

```
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 CoolProp.CoolProp as CP
15 import pvlib as pvlib
16 import tkinter as tk
17 from tkinter.filedialog import askopenfilename, asksaveasfile
18 import tkinter.ttk as ttk
19 from tkinter import messagebox
20 # recipe-580746-1.py from
21 # http://code.activestate.com/recipes/
22 # 580746-t kinter-treeview-like-a-table-or-multicolumn-listb/
23 import recipe5807461 as table
24 import json
25 from decimal import Decimal
26 from datetime import datetime
27 import pandas as pd
28
29
30 class Interface(object):
31
32     _COOLPROP_FLUIDS = ['Water', 'INCOMP::TVP1', 'INCOMP::S800']
33
34     _MODELS = ['Barbero4thOrder', 'Barbero1stOrder', 'BarberoSimplified']
35
36     _DIR = {
37         'saved_configurations': './saved_configurations/',
38         'site_files': './site_files/',
39         'fluid_files': './fluid_files',
40         'hce_files': './hce_files',
41         'sca_files': './sca_files',
42         'fielddata_files': './fielddata_files',
43         'weather_files': './weather_files'}
44
45     cfg_settings = {
46         'simulation': {},
47         'solar_plant': {},
48         'site': {},
49         'weather': {},
50         'hce': {},
51         'SCA': {},
52         'hot_hce': {},
53         'cold_hce': {},
54         'cycle': {},
55         'hce_model_settings': {},
56         'hce_scattered_params': {},
57         'sca_scattered_params': {}}
58
59     def __init__(self):
60
```

```

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

```

```

121     self.varfastmode.set(False)
122     self.vardatafilename.set('')
123     self.vardatafilepath.set('')
124     self.vardatafileurl.set('')
125     self.varfirstdate.set(pd.to_datetime('2000/01/01 1:00')).strftime(
126         '%Y/%m/%d %H:%M'))
127     self.varlastdate.set(pd.to_datetime('2000/12/31 1:00')).strftime(
128         '%Y/%m/%d %H:%M'))
129     self.cmbmodelname.set('')
130     self.varmodelmaxerrt.set(1.0)
131     self.varmodelmaxerrtro.set(0.1)
132     self.varmodelmaxerrpr.set(0.01)

133
134     self.varsitename.set(0)
135     self.varsitelat.set(0)
136     self.varsitelong.set(0)
137     self.varsitealt.set(0)
138     self.checkoptions()
139     self.checkfastmode()

140
141 # Solarfield configuration
142 self.solarfield_table.table_data = []
143 self.columns_table.table_data = []

144
145     self.vartin.set(0)
146     self.vartout.set(0)
147     self.varpin.set(0)
148     self.varpout.set(0)
149     self.vartmin.set(0)
150     self.vartmax.set(0)
151     self.varratedmassflow.set(0)
152     self.varrecirculation.set(0)
153     self.varscas.set(0)
154     self.varhces.set(0)
155     self.varrowspacing.set(0)
156     self.varscatrackingtype.set(1)
157     self.varfluidtable.set(1)
158     self.varfluidname.set('')
159     self.varfluidtmax.set(0)
160     self.varfluidtmin.set(0)
161     self.fluid_table.table_data = []
162     self.cmbcoolpropID.set('')
163     self.checkfluid()

164
165 # SCA configuration
166     self.varscaname.set('')
167     self.varscalength.set(0)
168     self.varscaperture.set(0)
169     self.varscafocallen.set(0)
170     self.varscIAMF0.set(0)
171     self.varscIAMF1.set(0)
172     self.varscIAMF2.set(0)
173     self.varscareflectance.set(0)
174     self.varscageoaccuracy.set(0)
175     self.varscatracktwist.set(0)
176     self.varscacleanliness.set(0)
177     self.varscafactor.set(0)
178     self.varscavailability.set(0)

179
180 # HCE configuration

```

```

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

```

```

241     self.varsitealt.set(cfg['site']['altitude']))
242     self.checkoptions()
243     self.checkfastmode()
244
245     # Solarfield configuration
246     list_subfields = []
247     for r in cfg['subfields']:
248         list_subfields.append(list(r.values()))
249
250     self.solarfield_table.table_data = list_subfields
251
252     if 'tags' in cfg.keys():
253         list_tags = []
254         for r in cfg['tags']:
255             list_tags.append([r, ' ', cfg['tags'][r]])
256         self.columns_table.table_data = list_tags
257
258         self.vartin.set(cfg['loop']['rated_tin'])
259         self.vartout.set(cfg['loop']['rated_tout'])
260         self.varpin.set(cfg['loop']['rated_pin'])
261         self.varpout.set(cfg['loop']['rated_pout'])
262         self.vartmin.set(cfg['loop']['tmin'])
263         self.vartmax.set(cfg['loop']['tmax'])
264         self.varratedmassflow.set(cfg['loop']['rated_massflow'])
265         self.varrecirculation.set(cfg['loop']['min_massflow'])
266         self.varscas.set(cfg['loop']['scas'])
267         self.varhces.set(cfg['loop']['hces'])
268         self.varrowspacing.set(cfg['loop']['row_spacing'])
269         self.varscatrackingtype.set(cfg['loop']['Tracking Type'])
270
271     # HTF configuration
272     fluid_table = []
273
274     if cfg['HTF']['source'] == 'table':
275         self.varfluidtable.set(1)
276         self.varfluidname.set(cfg['HTF']['name'])
277         self.varfluidtmax.set(cfg['HTF']['tmax'])
278         self.varfluidtmin.set(cfg['HTF']['tmin'])
279         fluid_table.append(['mu'] + cfg['HTF']['mu'])
280         fluid_table.append(['cp'] + cfg['HTF']['cp'])
281         fluid_table.append(['rho'] + cfg['HTF']['rho'])
282         fluid_table.append(['kt'] + cfg['HTF']['kt'])
283         fluid_table.append(['h'] + cfg['HTF']['h'])
284         fluid_table.append(['t'] + cfg['HTF']['t'])
285         self.fluid_table.table_data = fluid_table
286
287     elif cfg['HTF']['source'] == 'CoolProp':
288         self.varfluidtable.set(2)
289         self.cmbcoolpropID.set(cfg['HTF']['CoolPropID'])
290         self.varfluidtmax.set(cfg['HTF']['tmax'])
291         self.varfluidtmin.set(cfg['HTF']['tmin'])
292
293     self.checkfluid()
294
295     # SCA configuration
296     self.varscaname.set(cfg['SCA']['Name'])
297     self.varscalength.set(cfg['SCA']['SCA Length'])
298     self.varscaperture.set(cfg['SCA']['Aperture'])
299     self.varscafocallen.set(cfg['SCA']['Focal Len'])
300     self.varscIAMF0.set(cfg['SCA']['IAM Coefficient F0'])

```

```

301     self.varscaIAMF1.set(cfg['SCA']['IAM Coefficient F1'])
302     self.varscaIAMF2.set(cfg['SCA']['IAM Coefficient F2'])
303     self.varscareflectance.set(cfg['SCA']['Reflectance'])
304     self.varscageoaccuracy.set(cfg['SCA']['Geom.Accuracy'])
305     self.varscatracktwist.set(cfg['SCA']['Track Twist'])
306     self.varscacleanliness.set(cfg['SCA']['Cleanliness'])
307     self.varscafactor.set(cfg['SCA']['Factor'])
308     self.varscavailability.set(cfg['SCA']['Availability'])

309
310     # HCE configuration
311     self.varhcename.set(cfg['HCE']['Name'])
312     self.varhcedri.set(cfg['HCE']['Absorber tube inner diameter'])
313     self.varhcedro.set(cfg['HCE']['Absorber tube outer diameter'])
314     self.varhcedgi.set(cfg['HCE']['Glass envelope inner diameter'])
315     self.varhcedgo.set(cfg['HCE']['Glass envelope outer diameter'])
316     self.varhcelength.set(cfg['HCE']['Length'])
317     self.varbellowsratio.set(cfg['HCE']['Bellows ratio'])
318     self.varshieldshading.set(cfg['HCE']['Shield shading'])
319     self.varhceemittanceA0.set(cfg['HCE']['Absorber emittance factor A0'])
320     self.varhceemittanceA1.set(cfg['HCE']['Absorber emittance factor A1'])
321     self.varhceabsorptance.set(cfg['HCE']['Absorber absorptance'])
322     self.varhcetransmittance.set(cfg['HCE']['Envelope transmittance'])
323     self.varhceinnerroughness.set(cfg['HCE']['Inner surface roughness'])
324     self.varhceminreynolds.set(cfg['HCE']['Min Reynolds'])
325     self.varhcebrackets.set(cfg['HCE']['Brackets'])
326     self.updateHCEperSCA()

327
328     self.fr_fluid.update()
329     self.fr_hce.update()
330     self.fr_solarfield.update()
331     self.fr_sca.update()
332     self.fr_simulation.update()

333
334     def simulation_save(self):
335
336         self.save_as_JSON(self.generate_json(), self.filename)

337
338
339     def simulation_save_as(self):
340
341         self.save_as_JSON(self.generate_json())

342
343     def generate_json(self):
344
345         self.tagslist = []

346
347         cfg = dict({'simulation': {},
348                     'model': {},
349                     'site': {},
350                     'loop': {},
351                     'SCA': {},
352                     'HCE': {},
353                     'HTF': {}})
354
355
356         cfg['simulation']['ID'] = self.varsimID.get()
357         cfg['simulation']['datatype'] = self.varsimdatatype.get()
358         cfg['simulation']['simulation'] = self.varsimulation.get()
359         cfg['simulation']['benchmark'] = self.varbenchmark.get()
360         cfg['simulation']['fastmode'] = self.varfastmode.get()

```

```

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

```

```

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

```

```

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

```

```

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

```

```

601     var = self.varfastmode.get()
602
603     if var:
604         self.varfastmodetext.set('ON')
605     else:
606         self.varfastmodetext.set('OFF')
607
608 def buildSimulationFrame(self):
609
610     self.varsimID = tk.StringVar(self.fr_simulation)
611     self.varsimdatatype = tk.IntVar(self.fr_simulation)
612     self.tagslist = []
613     self.varsimulation = tk.BooleanVar(self.fr_simulation)
614     self.varbenchmark = tk.BooleanVar(self.fr_simulation)
615     self.varfastmode = tk.BooleanVar(self.fr_simulation)
616     self.varfastmodetext = tk.StringVar(self.fr_simulation)
617     self.varfirstdate = tk.StringVar(self.fr_simulation)
618     self.varlastdate = tk.StringVar(self.fr_simulation)
619
620     self.varmodelmaxerrtro = tk.DoubleVar(self.fr_simulation)
621     self.varmodelmaxerrrt = tk.DoubleVar(self.fr_simulation)
622     self.varmodelmaxerrpr = tk.DoubleVar(self.fr_simulation)
623
624     self.varfastmode.set(False)
625     self.checkfastmode()
626
627     self.varloadsitedata = tk.BooleanVar(self.fr_simulation)
628     self.varloadsitedata.set(False)
629
630     self.varsitename = tk.StringVar(self.fr_simulation)
631     self.varsitelat = tk.DoubleVar(self.fr_simulation)
632     self.varsiteelong = tk.DoubleVar(self.fr_simulation)
633     self.varsitealt = tk.DoubleVar(self.fr_simulation)
634
635     self.vardatafileurl = tk.StringVar(self.fr_simulation)
636     self.vardatafilename = tk.StringVar(self.fr_simulation)
637     self.vardatafilepath = tk.StringVar(self.fr_simulation)
638
639     self.lbsimID = ttk.Label(
640         self.fr_simulation,
641         text='Simulation ID').grid(
642             row=0, column=0, sticky='W', padx=2, pady=5)
643     self.ensimID = ttk.Entry(
644         self.fr_simulation,
645         textvariable=self.varsimID).grid(
646             row=0, column=1, sticky='W', padx=2, pady=5)
647
648     self.lbmodelname = ttk.Label(
649         self.fr_simulation,
650         text='Model name').grid(
651             row=0, column=2, sticky='E', padx=2, pady=5)
652
653     self.cmbmodelname = ttk.Combobox(self.fr_simulation)
654     self.cmbmodelname['values'] = self._MODELS
655     self.cmbmodelname['state'] = 'readonly'
656     self.cmbmodelname.current(0)
657     self.cmbmodelname.grid(row=0, column=3, sticky='W', padx=2, pady=5)
658
659     self.frame2= ttk.Frame(self.fr_simulation)
660     self.frame2.grid(row=1, column=0, columnspan=6, padx=2, pady=5)

```

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

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

1077     path = askopenfilename(
1078         initialdir=self._DIR['fielddata_files'],
1079         title='choose your file',
1080         filetypes=(( 'csv files', '*.csv'),
1081                     ('all files', '*.*')))
1082
1083     self.vardatafilepath.set(os.path.dirname(path)+'/')
1084     self.vardatafilename.set(os.path.basename(path))
1085     self.vardatafileurl.set(os.path.dirname(path)+'/ ' +
1086                             os.path.basename(path))
1087
1088 else:
1089     tk.messagebox.showwarning(
1090         title='Warning',
1091         message='Check Data Source Selection')
1092
1093 if self.varloadsitedata.get():
1094
1095     strfilename, strext = os.path.splitext(path)
1096     if strext == '.csv':
1097         weatherdata = pvlib.iotools.tmy.read_tmy3(path)
1098         file = path
1099     elif (strext == '.tm2' or strext == '.tmy'):
1100         weatherdata = pvlib.iotools.tmy.read_tmy2(path)
1101         file = path
1102     elif strext == '.xls':
1103         pass
1104     else:
1105         print('unknow extension ', strext)
1106         return
1107
1108     self.varsitename.set(weatherdata[1]['City'])
1109     self.varsitelat.set(weatherdata[1]['latitude'])
1110     self.varsitelong.set(weatherdata[1]['longitude'])
1111     self.varsitealt.set(weatherdata[1]['altitude'])
1112
1113 def select_fluid_from_csv(self):
1114
1115     # abre una ventana de selección de csv.
1116     self.loadWindow= tk.Tk()
1117     self.loadWindow.attributes('-fullscreen', False)
1118     self.loadWindow.title('Selection Window')
1119
1120     # Menu
1121     self.loadWindow.menubar = tk.Menu(
1122         self.loadWindow)
1123     self.loadWindow.load_menu = tk.Menu(
1124         self.loadWindow.menubar, tearoff=0)
1125     self.loadWindow.load_menu.add_command(
1126         label='Open', command=self.open_csv())
1127     self.loadWindow.load_menu.add_separator()
1128     self.loadWindow.load_menu.add_command(
1129         label='Exit', command=None)
1130     self.loadWindow.menubar.add_cascade(
1131         label='Select File', menu=self.loadWindow.load_menu)
1132
1133     self.loadWindow.config(menu=self.loadWindow.menubar)
1134
1135 def open_csv(self, path=None):
1136

```

```

1137     df = pd.DataFrame()
1138
1139     try:
1140         if path is None:
1141             root = Tk()
1142             root.withdraw()
1143             path = askopenfilename(initialdir = '.fielddata_files/',
1144                                     title='choose your file',
1145                                     filetypes=([('csv files','*.csv'),
1146                                     ('all files','*.*')))
1147             root.update()
1148             root.destroy()
1149
1150         if path is None:
1151             return
1152         else:
1153             strfilename, strext = os.path.splitext(path)
1154
1155         if strext == '.csv':
1156             print('csv.....')
1157
1158             df = pd.read_csv(path, sep=';',
1159                               decimal= ',')
1159             file = path
1160         else:
1161             print('unknow extension ', strext)
1162             return
1163
1164     else:
1165         strfilename, strext = os.path.splitext(path)
1166
1167         if strext == '.csv':
1168             print('csv...')
1169             df = pd.read_csv(path, sep=';',
1170                               decimal= ',')
1170             file = path
1171         else:
1172             print('unknow extension ', strext)
1173             return
1174
1175
1176     except Exception:
1177         raise
1178
1179 def load_fluid_library(self):
1180
1181     path = askopenfilename(initialdir = self._DIR['fluid_files'],
1182                           title='choose your file',
1183                           filetypes=[('JSON files', '*.json')])
1184
1185     with open(path) as cfg_file:
1186         cfg = json.load(cfg_file, parse_float= float, parse_int= int)
1187
1188     self.fluid_list = cfg
1189
1190     self.cmbfluidname['values'] = [s['name'] for s in cfg ]
1191     self.cmbfluidname.current(0)
1192     self.load_fluid_parameters()
1193
1194 def load_fluid_parameters(self):
1195
1196     self.fluid_config = {}

```

```

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

```

```

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

```

```

1317         self.varcoolproptmax.set(CP.PropsSI("TMAX", self.cmbcoolpropID.get()))
1318
1319
1320     def buildFluidFrame(self):
1321
1322         self.varfluidtmin = tk.DoubleVar(self.fr_fluid)
1323         self.varfluidtmax = tk.DoubleVar(self.fr_fluid)
1324         self.varcoolproptmin = tk.DoubleVar(self.fr_fluid)
1325         self.varcoolproptmax = tk.DoubleVar(self.fr_fluid)
1326         self.varfluidname = tk.StringVar(self.fr_fluid)
1327         self.fluid_list = []
1328         self.varfluidtable = tk.IntVar(self.fr_fluid)
1329
1330         self.varfluidtable.set(1) # 1 for Table, 2 for CoolProp Library
1331
1332         # RadioButton for fluid from library
1333         self.rbfliuidlib = tk.Radiobutton(
1334             self.fr_fluid,
1335             padx=1,
1336             text='Fluid Data from CoolProp',
1337             variable=self.varfluidtable, value=2,
1338             command=lambda: self.checkfluid()).grid(
1339                 row=0, column=0, sticky='W', padx=1, pady=5)
1340
1341         self.lbcoolpropID = tk.Label(
1342             self.fr_fluid, text='CoolProp ID (INCOMP::xxxx)')
1343         self.lbcoolpropID.grid(row=0, column=1, sticky='W', padx=1, pady=5)
1344
1345         self.cmbcoolpropID = ttk.Combobox(self.fr_fluid)
1346         self.cmbcoolpropID.bind(
1347             "<>ComboboxSelected>", lambda event: self.get_coolprop_data())
1348         self.cmbcoolpropID['values'] = self._COOLPROP_FLUIDS
1349         self.cmbcoolpropID['state'] = 'readonly'
1350         self.cmbcoolpropID.current(0)
1351         self.cmbcoolpropID.grid(row=0, column=2, sticky='W', padx=1, pady=5)
1352
1353         self.lbcoolproptmintext = tk.Label(
1354             self.fr_fluid,
1355             text='Tmin[K]')
1356         self.lbcoolproptmintext.grid(
1357             row=0, column=3, sticky='E', padx=1, pady=5)
1358
1359         self.lbcoolproptmin = tk.Label(
1360             self.fr_fluid,
1361             textvariable=self.varcoolproptmin)
1362         self.lbcoolproptmin.grid(
1363             row=0, column=4, sticky='E', padx=1, pady=5)
1364
1365         self.lbcoolproptmaxtext = tk.Label(
1366             self.fr_fluid,
1367             text='Tmax[K]')
1368         self.lbcoolproptmaxtext.grid(
1369             row=0, column=5, sticky='W', padx=1, pady=5)
1370
1371         self.lbcoolproptmax = tk.Label(
1372             self.fr_fluid,
1373             textvariable=self.varcoolproptmax)
1374         self.lbcoolproptmax.grid(
1375             row=0, column=6, sticky='W', padx=1, pady=5)
1376

```

```

1377
1378     self.separator = ttk.Separator(self.fr_fluid).grid(
1379         row=1, column=0, columnspan=99, sticky=(tk.W, tk.E))
1380
1381 # RadioButton for fluid from table
1382 self.rbfuidtable = tk.Radiobutton(
1383     self.fr_fluid,
1384     padx=2,
1385     text = 'Fluid Data from table',
1386     variable=self.varfluidtable, value=1,
1387     command=lambda: self.checkfluid()).grid(
1388         row=2, column=0, columnspan=2, sticky='W', padx=2, pady=5)
1389
1390 self.lbfluidname = ttk.Label(self.fr_fluid, text='Name')
1391 self.lbfluidname.grid(row=3, column=0, sticky='W', padx=2, pady=5)
1392 self.enfluidname = ttk.Entry(self.fr_fluid, textvariable=self.varfluidname)
1393 self.enfluidname.grid(row=3, column=1, sticky='W', padx=2, pady=5)
1394
1395 self.cmbfluidname = ttk.Combobox(self.fr_fluid)
1396 self.cmbfluidname.bind('<<ComboboxSelected>>',
1397                         lambda event: self.load_fluid_parameters())
1398 self.cmbfluidname['values'] = self.fluid_list
1399 self.cmbfluidname.grid(row=3, column=2, padx=5, pady=5)
1400
1401 self.btloadfluidcfg = ttk.Button(
1402     self.fr_fluid, text='Load config',
1403     command=lambda: self.load_fluid_library())
1404 self.btloadfluidcfg.grid(
1405     row=3, column=3, sticky='W', padx=2, pady=5)
1406
1407 self.lbfluidtmin = ttk.Label(self.fr_fluid, text='Tmin[K]')
1408 self.lbfluidtmin.grid(row=4, column=0, sticky='W', padx=2, pady=5)
1409 self.enfluidtmin = ttk.Entry(
1410     self.fr_fluid, textvariable=self.varfluidtmin)
1411 self.enfluidtmin.grid(row=4, column=1, sticky='W', padx=2, pady=5)
1412
1413 self.lbfluidtmax = ttk.Label(self.fr_fluid, text='Tmax [K]')
1414 self.lbfluidtmax.grid(row=4, column=2, sticky='W', padx=2, pady=5)
1415 self.enfluidtmax = ttk.Entry(self.fr_fluid, textvariable=self.varfluidtmax)
1416 self.enfluidtmax.grid(row=4, column=3, sticky='W', padx=2, pady=5)
1417
1418 self.fluid_table = table.Tk_Table(
1419     self.fr_fluid,
1420     ['Parameter',
1421      '          A x^0', '          B x^1',
1422      '          C x^2', '          D x^3',
1423      '          E x^4', '          F x^5',
1424      '          G x^6', '          H x^7',
1425      '          I x^8'],
1426     row_numbers=True,
1427     stripped_rows=('white', '#f2f2f2'),
1428     select_mode='none',
1429     cell_anchor='e',
1430     adjust_heading_to_content=True)
1431 self.fluid_table.grid(
1432     row=5, column=0, columnspan=7, sticky='W', padx=2, pady=5)
1433
1434     self.checkfluid()
1435
1436 # SCA Construction tab

```

```

1437     def load_sca_library(self):
1438
1439         path = askopenfilename(initialdir=self._DIR['sca_files'],
1440                               title='choose your file',
1441                               filetypes=[('JSON files', '*.json')])
1442
1443         with open(path) as cfg_file:
1444             cfg = json.load(cfg_file, parse_float= float, parse_int= int)
1445
1446             self.sca_list = cfg
1447             self.cmbscaname['values'] = [s['Name'] for s in cfg]
1448             self.cmbscaname.current(0)
1449             self.load_sca_parameters()
1450             self.checkfluid()
1451
1452
1453     def load_sca_parameters(self):
1454
1455         self.sca_config = {}
1456
1457         for item in self.sca_list:
1458             if item['Name'] == self.cmbscaname.get():
1459                 self.sca_config = item
1460
1461             self.varscname.set(self.sca_config['Name'])
1462             self.varscalength.set(self.sca_config['SCA Length'])
1463             self.varscaaperture.set(self.sca_config['Aperture'])
1464             self.varscafocallen.set(self.sca_config['Focal Len'])
1465             self.varscIAMF0.set(self.sca_config['IAM Coefficient F0'])
1466             self.varscIAMF1.set(self.sca_config['IAM Coefficient F1'])
1467             self.varscIAMF2.set(self.sca_config['IAM Coefficient F2'])
1468             self.varscareflectance.set(self.sca_config['Reflectance'])
1469             self.varscageoaccuracy.set(self.sca_config['Geom.Accuracy'])
1470             self.varscatracktwist.set(self.sca_config['Track Twist'])
1471             self.varscacleanliness.set(self.sca_config['Cleanliness'])
1472             self.varscafactor.set(self.sca_config['Factor'])
1473             self.varscavailability.set(self.sca_config['Availability'])
1474             self.updateHCEperSCA()
1475
1476     def buildSCAFrame(self):
1477
1478         self.sca_list = []
1479
1480         self.varscname = tk.StringVar(self.fr_sca)
1481         self.varscalength = tk.DoubleVar(self.fr_sca)
1482         self.varscaaperture = tk.DoubleVar(self.fr_sca)
1483         self.varscafocallen = tk.DoubleVar(self.fr_sca)
1484         self.varscIAMF0 = tk.DoubleVar(self.fr_sca)
1485         self.varscIAMF1 = tk.DoubleVar(self.fr_sca)
1486         self.varscIAMF2 = tk.DoubleVar(self.fr_sca)
1487         self.varscareflectance = tk.DoubleVar(self.fr_sca)
1488         self.varscageoaccuracy = tk.DoubleVar(self.fr_sca)
1489         self.varscatracktwist = tk.DoubleVar(self.fr_sca)
1490         self.varscacleanliness = tk.DoubleVar(self.fr_sca)
1491         self.varscafactor = tk.DoubleVar(self.fr_sca)
1492         self.varscavailability = tk.DoubleVar(self.fr_sca)
1493
1494         self.lbscname = ttk.Label(
1495             self.fr_sca,
1496             text='SCA base name').grid()

```

```
1497             row=0, column=0, sticky='W', padx=2, pady=5)
1498
1499     self.enscaname = ttk.Entry(
1500         self.fr_sca,
1501         textvariable=self.vars cname).grid(
1502             row=0, column=1, sticky='W', padx=2, pady=5)
1503
1504     self.btscaload = ttk.Button(
1505         self.fr_sca, text='Load Library',
1506         command=lambda: self.load_sca_library())
1507     self.btscaload.grid(row=0, column=7, sticky='W', padx=2, pady=5)
1508
1509     self.cmbscaname = ttk.Combobox(self.fr_sca)
1510     self.cmbscaname.bind(
1511         '<<ComboboxSelected>>', lambda event: self.load_sca_parameters())
1512     self.cmbscaname['values'] = self.sca_list
1513     self.cmbscaname.grid(row=0, column=3, columnspan=4, padx=5, pady=5)
1514
1515     self.lbscalength = ttk.Label(
1516         self.fr_sca,
1517         text='SCA Length [m]').grid(
1518             row=1, column=0, sticky='W', padx=2, pady=5)
1519     self.enscalength = ttk.Entry(
1520         self.fr_sca,
1521         textvariable=self.varscalength)
1522     self.enscalength.bind('<Key>', lambda event: self.updateHCEperSCA())
1523     self.enscalength.grid(
1524             row=1, column=1, sticky='W', padx=2, pady=5)
1525
1526     self.lbscaaperture = ttk.Label(
1527         self.fr_sca,
1528         text='SCA aperture [m]').grid(
1529             row=2, column=0, sticky='W', padx=2, pady=5)
1530     self.enscaaperture = ttk.Entry(
1531         self.fr_sca,
1532         textvariable=self.varscaperture).grid(
1533             row=2, column=1, sticky='W', padx=2, pady=5)
1534
1535     self.lbscafocallen = ttk.Label(
1536         self.fr_sca,
1537         text='SCA focal length [m]').grid(
1538             row=3, column=0, sticky='W', padx=2, pady=5)
1539     self.enscafocallen = ttk.Entry(
1540         self.fr_sca,
1541         textvariable=self.varscafocallen).grid(
1542             row=3, column=1, sticky='W', padx=2, pady=5)
1543
1544     self.lbscaIAMF0 = ttk.Label(
1545         self.fr_sca,
1546         text='SCA IAM facotr F0 []').grid(
1547             row=4, column=0, sticky='W', padx=2, pady=5)
1548     self.enscaIAMF0 = ttk.Entry(
1549         self.fr_sca,
1550         textvariable=self.varsc IAMF0).grid(
1551             row=4, column=1, sticky='W', padx=2, pady=5)
1552
1553     self.lbscaIAMF1 = ttk.Label(
1554         self.fr_sca,
1555         text='SCA IAM facotr F1 []').grid(
1556             row=5, column=0, sticky='W', padx=2, pady=5)
```

```
1557     self.enscaIAMF1 = ttk.Entry(
1558         self.fr_sca,
1559         textvariable=self.varscaIAMF1).grid(
1560             row=5, column=1, sticky='W', padx=2, pady=5)
1561
1562     self.lbscaIAMF2 = ttk.Label(
1563         self.fr_sca,
1564         text='SCA IAM facotr F2 []').grid(
1565             row=6, column=0, sticky='W', padx=2, pady=5)
1566     self.enscaIAMF2 = ttk.Entry(
1567         self.fr_sca,
1568         textvariable=self.varscaIAMF2).grid(
1569             row=6, column=1, sticky='W', padx=2, pady=5)
1570
1571     self.lbscareflectance = ttk.Label(
1572         self.fr_sca,
1573         text='SCA mirror reflectance []').grid(
1574             row=7, column=0, sticky='W', padx=2, pady=5)
1575     self.enscareflectance = ttk.Entry(
1576         self.fr_sca,
1577         textvariable=self.varscareflectance).grid(
1578             row=7, column=1, sticky='W', padx=2, pady=5)
1579
1580     self.lbscageoaccuracy = ttk.Label(
1581         self.fr_sca,
1582         text='SCA geometric accuracy []').grid(
1583             row=8, column=0, sticky='W', padx=2, pady=5)
1584     self.enscageoaccuracy = ttk.Entry(
1585         self.fr_sca,
1586         textvariable=self.varscageoaccuracy).grid(
1587             row=8, column=1, sticky='W', padx=2, pady=5)
1588
1589     self.lbscatracketwist = ttk.Label(
1590         self.fr_sca,
1591         text='SCA Tracking Twist []').grid(
1592             row=9, column=0, sticky='W', padx=2, pady=5)
1593     self.enscatracketwist = ttk.Entry(
1594         self.fr_sca,
1595         textvariable=self.varscatracketwist).grid(
1596             row=9, column=1, sticky='W', padx=2, pady=5)
1597
1598     self.lbscacleanliness = ttk.Label(
1599         self.fr_sca,
1600         text='SCA Cleanliness []').grid(
1601             row=10, column=0, sticky='W', padx=2, pady=5)
1602     self.enscacleanliness = ttk.Entry(
1603         self.fr_sca,
1604         textvariable=self.varscacleanliness).grid(
1605             row=10, column=1, sticky='W', padx=2, pady=5)
1606
1607     self.lbscafactor = ttk.Label(
1608         self.fr_sca,
1609         text='SCA Factor []').grid(
1610             row=11, column=0, sticky='W', padx=2, pady=5)
1611     self.enscafactor = ttk.Entry(
1612         self.fr_sca,
1613         textvariable=self.varscafactor).grid(
1614             row=11, column=1, sticky='W', padx=2, pady=5)
1615
1616     self.lbscaavailability = ttk.Label(
```

```

1617     self.fr_sca,
1618     text='SCA Availability []').grid(
1619         row=12, column=0, sticky='W', padx=2, pady=5)
1620 self.ensaavailability = ttk.Entry(
1621     self.fr_sca,
1622     textvariable=self.varscaavailability).grid(
1623         row=12, column=1, sticky='W', padx=2, pady=5)
1624
1625 def load_hce_parameters(self):
1626
1627     self.hce_config = {}
1628
1629     for item in self.hce_list:
1630
1631         if item['Name'] == self.cmbhcename.get():
1632             self.hce_config = item
1633
1634         self.varhcename.set(self.hce_config['Name'])
1635         self.varhcedri.set(self.hce_config['Absorber tube inner diameter'])
1636         self.varhcedro.set(self.hce_config['Absorber tube outer diameter'])
1637         self.varhcedgi.set(self.hce_config['Glass envelope inner diameter'])
1638         self.varhcedgo.set(self.hce_config['Glass envelope outer diameter'])
1639         self.varhcelength.set(self.hce_config['Length'])
1640         self.varhceinnerroughness.set(self.hce_config['Inner surface roughness'])
1641         self.varhceminreynolds.set(self.hce_config['Min Reynolds'])
1642         self.varhceemittanceA0.set(self.hce_config['Absorber emittance factor A0'])
1643         self.varhceemittanceA1.set(self.hce_config['Absorber emittance factor A1'])
1644         self.varhceabsorptance.set(self.hce_config['Absorber absorptance'])
1645         self.varhcetransmittance.set(self.hce_config['Envelope transmittance'])
1646         self.varhcebrackets.set(self.hce_config['Brackets'])
1647         self.updateHCEperSCA()
1648
1649 def load_hce_library(self, path = None):
1650
1651     if path is None:
1652         path = askopenfilename(initialdir = self._DIR['hce_files'],
1653                               title='choose your file',
1654                               filetypes=[('JSON files', '*.json')])
1655
1656     with open(path) as cfg_file:
1657         cfg = json.load(cfg_file, parse_float= float, parse_int= int)
1658
1659     self.hce_list = cfg
1660     self.cmbhcename['values'] = [s['Name'] for s in cfg]
1661     self.cmbhcename.current(0)
1662     self.load_hce_parameters()
1663
1664
1665 def updateHCEperSCA(self):
1666
1667     var1 = self.varscalength.get()
1668     var2 = self.varhcelength.get()
1669
1670     if var2 != 0:
1671         self.varhcepersca.set(var1 // var2)
1672     else:
1673         self.varhcepersca.set(0.0)
1674
1675     var3 = self.varhcepersca.get()
1676     var4 = self.varhces.get()

```

```

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

```

```
1736     text='HCE outer diameter [m]').grid(
1737         row=2, column=0, sticky='W', padx=2, pady=5)
1738 self.enhcedro = ttk.Entry(
1739     self.fr_hce,
1740     textvariable=self.varhcedro).grid(
1741         row=2, column=1, sticky='W', padx=2, pady=5)
1742
1743 self.lbhcedgi = ttk.Label(
1744     self.fr_hce,
1745     text='HCE glass inner diameter [m]').grid(
1746         row=3, column=0, sticky='W', padx=2, pady=5)
1747 self.enhcedgi = ttk.Entry(
1748     self.fr_hce,
1749     textvariable=self.varhcedgi).grid(
1750         row=3, column=1, sticky='W', padx=2, pady=5)
1751
1752 self.lbhcedgo = ttk.Label(
1753     self.fr_hce,
1754     text='HCE glass outer diameter [m]').grid(
1755         row=4, column=0, sticky='W', padx=2, pady=5)
1756 self.enhcedgo = ttk.Entry(
1757     self.fr_hce,
1758     textvariable=self.varhcedgo).grid(
1759         row=4, column=1, sticky='W', padx=2, pady=5)
1760
1761 self.lbhcelong = ttk.Label(
1762     self.fr_hce,
1763     text='HCE longitude [m]').grid(
1764         row=5, column=0, sticky='W', padx=2, pady=5)
1765 self.enhcelong = ttk.Entry(
1766     self.fr_hce,
1767     textvariable=self.varhcelength)
1768 self.enhcelong.bind('<Key>', lambda event: self.updateHCEperSCA())
1769 self.enhcelong.grid(row=5, column=1, sticky='W', padx=2, pady=5)
1770
1771 self.lbhcepersca = ttk.Label(
1772     self.fr_hce,
1773     textvariable=self.varhceperscatext).grid(
1774         row=5, column=2, sticky='W', padx=2, pady=5)
1775
1776 self.lbbellowsratio = ttk.Label(
1777     self.fr_hce,
1778     text='Bellows ratio [ ]').grid(
1779         row=6, column=0, sticky='W', padx=2, pady=5)
1780 self.enbellowsratio = ttk.Entry(
1781     self.fr_hce,
1782     textvariable=self.varbellowsratio).grid(
1783         row=6, column=1, sticky='W', padx=2, pady=5)
1784
1785 self.lbshieldshading = ttk.Label(
1786     self.fr_hce,
1787     text='Shield shading [ ]').grid(
1788         row=7, column=0, sticky='W', padx=2, pady=5)
1789 self.enshieldshading = ttk.Entry(
1790     self.fr_hce,
1791     textvariable=self.varshieldshading).grid(
1792         row=7, column=1, sticky='W', padx=2, pady=5)
1793
1794 self.lbbrackets = ttk.Label(
1795     self.fr_hce,
```

```
1796     text='Brackets spacing').grid(
1797         row=8, column=0, sticky='W', padx=2, pady=5)
1798 self.enbrackets = ttk.Entry(
1799     self.fr_hce,
1800     textvariable=self.varhcebrackets).grid(
1801         row=8, column=1, sticky='W', padx=2, pady=5)
1802
1803 self.lbhceinnerroughness = ttk.Label(
1804     self.fr_hce,
1805     text='HCE inner roughness [ ]').grid(
1806         row=10, column=0, sticky='W', padx=2, pady=5)
1807 self.enhceinnerroughness = ttk.Entry(
1808     self.fr_hce,
1809     textvariable=self.varhceinnerroughness).grid(
1810         row=10, column=1, sticky='W', padx=2, pady=5)
1811
1812 self.lbhceminreynolds = ttk.Label(
1813     self.fr_hce,
1814     text='HCE minimum Reynolds Number [ ]').grid(
1815         row=11, column=0, sticky='W', padx=2, pady=5)
1816 self.enhceminreynolds = ttk.Entry(
1817     self.fr_hce,
1818     textvariable=self.varhceminreynolds).grid(
1819         row=11, column=1, sticky='W', padx=2, pady=5)
1820
1821 self.lbhceabsorptance = ttk.Label(
1822     self.fr_hce,
1823     text='Absorptance [ ]').grid(
1824         row=12, column=0, sticky='W', padx=2, pady=5)
1825 self.enhceabsorptance = ttk.Entry(
1826     self.fr_hce,
1827     textvariable=self.varhceabsorptance).grid(
1828         row=12, column=1, sticky='W', padx=2, pady=5)
1829
1830 self.lbhcetransmittance = ttk.Label(
1831     self.fr_hce,
1832     text='Transmittance [ ]').grid(
1833         row=13, column=0, sticky='W', padx=2, pady=5)
1834 self.enhcetransmittance = ttk.Entry(
1835     self.fr_hce,
1836     textvariable=self.varhcetransmittance).grid(
1837         row=13, column=1, sticky='W', padx=2, pady=5)
1838
1839 self.lbemittanceA0 = ttk.Label(
1840     self.fr_hce,
1841     text='Emittance Factor A0 [ ]').grid(
1842         row=14, column=0, sticky='W', padx=2, pady=5)
1843 self.enemittanceA0 = ttk.Entry(
1844     self.fr_hce,
1845     textvariable=self.varhceemittanceA0).grid(
1846         row=14, column=1, sticky='W', padx=2, pady=5)
1847
1848 self.lbemittanceA1 = ttk.Label(
1849     self.fr_hce,
1850     text='Emittance Factor A1 [ ]').grid(
1851         row=15, column=0, sticky='W', padx=2, pady=5)
1852 self.enemittanceA1 = ttk.Entry(
1853     self.fr_hce,
1854     textvariable=self.varhceemittanceA1).grid(
1855         row=15, column=1, sticky='W', padx=2, pady=5)
```

```
1856
1857
1858 if __name__ == '__main__':
1859
1860     try:
1861         from Tkinter import Tk
1862         import tkMessageBox as messagebox
1863
1864     except ImportError:
1865         from tkinter import Tk
1866         from tkinter import messagebox
1867
1868     interface = Interface()
1869     interface.root.mainloop()
```