavailability.set(cfg['SCA']['Availability'])
310
311     # HCE configuration
312     self.varhcename.set(cfg['HCE']['Name'])
313     self.varhcedri.set(cfg['HCE']['Absorber tube inner diameter'])
314     self.varhcedro.set(cfg['HCE']['Absorber tube outer diameter'])
315     self.varhcedgi.set(cfg['HCE']['Glass envelope inner diameter'])
316     self.varhcedgo.set(cfg['HCE']['Glass envelope outer diameter'])
317     self.varhcelength.set(cfg['HCE']['Length'])
318     self.varbellowsratio.set(cfg['HCE']['Bellows ratio'])
319     self.varshieldshading.set(cfg['HCE']['Shield shading'])
320     self.varhceemittanceA0.set(cfg['HCE']['Absorber emittance factor A0'])
321     self.varhceemittanceA1.set(cfg['HCE']['Absorber emittance factor A1'])
322     self.varhceabsorptance.set(cfg['HCE']['Absorber absorptance'])
323     self.varhcetransmittance.set(cfg['HCE']['Envelope transmittance'])
324     self.varhceinnerroughness.set(cfg['HCE']['Inner surface roughness'])
325     self.varhceminreynolds.set(cfg['HCE']['Min Reynolds'])
326     self.varhcebrackets.set(cfg['HCE']['Brackets'])
327     self.updateHCEperSCA()
328
329     self.fr_fluid.update()
330     self.fr_hce.update()
331     self.fr_solarfield.update()
332     self.fr_sca.update()
333     self.fr_simulation.update()
334
335 def simulation_save(self):
336
337     self.save_as_JSON(self.generate_json(), self.filename)
338
339
340 def simulation_save_as(self):
341
342     self.save_as_JSON(self.generate_json())
343
344 def generate_json(self):
345
346     self.tagslist = []
347
348     cfg = dict({'simulation': {},
349                 'model': {},
350                 'site': {},
351                 'loop': {},
352                 'SCA': {},
353                 'HCE': {},
354                 'HTF': {}})
355
356
357     cfg['simulation'][ID] = self.varsimID.get()
358     cfg['simulation'][datatype] = self.varsimdatatype.get()
359     cfg['simulation'][simulation] = self.varsimulation.get()
360     cfg['simulation'][benchmark] = self.varbenchmark.get()

```

```

361     cfg['simulation']['fastmode'] = self.varfastmode.get()
362     cfg['simulation']['filename'] = self.vardatafilename.get()
363     cfg['simulation']['filepath'] = self.vardatafilepath.get()
364     cfg['simulation']['first_date'] = pd.to_datetime(
365         self.varfirstdate.get()).strftime(
366             '%Y/%m/%d %H:%M%z')
367     cfg['simulation']['last_date'] = pd.to_datetime(
368         self.varlastdate.get()).strftime(
369             '%Y/%m/%d %H:%M%z')
370
371     cfg['model']['name'] = self.cmbmodelname.get()
372     cfg['model']['max_err_t'] = self.varmodelmaxerrt.get()
373     cfg['model']['max_err_tro'] = self.varmodelmaxerrtro.get()
374     cfg['model']['max_err_pr'] = self.varmodelmaxerrpr.get()
375
376     # Site configuration
377     cfg['site']['name'] = self.varsitename.get()
378     cfg['site']['latitude'] = self.varsitelat.get()
379     cfg['site']['longitude'] = self.varsitelong.get()
380     cfg['site']['altitude'] = self.varsitealt.get()
381
382     # HTF configuration
383     if self.varfluidtable.get() == 1:
384         cfg['HTF']['source'] = 'table'
385         cfg['HTF']['name'] = self.varfluidname.get()
386
387         parameters_table = list(self.fluid_table.table_data)
388
389         for r in parameters_table:
390             param_name = r[0]
391             param_values = r[1:]
392             param_values = list(map(self.to_number, r[1:]))
393             cfg['HTF'].update(dict({param_name : param_values}))
394
395         cfg['HTF'].update({'tmax' : float(self.entmax.get())})
396         cfg['HTF'].update({'tmin' : float(self.entmin.get())})
397
398     elif self.varfluidtable.get() == 2:
399         cfg['HTF']['source'] = 'CoolProp'
400         cfg['HTF']['CoolPropID'] = self.cmbcoolpropID.get()
401         cfg['HTF']['tmax'] = float(self.varcoolproptmax.get())
402         cfg['HTF']['tmin'] = float(self.varcoolproptmin.get())
403
404
405     # SCA configuration
406     cfg['SCA']['Name'] = self.varscaname.get()
407     cfg['SCA']['SCA Length'] = self.varscalength.get()
408     cfg['SCA']['Aperture'] = self.varscaperture.get()
409     cfg['SCA']['Focal Len'] = self.varscafocallen.get()
410     cfg['SCA']['IAM Coefficient F0'] = self.varsc IAMF0.get()
411     cfg['SCA']['IAM Coefficient F1'] = self.varsc IAMF1.get()
412     cfg['SCA']['IAM Coefficient F2'] = self.varsc IAMF2.get()
413     cfg['SCA']['Track Twist'] = self.varscatracktwist.get()
414     cfg['SCA']['Geom.Accuracy'] = self.varscageoaccuracy.get()
415     cfg['SCA']['Reflectance'] = self.varscareflectance.get()
416     cfg['SCA']['Cleanliness'] = self.varscacleanliness.get()
417     cfg['SCA']['Factor'] = self.varscafactor.get()
418     cfg['SCA']['Availability'] = self.varscavailability.get()
419
420     # HCE configuration

```

```

421     cfg['HCE']['Name'] = self.varhcename.get()
422     cfg['HCE']['Length'] = self.varhcelength.get()
423     cfg['HCE']['Bellows ratio'] = self.varbellowsratio.get()
424     cfg['HCE']['Shield shading'] = self.varshieldshading.get()
425     cfg['HCE']['Absorber tube inner diameter'] = self.varhcedri.get()
426     cfg['HCE']['Absorber tube outer diameter'] = self.varhcedro.get()
427     cfg['HCE']['Glass envelope inner diameter'] = self.varhcedgi.get()
428     cfg['HCE']['Glass envelope outer diameter'] = self.varhcedgo.get()
429     cfg['HCE']['Min Reynolds'] = self.varhceminreynolds.get()
430     cfg['HCE']['Inner surface roughness'] = self.varhceinnerroughness.get()
431     cfg['HCE']['Envelope transmittance'] = self.varhcetransmittance.get()
432     cfg['HCE']['Absorber emittance factor A0'] = self.varhceemittanceA0.get()
433     cfg['HCE']['Absorber emittance factor A1'] = self.varhceemittanceA1.get()
434     cfg['HCE']['Absorber absorptance'] = self.varhceabsorptance.get()
435     cfg['HCE']['Brackets'] = self.varhcebrackets.get()
436
437
438 # loop Configuration
439 cfg['loop']['scas'] = self.varscas.get()
440 cfg['loop']['hces'] = self.varhces.get()
441 cfg['loop']['rated_tin'] = self.vartin.get()
442 cfg['loop']['rated_tout'] = self.vartout.get()
443 cfg['loop']['rated_pin'] = self.varpin.get()
444 cfg['loop']['rated_pout'] = self.varpout.get()
445 cfg['loop']['tmin'] = self.vartmin.get()
446 cfg['loop']['tmax'] = self.vartmax.get()
447 cfg['loop']['rated_massflow'] = self.varratedmassflow.get()
448 cfg['loop']['min_massflow'] = self.varrecirculation.get()
449 cfg['loop']['row_spacing'] = self.varrowspacing.get()
450 cfg['loop']['Tracking Type'] = self.varscatrackingtype.get()
451
452
453 datarow=list(self.solarfield_table.table_data)
454 dictkeys =['name', 'loops']
455
456 subfields = []
457
458 for r in datarow:
459     sf = {}
460     index = 0
461     for v in r:
462         k = dictkeys[index]
463         sf[k]= self.to_number(v)
464         index += 1
465     subfields.append(sf)
466
467 cfg.update({'subfields': subfields})
468
469 if self.varsimdatatype.get() == 2:
470
471     cfg['tags'] = dict({})
472
473     for r in self.columns_table.table_data:
474         cfg['tags'][r[0]] = r[2]
475
476 return cfg
477
478 def save_as_JSON(self, cfg, filename=None):
479
480     if filename is None:

```

```

481         f = asksaveasfile(initialdir=self._DIR['saved_configurations'],
482                             title='choose your file name',
483                             filetypes=[('JSON files', '*.json')],
484                             defaultextension='json')
485     else:
486         f = open(filename, 'w')
487
488     if f is not None:
489         f.write(json.dumps(cfg,
490                             indent= True,
491                             ensure_ascii=False))
492         f.close()
493     else:
494         pass
495
496
497     def __insert_rows__(self, table):
498
499         lst = []
500         for c in table._multicolumn_listbox._columns:
501             lst.append('')
502
503             rows = table.number_of_rows
504             table.insert_row(lst, index = rows)
505
506     def __del_rows__(self, table):
507         table.delete_all_selected_rows()
508
509     def run_simulation(self):
510         # import subprocess
511         import threading
512         try:
513             cfg = self.generate_json()
514
515             SIM = cs.SolarFieldSimulation(cfg)
516             FLAG_00 = datetime.now()
517             hilo = threading.Thread(target=SIM.runSimulation())
518             hilo.start()
519             FLAG_01 = datetime.now()
520             DELTA_01 = FLAG_01 - FLAG_00
521
522         except Exception as e:
523             print("serialization failed", e)
524
525     def help_help():
526         pass
527
528     def help_about():
529         pass
530
531     def simulation_exit(self):
532
533         self.root.destroy()
534
535     def to_number(self, s):
536
537         try:
538             i = int(s)
539             return(i)
540         except ValueError:

```

```

541         pass
542     try:
543         f = float(s)
544         return(f)
545     except ValueError:
546         pass
547     return s
548
549
550 def buildNotebook(self):
551
552     self.nb.add(self.fr_simulation, text='Simulation Configuration', padding=2)
553     self.nb.add(self.fr_solarfield, text=' Solar Field Layout ', padding=2)
554     self.nb.add(self.fr_fluid, text=' HTF ', padding=2)
555     self.nb.add(self.fr_sca, text='SCA: Solar Collector Assembly', padding=2)
556     self.nb.add(self.fr_hce, text='HCE: Heat Collector Element', padding=2)
557     self.nb.select(self.fr_simulation)
558     self.nb.enable_traversal()
559     self.nb.pack()
560
561 # Simulaton Configuration tab
562 def simulationLoadDialog(self, title, labeltext=''):
563
564     path = askopenfilename(initialdir = self._DIR['saved_configurations'],
565                           title='choose your file',
566                           filetypes=[('JSON files', '*.json')])
567
568     with open(path) as cfg_file:
569         cfg = json.load(cfg_file,
570                         parse_float= float,
571                         parse_int= int)
572
573     self.varsimID.set(cfg['simulation']['ID'])
574     self.varsimdatatype.set(cfg['simulation']['datatype'])
575     self.varsimulation.set(cfg['simulation']['simulation'])
576     self.varbenchmark.set(cfg['simulation']['benchmark'])
577     self.varfastmode.set(cfg['simulation']['fastmode'])
578
579 def checkoptions(self):
580
581     var = self.varsimdatatype.get()
582
583     if var == 1: # Weather File
584         self.varbenchmark.set(False)
585         self.cbbenchmark['state'] ='disabled'
586         self.cbloadsitedata['state'] = 'normal'
587         self.bttagswizard['state']= 'disabled'
588         self.btloadtags['state']= 'disabled'
589         # self.tags_table.state('disabled')
590     elif var == 2: # Field Data File
591         self.varloadsitedata.set(False)
592         self.cbbenchmark['state']= 'normal'
593         self.cbloadsitedata['state'] = 'disabled'
594         self.bttagswizard['state']= 'normal'
595         self.btloadtags['state']= 'normal'
596         # self.tags_table.state(( 'active' ))
597     else:
598         pass
599
600 def checkfastmode(self):

```

```
601
602     var = self.varfastmode.get()
603
604     if var:
605         self.varfastmodetext.set('ON')
606     else:
607         self.varfastmodetext.set('OFF')
608
609 def buildSimulationFrame(self):
610
611     self.varsimID = tk.StringVar(self.fr_simulation)
612     self.varsimdatatype = tk.IntVar(self.fr_simulation)
613     self.tagslist = []
614     self.varsimulation = tk.BooleanVar(self.fr_simulation)
615     self.varbenchmark = tk.BooleanVar(self.fr_simulation)
616     self.varfastmode = tk.BooleanVar(self.fr_simulation)
617     self.varfastmodetext = tk.StringVar(self.fr_simulation)
618     self.varfirstdate = tk.StringVar(self.fr_simulation)
619     self.varlastdate = tk.StringVar(self.fr_simulation)
620
621     self.varmodelmaxerrtro = tk.DoubleVar(self.fr_simulation)
622     self.varmodelmaxerrrt = tk.DoubleVar(self.fr_simulation)
623     self.varmodelmaxerrpr = tk.DoubleVar(self.fr_simulation)
624
625     self.varfastmode.set(False)
626     self.checkfastmode()
627
628     self.varloadsitedata = tk.BooleanVar(self.fr_simulation)
629     self.varloadsitedata.set(False)
630
631     self.varsitename = tk.StringVar(self.fr_simulation)
632     self.varsitelat = tk.DoubleVar(self.fr_simulation)
633     self.varsitelong = tk.DoubleVar(self.fr_simulation)
634     self.varsitealt = tk.DoubleVar(self.fr_simulation)
635
636     self.vardatafileurl = tk.StringVar(self.fr_simulation)
637     self.vardatafilename = tk.StringVar(self.fr_simulation)
638     self.vardatafilepath = tk.StringVar(self.fr_simulation)
639
640     self.lbsimID = ttk.Label(
641         self.fr_simulation,
642         text='Simulation ID').grid(
643             row=0, column=0, sticky='W', padx=2, pady=5)
644     self.ensimID = ttk.Entry(
645         self.fr_simulation,
646         textvariable=self.varsimID).grid(
647             row=0, column=1, sticky='W', padx=2, pady=5)
648
649     self.lbmodelname = ttk.Label(
650         self.fr_simulation,
651         text='Model name').grid(
652             row=0, column=2, sticky='E', padx=2, pady=5)
653
654     self.cmbmodelname = ttk.Combobox(self.fr_simulation)
655     self.cmbmodelname['values'] = self._MODELS
656     self.cmbmodelname['state'] = 'readonly'
657     self.cmbmodelname.current(0)
658     self.cmbmodelname.grid(row=0, column=3, sticky='W', padx=2, pady=5)
659
660     self.frame2= ttk.Frame(self.fr_simulation)
```

```

661     self.frame2.grid(row=1, column=0, columnspan=6, padx=2, pady=5)
662
663     self.lbmodelmaxerrtro = ttk.Label(
664         self.frame2,
665         text='Max. Err Tro').grid(
666             row=1, column=0, sticky='W', padx=2, pady=5)
667
668     self.enmodelmaxerrtro = ttk.Entry(
669         self.frame2, textvariable=self.varmodelmaxerrtro).grid(
670             row=1, column=1, sticky='W', padx=2, pady=5)
671
672     self.lbmodelmaxerrrt = ttk.Label(
673         self.frame2,
674         text='Max. Err Tout').grid(
675             row=1, column=2, sticky='W', padx=2, pady=5)
676
677     self.enmodelmaxerrrt = ttk.Entry(
678         self.frame2, textvariable=self.varmodelmaxerrrt).grid(
679             row=1, column=3, sticky='W', padx=2, pady=5)
680
681     self.lbmodelmaxerrpr = ttk.Label(
682         self.frame2,
683         text='Max. Err PR').grid(
684             row=1, column=4, sticky='W', padx=2, pady=5)
685
686     self.enmodelmaxerrpr = ttk.Entry(
687         self.frame2, textvariable=self.varmodelmaxerrpr).grid(
688             row=1, column=5, sticky='W', padx=2, pady=5)
689
690     self.lbdatatype = ttk.Label(
691         self.fr_simulation,
692         text='Choose a Data Source Type:').grid(
693             row=2, column=0, columnspan = 2, sticky='W', padx=2, pady=5)
694
695     self.rbweather = tk.Radiobutton(
696         self.fr_simulation,
697         padx=5,
698         text = 'Weather File',
699         variable=self.varsimdatatype, value=1,
700         command=lambda: self.checkoptions()).grid(
701             row=3, column=0, sticky='W', padx=2, pady=5)
702
703 # RadioButton for Field Data File
704     self.rbfielddata = tk.Radiobutton(
705         self.fr_simulation,
706         padx=5,
707         text = 'Field Data File',
708         variable=self.varsimdatatype, value=2,
709         command=lambda: self.checkoptions()).grid(
710             row=3, column=1, sticky='W', padx=2, pady=5)
711
712     self.btselectdatasource = ttk.Button(
713         self.fr_simulation, text='Select File',
714         command=lambda: self.dataLoadDialog(
715             'Select File',
716             labeltext = 'Select File'))
717     self.btselectdatasource.grid(
718             row=4, column=0, sticky='W', padx=2, pady=5)
719
720 # Data source path

```

```
721     self.vardatafileurl.set('Data source file path...')  
722     self.lbdatasourcepath = ttk.Label(  
723         self.fr_simulation, textvariable=self.vardatafileurl).grid(  
724             row=4, column= 1, columnspan=4, sticky='W', padx=2, pady=5)  
725  
726     self.lbfirstdate = ttk.Label(  
727         self.fr_simulation, text='First Date').grid(  
728             row=5, column=0,sticky='W', padx=2, pady=5)  
729     self.enfirstdate = ttk.Entry(  
730         self.fr_simulation, textvariable=self.varfirstdate).grid(  
731             row=5, column=1, sticky='W', padx=2, pady=5)  
732  
733     self.lblastdate = ttk.Label(  
734         self.fr_simulation, text='Last Date').grid(  
735             row=5, column=2,sticky='E', padx=2, pady=5)  
736     self.enlastdate = ttk.Entry(  
737         self.fr_simulation, textvariable=self.varlastdate).grid(  
738             row=5, column=3, sticky='W', padx=2, pady=5)  
739  
740 # Checkbox for Simulation  
741     self.lbsimulation = ttk.Label(  
742         self.fr_simulation, text='Run test type...').grid(  
743             row=6, column=0, sticky='W', padx=2, pady=5)  
744     self.cbsimulation = ttk.Checkbutton(  
745         self.fr_simulation,  
746             text='Simulation',  
747             variable=self.varsimulation)  
748     self.cbsimulation.grid(  
749             row=6, column=1, sticky='W', padx=2, pady=5)  
750  
751     self.cbbenchmark = ttk.Checkbutton(  
752         self.fr_simulation,  
753             text='Benchmark',  
754             variable=self.varbenchmark)  
755     self.cbbenchmark.grid(  
756             row=6, column=2, sticky='W', padx=2, pady=5)  
757  
758     self.lbfastmode = ttk.Label(  
759         self.fr_simulation, text='Fast mode').grid(  
760             row=7, column=0, sticky='W', padx=2, pady=5)  
761     self.cbfastmode = ttk.Checkbutton(  
762         self.fr_simulation,  
763             textvariable=self.varfastmodetext,  
764             variable= self.varfastmode,  
765             command=lambda: self.checkfastmode()).grid(  
766             row=7, column=1, sticky='W', padx=2, pady=5)  
767  
768     self.lbsitename = ttk.Label(  
769         self.fr_simulation, text='Site').grid(  
770             row=8, column=0, sticky='W', padx=2, pady=5)  
771     self.ensitename = ttk.Entry(  
772         self.fr_simulation, textvariable=self.varsitename).grid(  
773             row=8, column=1, sticky='W', padx=2, pady=5)  
774  
775     self.cbloadsitedata = ttk.Checkbutton(  
776         self.fr_simulation,  
777             text='Load site data from weather file',  
778             variable=self.varloadsitedata)  
779     self.cbloadsitedata.grid(  
780             row=8, column=2, sticky='W', padx=2, pady=5)
```

```

781
782     self.lbsitelat = ttk.Label(
783         self.fr_simulation, text='Latitude [°]').grid(
784             row=10, column=0, sticky='W', padx=2, pady=5)
785     self.ensitelat = ttk.Entry(
786         self.fr_simulation, textvariable=self.varsitelat).grid(
787             row=10, column=1, sticky='W', padx=2, pady=5)
788     self.lbsitelong = ttk.Label(
789         self.fr_simulation, text='Longitude [°']).grid(
790             row=11, column=0, sticky='W', padx=2, pady=5)
791     self.ensitelong = ttk.Entry(
792         self.fr_simulation, textvariable=self.varsitelong).grid(
793             row=11, column=1, sticky='W', padx=2, pady=5)
794     self.lbsitealt = ttk.Label(
795         self.fr_simulation, text='Altitude [m]').grid(
796             row=12, column=0, sticky='W', padx=2, pady=5)
797     self.ensitealt = ttk.Entry(
798         self.fr_simulation, textvariable=self.varsitealt).grid(
799             row=12, column=1, sticky='W', padx=2, pady=5)
800
801     self.varsimdatatype.set(1) # 1 for Weather File, 2 for Field Data File
802     self.checkoptions()
803     self.fr_solarfield.update()
804
805
806 def solarfield_save_dialog(self, title, labeltext = '') :
807
808     #encoder.FLOAT_REPR = lambda o: format(o, '.2f')
809     f = asksaveasfile(initialdir = self._DIR['saved_configurations'],
810                         title='choose your file name',
811                         filetypes=[('JSON files', '*.json')],
812                         defaultextension = 'json')
813
814     cfg = dict({'solarfield' : {}})
815     cfg['solarfield'].update(dict({'name' : self.enname.get()}))
816     cfg['solarfield'].update(dict({'rated_tin' : self.enratedtin.get()}))
817     cfg['solarfield'].update(dict({'rated_tout' : self.enratedtout.get()}))
818     cfg['solarfield'].update(dict({'rated_pin' : self.enratedpin.get()}))
819     cfg['solarfield'].update(dict({'rated_pout' : self.enratedpout.get()}))
820     cfg['solarfield'].update(dict({'rated_massflow' :
821         self.enratedmassflow.get()}))
822     cfg['solarfield'].update(dict({'min_massflow' :
823         self.enrecirculationmassflow.get()}))
824     cfg['solarfield'].update(dict({'tmin' : self.entmin.get()}))
825     cfg['solarfield'].update(dict({'tmax' : self.entmax.get()}))
826     cfg['solarfield'].update(dict({'loop': dict({'scas': self.enscas.get(),
827                                         'hces': self.enhces.get()})}))
828
829
830     subfields = []
831
832     for r in datarow:
833         sf = {}
834         index = 0
835         for v in r:
836             k = dictkeys[index]
837             sf[k]= self.to_number(v)
838             index += 1

```

```

839         subfields.append(sf)
840
841     cfg['solarfield'].update({'subfields' : subfields})
842     # cfg_settings['solarfield'].update(dict(cfg))
843     f.write(json.dumps(cfg))
844     f.close()
845
846 def showTagsTable(self):
847
848     self.msg = tk.Tk()
849     #self.fr_tags = tk.Frame(self.msg, )
850     self.strtags = tk.StringVar(self.msg)
851
852     for row in self.tagslist:
853         self.strtags.set(self.strtags.get() + '# ' +
854                         str(row[0]) + ' ---> ' +
855                         str(row[1]) + '\n')
856
857     self.lbtagslist = ttk.Label(self.msg,
858                                 textvariable =self.strtags).pack()
859
860     self.msg.mainloop()
861
862 def openTagsWizard(self):
863
864     tags_table = []
865
866     for tag in self.tagslist:
867         tags_table.append(['', tag])
868
869     columns_names = []
870
871     if self.varsimdatatype.get() == 2: # Data from field data file
872
873         columns_names.append(['DNI', '', ''])
874         columns_names.append(['Wspd', '', ''])
875         columns_names.append(['DryBulb', '', ''])
876         columns_names.append(['Pressure', '', ''])
877         columns_names.append(['GrossPower', '', ''])
878         columns_names.append(['AuxPower', '', ''])
879         columns_names.append(['NetPower', '', ''])
880
881     if self.varbenchmark.get():
882
883         for row in self.solarfield_table.table_data:
884             columns_names.append(['SB.'+row[0]+'.a.mf', '', ''])
885             columns_names.append(['SB.'+row[0]+'.a.tin', '', ''])
886             columns_names.append(['SB.'+row[0]+'.a.tout', '', ''])
887             columns_names.append(['SB.'+row[0]+'.a.pin', '', ''])
888             columns_names.append(['SB.'+row[0]+'.a.pout', '', ''])
889
890     self.columns_table.table_data = columns_names
891
892     self.showTagsTable()
893
894 def loadSelectedTags(self):
895
896     index = 0
897     for row in self.columns_table.table_data:
898         tag_index = self.to_number(row[1])

```

```

899     if  isinstance(tag_index, int):
900         new_row=[row[0], row[1], self.tagslist[tag_index-1][1]]
901         self.columns_table.update_row(
902             index,new_row)
903     index += 1
904
905 def buildSolarFieldFrame(self):
906
907     self.varsolarfieldname = tk.StringVar(self.fr_solarfield)
908     self.lbname = ttk.Label(self.fr_solarfield, text ='Solar Field Name' )
909     self.lbname.grid(row=0, column=0, sticky='W', padx=5, pady=5)
910     self.enname = ttk.Entry(self.fr_solarfield,
textvariable=self.varsolarfieldname)
911     self.enname.grid(row=0, column=1, sticky='W', padx=5, pady=5)
912
913     self.vartin = tk.DoubleVar(self.fr_solarfield)
914     self.lbratedtin = ttk.Label(self.fr_solarfield, text='Rated Tin [K]')
915     self.lbratedtin.grid(row=1, column=0, sticky='W', padx=5, pady=5)
916     self.enratedtin = ttk.Entry(self.fr_solarfield, textvariable=self.vartin)
917     self.enratedtin.grid(row=1, column=1, sticky='W', padx=5, pady=5)
918
919     self.vartout = tk.DoubleVar(self.fr_solarfield)
920     self.lbratedtout = ttk.Label(self.fr_solarfield, text='Rated Tout [K]')
921     self.lbratedtout.grid(row=1, column=2, sticky='W', padx=5, pady=5)
922     self.enratedtout = ttk.Entry(self.fr_solarfield, textvariable=self.vartout)
923     self.enratedtout.grid(row=1, column=3, sticky='W', padx=5, pady=5)
924
925     self.varpin = tk.DoubleVar(self.fr_solarfield)
926     self.lbratedpin = ttk.Label(self.fr_solarfield, text='Rated Pin [Pa]')
927     self.lbratedpin.grid(row=2, column=0, sticky='W', padx=5, pady=5)
928     self.enratedpin = ttk.Entry(self.fr_solarfield, textvariable=self.varpin)
929     self.enratedpin.grid(row=2, column=1, sticky='W', padx=5, pady=5)
930
931     self.varpout = tk.DoubleVar(self.fr_solarfield)
932     self.lbratedpout = ttk.Label(self.fr_solarfield, text='Rated Pout [Pa]')
933     self.lbratedpout.grid(row=2, column=2, sticky='W', padx=5, pady=5)
934     self.enratedpout = ttk.Entry(self.fr_solarfield, textvariable=self.varpout)
935     self.enratedpout.grid(row=2, column=3, sticky='W', padx=5, pady=5)
936
937     self.vartmin = tk.DoubleVar(self.fr_solarfield)
938     self.lbtmin = ttk.Label(self.fr_solarfield, text='Tmin [K]')
939     self.lbtmin.grid(row=3, column=0, sticky='W', padx=5, pady=5)
940     self.entmin = ttk.Entry(self.fr_solarfield, textvariable=self.vartmin)
941     self.entmin.grid(row=3, column=1, sticky='W', padx=5, pady=5)
942
943     self.vartmax = tk.DoubleVar(self.fr_solarfield)
944     self.lbtmax = ttk.Label(self.fr_solarfield, text='Tmax [K]')
945     self.lbtmax.grid(row=3, column=2, sticky='W', padx=5, pady=5)
946     self.entmax = ttk.Entry(self.fr_solarfield, textvariable=self.vartmax)
947     self.entmax.grid(row=3, column=3, sticky='W', padx=5, pady=5)
948
949     self.varratedmassflow = tk.DoubleVar(self.fr_solarfield)
950     self.lbratedmassflow = ttk.Label(
951         self.fr_solarfield, text='Loop Rated massflow [kg/s]')
952     self.lbratedmassflow.grid(row=4, column=0, sticky='W', padx=5, pady=5)
953     self.enratedmassflow = ttk.Entry(
954         self.fr_solarfield, textvariable=self.varratedmassflow)
955     self.enratedmassflow.grid(row=4, column=1, sticky='W', padx=5, pady=5)
956
957     self.varrecirculation = tk.DoubleVar(self.fr_solarfield)

```

```

958     self.lbrecirculationmassflow = ttk.Label(
959         self.fr_solarfield, text='Loop minimum massflow [kg/s]')
960     self.lbrecirculationmassflow.grid(
961         row=4, column=2, sticky='W', padx=5, pady=5)
962     self.enrecirculationmassflow = ttk.Entry(self.fr_solarfield,
963         textvariable=self.varrecirculation)
964     self.enrecirculationmassflow.grid(
965         row=4, column=3, sticky='W', padx=5, pady=5)
966
967     self.varrowspacing = tk.DoubleVar(self.fr_solarfield)
968     self.lbrowspacing = ttk.Label(
969         self.fr_solarfield, text='Row Spacing [m]')
970     self.lbrowspacing.grid(row=5, column=0, sticky='W', padx=5, pady=5)
971     self.enrowspacing = ttk.Entry(
972         self.fr_solarfield, textvariable=self.varrowspacing)
973     self.enrowspacing.grid(row=5, column=1, sticky='W', padx=5, pady=5)
974
975     self.varscattrackingtype = tk.IntVar(self.fr_solarfield)
976
977     self.lbscattrackingtype = ttk.Label(
978         self.fr_solarfield,
979         text='Tracking Axis (1: N-S, 2: E-W)').grid(
980             row=5, column=2, sticky='W', padx=2, pady=5)
981
982 # RadioButton for Tracking Type (Tracking Axis 1: N-S, 2: E-W)
983     self.rbtrackingtype = tk.Radiobutton(
984         self.fr_solarfield,
985         padx=2,
986         text = 'Tracking Axis N-S',
987         variable= self.varscattrackingtype, value=1).grid(
988             row=5, column=3, sticky='W', padx=2, pady=5)
989
990 # RadioButton for Tracking Type (Tracking Axis 1: N-S, 2: E-W)
991     self.rbtrackingtype = tk.Radiobutton(
992         self.fr_solarfield,
993         padx=2,
994         text = 'Tracking Axis E-W',
995         variable= self.varscattrackingtype, value=2).grid(
996             row=5, column=4, sticky='W', padx=2, pady=5)
997
998     self.varscas = tk.IntVar(self.fr_solarfield)
999     self.lbscas = ttk.Label(self.fr_solarfield, text='SCAs per Loop')
1000    self.lbscas.grid(row=6, column=0, sticky='W', padx=5, pady=5)
1001    self.enscas = ttk.Entry(self.fr_solarfield, textvariable=self.varscas)
1002    self.enscas.grid(row=6, column=1, sticky='W', padx=5, pady=5)
1003
1004    self.varhces = tk.IntVar(self.fr_solarfield)
1005    self.lbhces = ttk.Label(self.fr_solarfield, text='HCEs per SCA')
1006    self.lbhces.grid(row=6, column=2, sticky='W', padx=5, pady=5)
1007    self.enhces = ttk.Entry(self.fr_solarfield, textvariable=self.varhces)
1008    self.enhces.bind('<Key>', lambda event: self.updateHCEperSCA())
1009    self.enhces.grid(row=6, column=3, sticky='W', padx=5, pady=5)
1010
1011    self.solarfield_table = table.Tk_Table(
1012        self.fr_solarfield,
1013        ['SUBFIELD NAME', 'NUMBER OF LOOPS'],
1014        row_numbers=True,
1015        stripped_rows=('white', '#f2f2f2'),
1016        select_mode='none',
1017        cell_anchor='center',

```

```

1017         adjust_heading_to_content=True)
1018     self.solarfield_table.grid(
1019         row=7, column=0, columnspan =2, padx=5, pady=5)
1020
1021     self.columns_table = table.Tk_Table(
1022         self.fr_solarfield,
1023         [ ' COLUMN      ', ' TAG #' , '          TAG        ' ],
1024         row_numbers=True,
1025         stripped_rows=('white', '#f2f2f2'),
1026         select_mode='none',
1027         cell_anchor='center',
1028         adjust_heading_to_content=True)
1029     self.columns_table.grid(
1030         row=7, column=2, columnspan=2, padx=5, pady=5)
1031
1032     self.frame0 = ttk.Frame(self.fr_solarfield)
1033     self.frame0.grid(row=8, column=0, columnspan=2, padx=2, pady=5)
1034
1035     self.btnewrow = ttk.Button(
1036         self.frame0,
1037         text='Insert',
1038         command=lambda: self.__insert_rows__(self.solarfield_table))
1039     self.btnewrow.pack(side=tk.LEFT)
1040
1041     self.btdelrows = ttk.Button(
1042         self.frame0,
1043         text='Delete',
1044         command=lambda: self.__del_rows__(self.solarfield_table))
1045     self.btdelrows.pack(side=tk.LEFT)
1046
1047     self.frame1 = ttk.Frame(self.fr_solarfield)
1048     self.frame1.grid(row=8, column=2, columnspan=2, padx=2, pady=5)
1049
1050     self.bttagswizard = ttk.Button(
1051         self.frame1,
1052         text='Open TAGS wizard',
1053         command=lambda: self.openTagsWizard())
1054     self.bttagswizard.pack(side=tk.LEFT)
1055
1056     self.btloadtags = ttk.Button(
1057         self.frame1,
1058         text='Load TAGs',
1059         command=lambda: self.loadSelectedTags())
1060     self.btloadtags.pack(side=tk.LEFT)
1061
1062 # Site & Weather contruction Tab
1063 def dataLoadDialog(self, title, labeltext=''):
1064
1065     if self.varsimdatatype.get() == 1:
1066         path = askopenfilename(
1067             initialdir=self._DIR['weather_files'],
1068             title='choose your file',
1069             filetypes=(( 'TMY files' , ' *.tm2 '),
1070                         ( 'TMY files' , ' *.tm3 '),
1071                         ( 'csv files' , ' *.csv '),
1072                         ( 'all files' , ' *.* ')))
1073
1074         self.vardatafilepath.set(os.path.dirname(path)+ '/')
1075         self.vardatafilename.set(os.path.basename(path))
1076         self.vardatafileurl.set(os.path.dirname(path)+ '/' +

```

```

1077                               os.path.basename(path))
1078             elif self.varsimdatatype.get() == 2:
1079                 path = askopenfilename(
1080                     initialdir=self._DIR['fielddata_files'],
1081                     title='choose your file',
1082                     filetypes=(( 'csv files', '*.csv'),
1083                               ('all files', '*.*')))
1084
1085             self.vardatafilepath.set(os.path.dirname(path)+'/')
1086             self.vardatafilename.set(os.path.basename(path))
1087             self.vardatafileurl.set(os.path.dirname(path)+'/' +
1088                                     os.path.basename(path))
1089             df = pd.read_csv(path, sep=';',
1090                               decimal= ',',
1091                               dayfirst=True,
1092                               index_col=0)
1093             count = 0
1094             for r in list(df):
1095                 count += 1
1096                 self.tagslist.append([count, r])
1097         else:
1098             tk.messagebox.showwarning(
1099                 title='Warning',
1100                 message='Check Data Source Selection')
1101
1102     if self.varloadsitedata.get():
1103
1104         strfilename, strext = os.path.splitext(path)
1105         if strext == '.csv':
1106             weatherdata = pvlib.iotools.tmy.read_tmy3(path)
1107             file = path
1108         elif (strext == '.tm2' or strext == '.tmy'):
1109             weatherdata = pvlib.iotools.tmy.read_tmy2(path)
1110             file = path
1111         elif strext == '.xls':
1112             pass
1113         else:
1114             print('unknow extension ', strext)
1115             return
1116
1117         self.varsitename.set(weatherdata[1]['City'])
1118         self.varsitelat.set(weatherdata[1]['latitude'])
1119         self.varsitelong.set(weatherdata[1]['longitude'])
1120         self.varsitealt.set(weatherdata[1]['altitude'])
1121
1122     def select_fluid_from_csv(self):
1123
1124         # abre una ventana de selección de csv.
1125         self.loadWindow= tk.Tk()
1126         self.loadWindow.attributes('-fullscreen', False)
1127         self.loadWindow.title('Selection Window')
1128
1129         # Menu
1130         self.loadWindow.menubar = tk.Menu(
1131             self.loadWindow)
1132         self.loadWindow.load_menu = tk.Menu(
1133             self.loadWindow.menubar, tearoff=0)
1134         self.loadWindow.load_menu.add_command(
1135             label='Open', command=self.open_csv())
1136         self.loadWindow.load_menu.add_separator()

```

```

1137     self.loadWindow.load_menu.add_command(
1138         label='Exit', command=None)
1139     self.loadWindow.menuBar.add_cascade(
1140         label='Select File', menu=self.loadWindow.load_menu)
1141
1142     self.loadWindow.config(menu=self.loadWindow.menuBar)
1143
1144 def open_csv(self, path=None):
1145
1146     df = pd.DataFrame()
1147
1148     try:
1149         if path is None:
1150             root = Tk()
1151             root.withdraw()
1152             path = askopenfilename(initialdir = '.\fielddata_files\' ,
1153                                     title='choose your file',
1154                                     filetypes=(('csv files','*.csv'),
1155                                     ('all files','*.*')))
1156             root.update()
1157             root.destroy()
1158
1159         if path is None:
1160             return
1161         else:
1162             strfilename, strext = os.path.splitext(path)
1163
1164         if  strext == '.csv':
1165             print('csv.....')
1166
1167             df = pd.read_csv(path, sep=';',
1168                               decimal= ',')
1169             file = path
1170         else:
1171             print('unknow extension ', strext)
1172             return
1173     else:
1174         strfilename, strext = os.path.splitext(path)
1175
1176         if  strext == '.csv':
1177             print('csv...')
1178             df = pd.read_csv(path, sep=';',
1179                               decimal= ',')
1180             file = path
1181         else:
1182             print('unknow extension ', strext)
1183             return
1184
1185     except Exception:
1186         raise
1187
1188 def load_fluid_library(self):
1189
1190     path = askopenfilename(initialdir = self._DIR['fluid_files'],
1191                           title='choose your file',
1192                           filetypes=[('JSON files', '*.json')])'
1193
1194     with open(path) as cfg_file:
1195         cfg = json.load(cfg_file, parse_float= float, parse_int= int)
1196

```

```

1197     self.fluid_list = cfg
1198
1199     self.cmbfluidname['values'] = [s['name'] for s in cfg ]
1200     self.cmbfluidname.current(0)
1201     self.load_fluid_parameters()
1202
1203 def load_fluid_parameters(self):
1204
1205     self.fluid_config = {}
1206
1207     for item in self.fluid_list:
1208
1209         if item['name'] == self.cmbfluidname.get():
1210             self.fluid_config = item
1211
1212     datarow=[ ]
1213
1214     for parameter in self.fluid_config.keys():
1215         if parameter in ['cp','mu','rho','kt','h','t']:
1216             datarow.append([parameter] + self.fluid_config[parameter])
1217
1218     self.fluid_table.table_data = datarow
1219
1220     self.varfluidname.set(self.fluid_config['name'])
1221     self.varfluidtmin.set(self.fluid_config['tmin'])
1222     self.varfluidtmax.set(self.fluid_config['tmax'])
1223
1224 def fluid_load_dialog(self):
1225
1226     path = askopenfilename(initialdir=self._DIR['fluid_files'],
1227                           title='choose your file',
1228                           filetypes=[('JSON files', '*.json')])
1229
1230     with open(path) as cfg_file:
1231         cfg = json.load(cfg_file, parse_float=float, parse_int=int)
1232
1233     cp_coefs = [[[]]*10
1234     rho_coefs = [[[]]*10
1235     mu_coefs = [[[]]*10
1236     kt_coefs = [[[]]*10
1237     h_coefs = [[[]]*10
1238     t_coefs = [[[]]*10
1239
1240     temp_cp = ['cp']
1241     temp_rho = ['rho']
1242     temp_mu = ['mu']
1243     temp_kt = ['kt']
1244     temp_h = ['h']
1245     temp_t = ['t']
1246
1247     grades = ['Factor']
1248     grades.extend([0, 1, 2, 3, 4, 5, 6, 7, 8])
1249
1250     temp_cp.extend(list(cfg['hot_fluid']['cp']))
1251     temp_rho.extend(list(cfg['hot_fluid']['rho']))
1252     temp_mu.extend(list(cfg['hot_fluid']['mu']))
1253     temp_kt.extend(list(cfg['hot_fluid']['kt']))
1254     temp_h.extend(list(cfg['hot_fluid']['h']))
1255     temp_t.extend(list(cfg['hot_fluid']['t']))
1256

```

```

1257     for index in range(len(temp_cp)):
1258         cp_coefs[index] = temp_cp[index]
1259     cp_coefs[1:] = [Decimal(s) for s in cp_coefs[1:]]
1260     for index in range(len(temp_rho)):
1261         rho_coefs[index] = temp_rho[index]
1262     rho_coefs[1:] = [Decimal(s) for s in rho_coefs[1:]]
1263     for index in range(len(temp_mu)):
1264         mu_coefs[index] = temp_mu[index]
1265     mu_coefs[1:] = [Decimal(s) for s in mu_coefs[1:]]
1266     for index in range(len(temp_kt)):
1267         kt_coefs[index] = temp_kt[index]
1268     kt_coefs[1:] = [Decimal(s) for s in kt_coefs[1:]]
1269     for index in range(len(temp_h)):
1270         h_coefs[index] = temp_h[index]
1271     h_coefs[1:] = [Decimal(s) for s in h_coefs[1:]]
1272     for index in range(len(temp_t)):
1273         t_coefs[index] = temp_t[index]
1274     t_coefs[1:] = [Decimal(s) for s in t_coefs[1:]]
1275
1276     datarow = []
1277     datarow.append(cp_coefs)
1278     datarow.append(rho_coefs)
1279     datarow.append(mu_coefs)
1280     datarow.append(kt_coefs)
1281     datarow.append(h_coefs)
1282     datarow.append(t_coefs)
1283
1284     self.fluid_table.table_data = datarow
1285     self.varfluidtmin.set(cfg['tmin'])
1286     self.varfluidtmax.set(cfg['tmax'])
1287     self.varfluidname.set(cfg['name'])
1288
1289     self.fr_fluid.update()
1290
1291 def checkfluid(self):
1292
1293     var = self.varfluidtable.get()
1294
1295     if var == 1: # Fluid from table
1296         self.cmbcoolpropID.set('')
1297         self.cmbcoolpropID['state'] = 'disabled'
1298         self.enfluidname['state'] = 'normal'
1299         self.btloadfluidcfg['state'] = 'normal'
1300         self.enfluidtmin['state'] = 'normal'
1301         self.enfluidtmax['state'] = 'normal'
1302         self.varfluidtmax.set('')
1303         self.varfluidtmin.set('')
1304         self.varcoolproptmin.set('')
1305         self.varcoolproptmax.set('')
1306     elif var == 2: # Fluid from CoolProp
1307         self.varcoolproptmax.set('')
1308         self.varcoolproptmin.set('')
1309         self.varfluidname.set('')
1310         self.cmbcoolpropID['state'] = 'readonly'
1311         self.enfluidname['state'] = 'disabled'
1312         self.btloadfluidcfg['state'] = 'disabled'
1313         self.enfluidtmin['state'] = 'disabled'
1314         self.enfluidtmax['state'] = 'disabled'
1315         self.varfluidtmax.set('')
1316         self.varfluidtmin.set('')

```

```

1317         self.varcoolproptmin.set('')
1318         self.varcoolproptmax.set('')
1319         self.fluid_table.table_data = []
1320     else:
1321         pass
1322
1323     def get_coolprop_data(self):
1324
1325         self.varcoolproptmin.set(CP.PropsSI("TMIN", self.cmbcoolpropID.get()))
1326         self.varcoolproptmax.set(CP.PropsSI("TMAX", self.cmbcoolpropID.get()))
1327
1328
1329     def buildFluidFrame(self):
1330
1331         self.varfluidtmin = tk.DoubleVar(self.fr_fluid)
1332         self.varfluidtmax = tk.DoubleVar(self.fr_fluid)
1333         self.varcoolproptmin = tk.DoubleVar(self.fr_fluid)
1334         self.varcoolproptmax = tk.DoubleVar(self.fr_fluid)
1335         self.varfluidname = tk.StringVar(self.fr_fluid)
1336         self.fluid_list = []
1337         self.varfluidtable = tk.IntVar(self.fr_fluid)
1338
1339         self.varfluidtable.set(1) # 1 for Table, 2 for CoolProp Library
1340
1341         # RadioButton for fluid from library
1342         self.rbfliuidlib = tk.Radiobutton(
1343             self.fr_fluid,
1344             padx=1,
1345             text='Fluid Data from CoolProp',
1346             variable=self.varfluidtable, value=2,
1347             command=lambda: self.checkfluid()).grid(
1348                 row=0, column=0, sticky='W', padx=1, pady=5)
1349
1350         self.lbcoolpropID = tk.Label(
1351             self.fr_fluid, text='CoolProp ID (INCOMP::xxxx)')
1352         self.lbcoolpropID.grid(row=0, column=1, sticky='W', padx=1, pady=5)
1353
1354         self.cmbcoolpropID = ttk.Combobox(self.fr_fluid)
1355         self.cmbcoolpropID.bind(
1356             "<<ComboboxSelected>>", lambda event: self.get_coolprop_data())
1357         self.cmbcoolpropID['values'] = self._COOLPROP_FLUIDS
1358         self.cmbcoolpropID['state'] = 'readonly'
1359         self.cmbcoolpropID.current(0)
1360         self.cmbcoolpropID.grid(row=0, column=2, sticky='W', padx=1, pady=5)
1361
1362         self.lbcoolproptmintext = tk.Label(
1363             self.fr_fluid,
1364             text='Tmin[K]')
1365         self.lbcoolproptmintext.grid(
1366             row=0, column=3, sticky='E', padx=1, pady=5)
1367
1368         self.lbcoolproptmin = tk.Label(
1369             self.fr_fluid,
1370             textvariable=self.varcoolproptmin)
1371         self.lbcoolproptmin.grid(
1372             row=0, column=4, sticky='E', padx=1, pady=5)
1373
1374         self.lbcoolproptmaxtext = tk.Label(
1375             self.fr_fluid,
1376             text='Tmax[K]')

```

```

1377     self.lbcoolproptmaxtext.grid(
1378         row=0, column=5, sticky='W', padx=1, pady=5)
1379
1380     self.lbcoolproptmax = tk.Label(
1381         self.fr_fluid,
1382         textvariable=self.varcoolproptmax)
1383     self.lbcoolproptmax.grid(
1384         row=0, column=6, sticky='W', padx=1, pady=5)
1385
1386
1387     self.separator = ttk.Separator(self.fr_fluid).grid(
1388         row=1, column=0, columnspan=99, sticky=(tk.W, tk.E))
1389
1390     # RadioButton for fluid from table
1391     self.rbfluidtable = tk.Radiobutton(
1392         self.fr_fluid,
1393         padx=2,
1394         text = 'Fluid Data from table',
1395         variable=self.varfluidtable, value=1,
1396         command=lambda: self.checkfluid()).grid(
1397             row=2, column=0, columnspan=2, sticky='W', padx=2, pady=5)
1398
1399     self.lbfluidname = ttk.Label(self.fr_fluid, text='Name')
1400     self.lbfluidname.grid(row=3, column=0, sticky='W', padx=2, pady=5)
1401     self.enfluidname = ttk.Entry(self.fr_fluid, textvariable=self.varfluidname)
1402     self.enfluidname.grid(row=3, column=1, sticky='W', padx=2, pady=5)
1403
1404     self.cmbfluidname = ttk.Combobox(self.fr_fluid)
1405     self.cmbfluidname.bind('<<ComboboxSelected>>',
1406                             lambda event: self.load_fluid_parameters())
1407     self.cmbfluidname['values'] = self.fluid_list
1408     self.cmbfluidname.grid(row=3, column=2, padx=5, pady=5)
1409
1410     self.btloadfluidcfg = ttk.Button(
1411         self.fr_fluid, text='Load config',
1412         command=lambda: self.load_fluid_library())
1413     self.btloadfluidcfg.grid(
1414         row=3, column=3, sticky='W', padx=2, pady=5)
1415
1416     self.lbfluidtmin = ttk.Label(self.fr_fluid, text='Tmin[K]')
1417     self.lbfluidtmin.grid(row=4, column=0, sticky='W', padx=2, pady=5)
1418     self.enfluidtmin = ttk.Entry(
1419         self.fr_fluid, textvariable=self.varfluidtmin)
1420     self.enfluidtmin.grid(row=4, column=1, sticky='W', padx=2, pady=5)
1421
1422     self.lbfluidtmax = ttk.Label(self.fr_fluid, text='Tmax [K]')
1423     self.lbfluidtmax.grid(row=4, column=2, sticky='W', padx=2, pady=5)
1424     self.enfluidtmax = ttk.Entry(self.fr_fluid, textvariable=self.varfluidtmax)
1425     self.enfluidtmax.grid(row=4, column=3, sticky='W', padx=2, pady=5)
1426
1427     self.fluid_table = table.Tk_Table(
1428         self.fr_fluid,
1429         ['Parameter',
1430          'A x^0', 'B x^1',
1431          'C x^2', 'D x^3',
1432          'E x^4', 'F x^5',
1433          'G x^6', 'H x^7',
1434          'I x^8'],
1435         row_numbers=True,
1436         stripped_rows=('white', '#f2f2f2'),

```

```

1437         select_mode='none',
1438         cell_anchor='e',
1439         adjust_heading_to_content=True)
1440     self.fluid_table.grid(
1441         row=5, column=0, columnspan=7, sticky='W', padx=2, pady=5)
1442
1443     self.checkfluid()
1444
1445 # SCA Construction tab
1446 def load_sca_library(self):
1447
1448     path = askopenfilename(initialdir=self._DIR['sca_files'],
1449                           title='choose your file',
1450                           filetypes=[('JSON files', '*.json')])
1451
1452     with open(path) as cfg_file:
1453         cfg = json.load(cfg_file, parse_float= float, parse_int= int)
1454
1455     self.sca_list = cfg
1456     self.cmbscaname['values'] = [s['Name'] for s in cfg]
1457     self.cmbscaname.current(0)
1458     self.load_sca_parameters()
1459     self.checkfluid()
1460
1461
1462 def load_sca_parameters(self):
1463
1464     self.sca_config = {}
1465
1466     for item in self.sca_list:
1467         if item['Name'] == self.cmbscaname.get():
1468             self.sca_config = item
1469
1470         self.varscaname.set(self.sca_config['Name'])
1471         self.varscalength.set(self.sca_config['SCA Length'])
1472         self.varscaaperture.set(self.sca_config['Aperture'])
1473         self.varscafocallen.set(self.sca_config['Focal Len'])
1474         self.varscIAMF0.set(self.sca_config['IAM Coefficient F0'])
1475         self.varscIAMF1.set(self.sca_config['IAM Coefficient F1'])
1476         self.varscIAMF2.set(self.sca_config['IAM Coefficient F2'])
1477         self.varscareflectance.set(self.sca_config['Reflectance'])
1478         self.varscageoaccuracy.set(self.sca_config['Geom.Accuracy'])
1479         self.varscatracktwist.set(self.sca_config['Track Twist'])
1480         self.varscacleanliness.set(self.sca_config['Cleanliness'])
1481         self.varscafactor.set(self.sca_config['Factor'])
1482         self.varscavailability.set(self.sca_config['Availability'])
1483         self.updateHCEperSCA()
1484
1485 def buildSCAFrame(self):
1486
1487     self.sca_list = []
1488
1489     self.varscaname = tk.StringVar(self.fr_sca)
1490     self.varscalength = tk.DoubleVar(self.fr_sca)
1491     self.varscaaperture = tk.DoubleVar(self.fr_sca)
1492     self.varscafocallen = tk.DoubleVar(self.fr_sca)
1493     self.varscIAMF0 = tk.DoubleVar(self.fr_sca)
1494     self.varscIAMF1 = tk.DoubleVar(self.fr_sca)
1495     self.varscIAMF2 = tk.DoubleVar(self.fr_sca)
1496     self.varscareflectance = tk.DoubleVar(self.fr_sca)

```

```
1497     self.varscageoaccuracy = tk.DoubleVar(self.fr_sca)
1498     self.varscatracktwist = tk.DoubleVar(self.fr_sca)
1499     self.varscacleanliness = tk.DoubleVar(self.fr_sca)
1500     self.varscafactor = tk.DoubleVar(self.fr_sca)
1501     self.varscaavailability = tk.DoubleVar(self.fr_sca)
1502
1503     self.lbscaname = ttk.Label(
1504         self.fr_sca,
1505         text='SCA base name').grid(
1506             row=0, column=0, sticky='W', padx=2, pady=5)
1507
1508     self.enscaname = ttk.Entry(
1509         self.fr_sca,
1510         textvariable=self.varscaavailability).grid(
1511             row=0, column=1, sticky='W', padx=2, pady=5)
1512
1513     self.btscaload = ttk.Button(
1514         self.fr_sca, text='Load Library',
1515         command=lambda: self.load_sca_library())
1516     self.btscaload.grid(row=0, column=7, sticky='W', padx=2, pady=5)
1517
1518     self.cmbscaname = ttk.Combobox(self.fr_sca)
1519     self.cmbscaname.bind(
1520         '<>ComboboxSelected>>', lambda event: self.load_sca_parameters())
1521     self.cmbscaname['values'] = self.sca_list
1522     self.cmbscaname.grid(row=0, column=3, columnspan=4, padx=5, pady=5)
1523
1524     self.lbscalength = ttk.Label(
1525         self.fr_sca,
1526         text='SCA Length [m]').grid(
1527             row=1, column=0, sticky='W', padx=2, pady=5)
1528     self.enscalength = ttk.Entry(
1529         self.fr_sca,
1530         textvariable=self.varscalength)
1531     self.enscalength.bind('<Key>', lambda event: self.updateHCEperSCA())
1532     self.enscalength.grid(
1533             row=1, column=1, sticky='W', padx=2, pady=5)
1534
1535     self.lbscaaperture = ttk.Label(
1536         self.fr_sca,
1537         text='SCA aperture [m]').grid(
1538             row=2, column=0, sticky='W', padx=2, pady=5)
1539     self.enscaaperture = ttk.Entry(
1540         self.fr_sca,
1541         textvariable=self.varscaaperture).grid(
1542             row=2, column=1, sticky='W', padx=2, pady=5)
1543
1544     self.lbscafocallen = ttk.Label(
1545         self.fr_sca,
1546         text='SCA focal length [m]').grid(
1547             row=3, column=0, sticky='W', padx=2, pady=5)
1548     self.enscafocallen = ttk.Entry(
1549         self.fr_sca,
1550         textvariable=self.varscafocallen).grid(
1551             row=3, column=1, sticky='W', padx=2, pady=5)
1552
1553     self.lbscaIAMF0 = ttk.Label(
1554         self.fr_sca,
1555         text='SCA IAM factor F0 []').grid(
1556             row=4, column=0, sticky='W', padx=2, pady=5)
```

```
1557     self.enscaIAMF0 = ttk.Entry(
1558         self.fr_sca,
1559         textvariable=self.varscaIAMF0).grid(
1560             row=4, column=1, sticky='W', padx=2, pady=5)
1561
1562     self.lbscaIAMF1 = ttk.Label(
1563         self.fr_sca,
1564         text='SCA IAM factor F1 []').grid(
1565             row=5, column=0, sticky='W', padx=2, pady=5)
1566     self.enscaIAMF1 = ttk.Entry(
1567         self.fr_sca,
1568         textvariable=self.varscaIAMF1).grid(
1569             row=5, column=1, sticky='W', padx=2, pady=5)
1570
1571     self.lbscaIAMF2 = ttk.Label(
1572         self.fr_sca,
1573         text='SCA IAM factor F2 []').grid(
1574             row=6, column=0, sticky='W', padx=2, pady=5)
1575     self.enscaIAMF2 = ttk.Entry(
1576         self.fr_sca,
1577         textvariable=self.varscaIAMF2).grid(
1578             row=6, column=1, sticky='W', padx=2, pady=5)
1579
1580     self.lbscareflectance = ttk.Label(
1581         self.fr_sca,
1582         text='SCA mirror reflectance []').grid(
1583             row=7, column=0, sticky='W', padx=2, pady=5)
1584     self.enscareflectance = ttk.Entry(
1585         self.fr_sca,
1586         textvariable=self.varscareflectance).grid(
1587             row=7, column=1, sticky='W', padx=2, pady=5)
1588
1589     self.lbscageoaccuracy = ttk.Label(
1590         self.fr_sca,
1591         text='SCA geometric accuracy []').grid(
1592             row=8, column=0, sticky='W', padx=2, pady=5)
1593     self.enscageoaccuracy = ttk.Entry(
1594         self.fr_sca,
1595         textvariable=self.varscageoaccuracy).grid(
1596             row=8, column=1, sticky='W', padx=2, pady=5)
1597
1598     self.lbscatracketwist = ttk.Label(
1599         self.fr_sca,
1600         text='SCA Tracking Twist []').grid(
1601             row=9, column=0, sticky='W', padx=2, pady=5)
1602     self.enscatracketwist = ttk.Entry(
1603         self.fr_sca,
1604         textvariable=self.varscatracketwist).grid(
1605             row=9, column=1, sticky='W', padx=2, pady=5)
1606
1607     self.lbscacleanliness = ttk.Label(
1608         self.fr_sca,
1609         text='SCA Cleanliness []').grid(
1610             row=10, column=0, sticky='W', padx=2, pady=5)
1611     self.enscacleanliness = ttk.Entry(
1612         self.fr_sca,
1613         textvariable=self.varscacleanliness).grid(
1614             row=10, column=1, sticky='W', padx=2, pady=5)
1615
1616     self.lbscafactor = ttk.Label(
```

```

1617     self.fr_sca,
1618     text='SCA Factor []').grid(
1619         row=11, column=0, sticky='W', padx=2, pady=5)
1620 self.enscafactor = ttk.Entry(
1621     self.fr_sca,
1622     textvariable=self.varscafactor).grid(
1623         row=11, column=1, sticky='W', padx=2, pady=5)
1624
1625     self.lbscaavailability = ttk.Label(
1626         self.fr_sca,
1627         text='SCA Availability []').grid(
1628             row=12, column=0, sticky='W', padx=2, pady=5)
1629 self.ensaavailability = ttk.Entry(
1630     self.fr_sca,
1631     textvariable=self.varscaavailability).grid(
1632         row=12, column=1, sticky='W', padx=2, pady=5)
1633
1634 def load_hce_parameters(self):
1635
1636     self.hce_config = {}
1637
1638     for item in self.hce_list:
1639
1640         if item['Name'] == self.cmbhcename.get():
1641             self.hce_config = item
1642
1643             self.varhcename.set(self.hce_config['Name'])
1644             self.varhcedri.set(self.hce_config['Absorber tube inner diameter'])
1645             self.varhcedro.set(self.hce_config['Absorber tube outer diameter'])
1646             self.varhcedgi.set(self.hce_config['Glass envelope inner diameter'])
1647             self.varhcedgo.set(self.hce_config['Glass envelope outer diameter'])
1648             self.varhcelength.set(self.hce_config['Length'])
1649             self.varhceinnerroughness.set(self.hce_config['Inner surface roughness'])
1650             self.varhceminreynolds.set(self.hce_config['Min Reynolds'])
1651             self.varhceemittanceA0.set(self.hce_config['Absorber emittance factor A0'])
1652             self.varhceemittanceA1.set(self.hce_config['Absorber emittance factor A1'])
1653             self.varhceabsorptance.set(self.hce_config['Absorber absorptance'])
1654             self.varhcetransmittance.set(self.hce_config['Envelope transmittance'])
1655             self.varhcebrackets.set(self.hce_config['Brackets'])
1656             self.updateHCEperSCA()
1657
1658 def load_hce_library(self, path = None):
1659
1660     if path is None:
1661         path = askopenfilename(initialdir = self._DIR['hce_files'],
1662                             title='choose your file',
1663                             filetypes=[('JSON files', '*.json')])
1664
1665     with open(path) as cfg_file:
1666         cfg = json.load(cfg_file, parse_float= float, parse_int= int)
1667
1668         self.hce_list = cfg
1669         self.cmbhcename['values'] = [s['Name'] for s in cfg ]
1670         self.cmbhcename.current(0)
1671         self.load_hce_parameters()
1672
1673
1674 def updateHCEperSCA(self):
1675
1676     var1 = self.varscalelength.get()

```

```

1677     var2 = self.varhcelength.get()
1678
1679     if var2 != 0:
1680         self.varhcepersca.set(var1 // var2)
1681     else:
1682         self.varhcepersca.set(0.0)
1683
1684     var3 = self.varhcepersca.get()
1685     var4 = self.varhces.get()
1686
1687     if var3 >= var4:
1688         self.varhceperscatext.set('Max. ' + str(var3) + ' HCE per SCA')
1689     else:
1690         self.varhceperscatext.set(
1691             'Error: ' + str(var4) +
1692             ' exceeded max. HCE per SCA = ' + str(var3))
1693
1694 def buildHCEFrame(self):
1695
1696     self.hce_list = []
1697     self.varhcename = tk.StringVar(self.fr_hce)
1698     self.varhcedri = tk.DoubleVar(self.fr_hce)
1699     self.varhcedro = tk.DoubleVar(self.fr_hce)
1700     self.varhcedgi = tk.DoubleVar(self.fr_hce)
1701     self.varhcedgo = tk.DoubleVar(self.fr_hce)
1702     self.varhcelength = tk.DoubleVar(self.fr_hce)
1703     self.varhceinnerroughness = tk.DoubleVar(self.fr_hce)
1704     self.varhceminreynolds = tk.DoubleVar(self.fr_hce)
1705     self.varhcepersca = tk.DoubleVar(self.fr_hce)
1706     self.varhceperscatext = tk.StringVar(self.fr_hce)
1707     self.varhceabsorptance = tk.DoubleVar(self.fr_hce)
1708     self.varhcetransmittance = tk.DoubleVar(self.fr_hce)
1709     self.varhceemittanceA0 = tk.DoubleVar(self.fr_hce)
1710     self.varhceemittanceA1 = tk.DoubleVar(self.fr_hce)
1711     self.varbellowsratio = tk.DoubleVar(self.fr_hce)
1712     self.varshieldshading = tk.DoubleVar(self.fr_hce)
1713     self.varhcebrackets = tk.DoubleVar(self.fr_hce)
1714
1715     self.lbhcename = ttk.Label(
1716         self.fr_hce,
1717         text='HCE base name').grid(
1718             row=0, column=0, sticky='W', padx=2, pady=5)
1719     self.enhcename = ttk.Entry(
1720         self.fr_hce,
1721         textvariable=self.varhcename).grid(
1722             row=0, column=1, sticky='W', padx=2, pady=5)
1723
1724     self.cmbhcename = ttk.Combobox(self.fr_hce)
1725     self.cmbhcename.bind('<<ComboboxSelected>>', lambda event:
1726         self.load_hce_parameters())
1727         self.cmbhcename['values'] = self.hce_list
1728         self.cmbhcename.grid(row=0, column=2, padx=5, pady=5)
1729
1730         self.bthceload = ttk.Button(
1731             self.fr_hce, text='Load Library',
1732             command=lambda: self.load_hce_library())
1733         self.bthceload.grid(row=0, column=3, sticky='W', padx=2, pady=5)
1734
1735         self.lbhcedri = ttk.Label(
1736             self.fr_hce,

```

```
1736     text='HCE inner diameter [m]').grid(
1737         row=1, column=0, sticky='W', padx=2, pady=5)
1738 self.enhcedri = ttk.Entry(
1739     self.fr_hce,
1740     textvariable=self.varhcedri).grid(
1741         row=1, column=1, sticky='W', padx=2, pady=5)
1742
1743 self.lbhcedro = ttk.Label(
1744     self.fr_hce,
1745     text='HCE outer diameter [m]').grid(
1746         row=2, column=0, sticky='W', padx=2, pady=5)
1747 self.enhcedro = ttk.Entry(
1748     self.fr_hce,
1749     textvariable=self.varhcedro).grid(
1750         row=2, column=1, sticky='W', padx=2, pady=5)
1751
1752 self.lbhcedgi = ttk.Label(
1753     self.fr_hce,
1754     text='HCE glass inner diameter [m]').grid(
1755         row=3, column=0, sticky='W', padx=2, pady=5)
1756 self.enhcedgi = ttk.Entry(
1757     self.fr_hce,
1758     textvariable=self.varhcedgi).grid(
1759         row=3, column=1, sticky='W', padx=2, pady=5)
1760
1761 self.lbhcedgo = ttk.Label(
1762     self.fr_hce,
1763     text='HCE glass outer diameter [m]').grid(
1764         row=4, column=0, sticky='W', padx=2, pady=5)
1765 self.enhcedgo = ttk.Entry(
1766     self.fr_hce,
1767     textvariable=self.varhcedgo).grid(
1768         row=4, column=1, sticky='W', padx=2, pady=5)
1769
1770 self.lbhcelong = ttk.Label(
1771     self.fr_hce,
1772     text='HCE longitude [m]').grid(
1773         row=5, column=0, sticky='W', padx=2, pady=5)
1774 self.enhcelong = ttk.Entry(
1775     self.fr_hce,
1776     textvariable=self.varhcelength)
1777 self.enhcelong.bind('<Key>', lambda event: self.updateHCEperSCA())
1778 self.enhcelong.grid(row=5, column=1, sticky='W', padx=2, pady=5)
1779
1780 self.lbhcepersca = ttk.Label(
1781     self.fr_hce,
1782     textvariable=self.varhceperscatext).grid(
1783         row=5, column=2, sticky='W', padx=2, pady=5)
1784
1785 self.lbbellowsratio = ttk.Label(
1786     self.fr_hce,
1787     text='Bellows ratio [ ]').grid(
1788         row=6, column=0, sticky='W', padx=2, pady=5)
1789 self.enbellowsratio = ttk.Entry(
1790     self.fr_hce,
1791     textvariable=self.varbellowsratio).grid(
1792         row=6, column=1, sticky='W', padx=2, pady=5)
1793
1794 self.lbshieldshading = ttk.Label(
1795     self.fr_hce,
```

```
1796     text='Shield shading [ ]').grid(
1797         row=7, column=0, sticky='W', padx=2, pady=5)
1798 self.enshieldshading = ttk.Entry(
1799     self.fr_hce,
1800     textvariable=self.varshieldshading).grid(
1801         row=7, column=1, sticky='W', padx=2, pady=5)
1802
1803 self.lbbrackets = ttk.Label(
1804     self.fr_hce,
1805     text='Brackets spacing').grid(
1806         row=8, column=0, sticky='W', padx=2, pady=5)
1807 self.enbrackets = ttk.Entry(
1808     self.fr_hce,
1809     textvariable=self.varhcebrackets).grid(
1810         row=8, column=1, sticky='W', padx=2, pady=5)
1811
1812 self.lbhceinnerroughness = ttk.Label(
1813     self.fr_hce,
1814     text='HCE inner roughness [ ]').grid(
1815         row=10, column=0, sticky='W', padx=2, pady=5)
1816 self.enhceinnerroughness = ttk.Entry(
1817     self.fr_hce,
1818     textvariable=self.varhceinnerroughness).grid(
1819         row=10, column=1, sticky='W', padx=2, pady=5)
1820
1821 self.lbhceminreynolds = ttk.Label(
1822     self.fr_hce,
1823     text='HCE minimum Reynolds Number [ ]').grid(
1824         row=11, column=0, sticky='W', padx=2, pady=5)
1825 self.enhceminreynolds = ttk.Entry(
1826     self.fr_hce,
1827     textvariable=self.varhceminreynolds).grid(
1828         row=11, column=1, sticky='W', padx=2, pady=5)
1829
1830 self.lbhceabsorptance = ttk.Label(
1831     self.fr_hce,
1832     text='Absorptance [ ]').grid(
1833         row=12, column=0, sticky='W', padx=2, pady=5)
1834 self.enhceabsorptance = ttk.Entry(
1835     self.fr_hce,
1836     textvariable=self.varhceabsorptance).grid(
1837         row=12, column=1, sticky='W', padx=2, pady=5)
1838
1839 self.lbhctransmittance = ttk.Label(
1840     self.fr_hce,
1841     text='Transmittance [ ]').grid(
1842         row=13, column=0, sticky='W', padx=2, pady=5)
1843 self.enhcetransmittance = ttk.Entry(
1844     self.fr_hce,
1845     textvariable=self.varhcetransmittance).grid(
1846         row=13, column=1, sticky='W', padx=2, pady=5)
1847
1848 self.lbemittanceA0 = ttk.Label(
1849     self.fr_hce,
1850     text='Emittance Factor A0 [ ]').grid(
1851         row=14, column=0, sticky='W', padx=2, pady=5)
1852 self.enemittanceA0 = ttk.Entry(
1853     self.fr_hce,
1854     textvariable=self.varhceemittanceA0).grid(
1855         row=14, column=1, sticky='W', padx=2, pady=5)
```

```
1856
1857     self.lbemittanceA1 = ttk.Label(
1858         self.fr_hce,
1859         text='Emittance Factor A1 [ ]').grid(
1860             row=15, column=0, sticky='W', padx=2, pady=5)
1861     self.enemittanceA1 = ttk.Entry(
1862         self.fr_hce,
1863         textvariable=self.varhceemittanceA1).grid(
1864             row=15, column=1, sticky='W', padx=2, pady=5)
1865
1866
1867 if __name__ == '__main__':
1868
1869     try:
1870         from Tkinter import Tk
1871         import tkMessageBox as messagebox
1872
1873     except ImportError:
1874         from tkinter import Tk
1875         from tkinter import messagebox
1876
1877     interface = Interface()
1878     interface.root.mainloop()
```