```python
# -*- coding: utf-8 -*-

"""
interface.py: A Tkinter application for creating configuration files to
run simulation with csenergy.py
@author: pacomunuera
2020
"""


import sys
sys.path.append('./libs')
import os.path
import CoolProp.CoolProp as CP
import pvlib as pvlib
import tkinter as tk
from tkinter.filedialog import askopenfilename, asksaveasfile
import tkinter.ttk as ttk
from tkinter import messagebox
# recipe-580746-1.py from
# http://code.activestate.com/recipes/
# 580746-t kinter-treeview-like-a-table-or-multicolumn-listb/
import recipe5807461 as table
import json
from decimal import Decimal
from datetime import datetime
import pandas as pd


class Interface(object):

    _COOLPROP_FLUIDS = ['Water', 'INCOMP::TVP1', 'INCOMP::S800']

    _MODELS = ['Barbero4thOrder', 'Barbero1stOrder', 'BarberoSimplified']

    _DIR = {
        'saved_configurations': './saved_configurations/',
        'site_files': './site_files/',
        'fluid_files': './fluid_files',
        'hce_files': './hce_files',
        'sca_files': './sca_files',
        'fielddata_files': './fielddata_files',
        'weather_files': './weather_files'}

    cfg_settings = {
        'simulation': {},
        'solar_plant': {},
        'site': {},
        'weather': {},
        'hce': {},
        'SCA': {},
        'hot_hce': {},
        'cold_hce': {},
        'cycle': {},
        'hce_model_settings': {},
        'hce_scattered_params': {},
        'sca_scattered_params': {}}

    def __init__(self):

```

```
        # Main window
        self.root = tk.Tk()
        self.root.attributes('-fullscreen', False)
        self.varrroottitle = tk.StringVar(self.root)
        self.root.title('Solar Field Configurator ')
        w, h = self.root.winfo_screenwidth(), self.root.winfo_screenheight()
        self.root.geometry('%dx%d+0+0' % (w-200, h-200))
        self.root.geometry('%dx%d+0+0' % (800, 800))

        # Menu
        self.menubar = tk.Menu(self.root)

        self.simulation_menu = tk.Menu(self.menubar, tearoff=0)
        self.simulation_menu.add_command(
            label='New', command=self.simulation_new)
        self.simulation_menu.add_command(
            label='Open', command=self.simulation_open)
        self.simulation_menu.add_command(
            label='Save', command=self.simulation_save)
        self.simulation_menu.add_command(
            label='Save as...', command=self.simulation_save_as)
        self.simulation_menu.add_separator()
        self.simulation_menu.add_command(
            label='Exit', command=self.simulation_exit)
        self.menubar.add_cascade(
            label='Simulation', menu=self.simulation_menu)
        self.run_menu = tk.Menu(self.menubar, tearoff=0)
        self.run_menu.add_command(label='Run', command=self.run_simulation)
        self.menubar.add_cascade(label='Run', menu=self.run_menu)
        self.help_menu = tk.Menu(self.menubar, tearoff=0)
        self.help_menu.add_command(label='Help', command=self.help_help)
        self.help_menu.add_command(label='About', command=self.help_about)
        self.menubar.add_cascade(label='Help', menu=self.help_menu)
        self.root.config(menu=self.menubar)

        # Notebook (tabs)
        self.nb = ttk.Notebook(self.root)

        self.fr_simulation = tk.Frame(self.nb)
        self.fr_solarfield = tk.Frame(self.nb)
        self.fr_fluid = tk.Frame(self.nb)
        self.fr_hce = tk.Frame(self.nb)
        self.fr_sca = tk.Frame(self.nb)

        self.buildNotebook()
        self.buildSolarFieldFrame()
        self.buildSimulationFrame()
        self.buildFluidFrame()
        self.buildHCEFrame()
        self.buildSCAFrame()

        self.simulation_open(self._DIR['saved_configurations']+'template.json')

    def simulation_new(self):

        #  Simulation configuration
        self.varsimID.set('')
        self.varsimdatatype.set(1)
        self.varsimulation.set(False)
        self.varbenchmark.set(False)
```

```
        self.varfastmode.set(False)
        self.vardatafilename.set('')
        self.vardatafilepath.set('')
        self.vardatafileurl.set('')
        self.varfirstdate.set(pd.to_datetime('2000/01/01 1:00').strftime(
                '%Y/%m/%d %H:%M'))
        self.varlastdate.set(pd.to_datetime('2000/12/31 1:00').strftime(
                '%Y/%m/%d %H:%M'))
        self.cmbmodelname.set('')
        self.varmodelmaxerrt.set(1.0)
        self.varmodelmaxerrtro.set(0.1)
        self.varmodelmaxerrpr.set(0.01)

        self.varsitename.set(0)
        self.varsitelat.set(0)
        self.varsitelong.set(0)
        self.varsitealt.set(0)
        self.checkoptions()
        self.checkfastmode()

        #  Solarfield configuration
        self.solarfield_table.table_data = []
        self.columns_table.table_data = []

        self.vartin.set(0)
        self.vartout.set(0)
        self.varpin.set(0)
        self.varpout.set(0)
        self.vartmin.set(0)
        self.vartmax.set(0)
        self.varratedmassflow.set(0)
        self.varrecirculation.set(0)
        self.varscas.set(0)
        self.varhces.set(0)
        self.varrowspacing.set(0)
        self.varscatrackingtype.set(1)
        self.varfluidtable.set(1)
        self.varfluidname.set('')
        self.varfluidtmax.set(0)
        self.varfluidtmin.set(0)
        self.fluid_table.table_data = []
        self.cmbcoolpropID.set('')
        self.checkfluid()

        #  SCA configuration
        self.varscaname.set('')
        self.varscalength.set(0)
        self.varscaaperture.set(0)
        self.varscafocallen.set(0)
        self.varscaIAMF0.set(0)
        self.varscaIAMF1.set(0)
        self.varscaIAMF2.set(0)
        self.varscareflectance.set(0)
        self.varscageoaccuracy.set(0)
        self.varscatracktwist.set(0)
        self.varscacleanliness.set(0)
        self.varscafactor.set(0)
        self.varscaavailability.set(0)

        #  HCE configuration
```

```
        self.varhcename.set('')
        self.varhcedri.set(0)
        self.varhcedro.set(0)
        self.varhcedgi.set(0)
        self.varhcedgo.set(0)
        self.varhcelength.set(0)
        self.varhceemittanceA0.set(0)
        self.varhceemittanceA1.set(0)
        self.varhceabsorptance.set(0)
        self.varhcetransmittance.set(0)
        self.varhceinnerroughness.set(0)
        self.varhceminreynolds.set(0)
        self.varhcebrackets.set(0)
        self.updateHCEperSCA()

        self.fr_fluid.update()
        self.fr_hce.update()
        self.fr_solarfield.update()
        self.fr_sca.update()
        self.fr_simulation.update()

    def simulation_open(self, path=None):

        if path == None:
            path = askopenfilename(initialdir=self._DIR['saved_configurations'],
                                   title='choose your file',
                                   filetypes=[('JSON files', '*.json')])

        head, tail = os.path.split(path)
        self.filename = path
        self.root.title('Solar Field Configurator: ' + tail)
        with open(path) as cfg_file:
            cfg = json.load(cfg_file,
                            parse_float= float,
                            parse_int= int)

        #  Simulation configuration
        self.varsimID.set(cfg['simulation']['ID'])
        self.varsimdatatype.set(cfg['simulation']['datatype'])
        self.varsimulation.set(cfg['simulation']['simulation'])
        self.varbenchmark.set(cfg['simulation']['benchmark'])
        self.varfastmode.set(cfg['simulation']['fastmode'])
        self.vardatafilename.set(cfg['simulation']['filename'])
        self.vardatafilepath.set(cfg['simulation']['filepath'])
        self.vardatafileurl.set(cfg['simulation']['filepath'] +
                                cfg['simulation']['filename'])

        self.varfirstdate.set(pd.to_datetime(
            cfg['simulation']['first_date']))
        self.varlastdate.set(pd.to_datetime(
            cfg['simulation']['last_date']))

        self.cmbmodelname.set(cfg['model']['name'])
        self.varmodelmaxerrt.set(cfg['model']['max_err_t'])
        self.varmodelmaxerrtro.set(cfg['model']['max_err_tro'])
        self.varmodelmaxerrpr.set(cfg['model']['max_err_pr'])

        self.varsitename.set(cfg['site']['name'])
        self.varsitelat.set(cfg['site']['latitude'])
        self.varsitelong.set(cfg['site']['longitude'])
```

```
        self.varsitealt.set(cfg['site']['altitude'])
        self.checkoptions()
        self.checkfastmode()

        #  Solarfield configuration
        list_subfields = []
        for r in cfg['subfields']:
            list_subfields.append(list(r.values()))

        self.solarfield_table.table_data = list_subfields

        if 'tags' in cfg.keys():
            list_tags = []
            for r in cfg['tags']:
                list_tags.append([r, '', cfg['tags'][r]])
            self.columns_table.table_data = list_tags

        self.vartin.set(cfg['loop']['rated_tin'])
        self.vartout.set(cfg['loop']['rated_tout'])
        self.varpin.set(cfg['loop']['rated_pin'])
        self.varpout.set(cfg['loop']['rated_pout'])
        self.vartmin.set(cfg['loop']['tmin'])
        self.vartmax.set(cfg['loop']['tmax'])
        self.varratedmassflow.set(cfg['loop']['rated_massflow'])
        self.varrecirculation.set(cfg['loop']['min_massflow'])
        self.varscas.set(cfg['loop']['scas'])
        self.varhces.set(cfg['loop']['hces'])
        self.varrowspacing.set(cfg['loop']['row_spacing'])
        self.varscatrackingtype.set(cfg['loop']['Tracking Type'])

        #  HTF configuration
        fluid_table = []

        if cfg['HTF']['source'] == 'table':
            self.varfluidtable.set(1)
            self.varfluidname.set(cfg['HTF']['name'])
            self.varfluidtmax.set(cfg['HTF']['tmax'])
            self.varfluidtmin.set(cfg['HTF']['tmin'])
            fluid_table.append(['mu'] + cfg['HTF']['mu'])
            fluid_table.append(['cp'] + cfg['HTF']['cp'])
            fluid_table.append(['rho'] + cfg['HTF']['rho'])
            fluid_table.append(['kt'] + cfg['HTF']['kt'])
            fluid_table.append(['h'] + cfg['HTF']['h'])
            fluid_table.append(['t'] + cfg['HTF']['t'])
            self.fluid_table.table_data = fluid_table

        elif cfg['HTF']['source'] == 'CoolProp':
            self.varfluidtable.set(2)
            self.cmbcoolpropID.set(cfg['HTF']['CoolPropID'])
            self.varfluidtmax.set(cfg['HTF']['tmax'])
            self.varfluidtmin.set(cfg['HTF']['tmin'])

        self.checkfluid()

        #  SCA configuration
        self.varscaname.set(cfg['SCA']['Name'])
        self.varscalength.set(cfg['SCA']['SCA Length'])
        self.varscaaperture.set(cfg['SCA']['Aperture'])
        self.varscafocallen.set(cfg['SCA']['Focal Len'])
        self.varscaIAMF0.set(cfg['SCA']['IAM Coefficient F0'])
```

```
        self.varscaIAMF1.set(cfg['SCA']['IAM Coefficient F1'])
        self.varscaIAMF2.set(cfg['SCA']['IAM Coefficient F2'])
        self.varscareflectance.set(cfg['SCA']['Reflectance'])
        self.varscageoaccuracy.set(cfg['SCA']['Geom.Accuracy'])
        self.varscatracktwist.set(cfg['SCA']['Track Twist'])
        self.varscacleanliness.set(cfg['SCA']['Cleanliness'])
        self.varscafactor.set(cfg['SCA']['Factor'])
        self.varscaavailability.set(cfg['SCA']['Availability'])

        #  HCE configuration
        self.varhcename.set(cfg['HCE']['Name'])
        self.varhcedri.set(cfg['HCE']['Absorber tube inner diameter'])
        self.varhcedro.set(cfg['HCE']['Absorber tube outer diameter'])
        self.varhcedgi.set(cfg['HCE']['Glass envelope inner diameter'])
        self.varhcedgo.set(cfg['HCE']['Glass envelope outer diameter'])
        self.varhcelength.set(cfg['HCE']['Length'])
        self.varbellowsratio.set(cfg['HCE']['Bellows ratio'])
        self.varshieldshading.set(cfg['HCE']['Shield shading'])
        self.varhceemittanceA0.set(cfg['HCE']['Absorber emittance factor A0'])
        self.varhceemittanceA1.set(cfg['HCE']['Absorber emittance factor A1'])
        self.varhceabsorptance.set(cfg['HCE']['Absorber absorptance'])
        self.varhcetransmittance.set(cfg['HCE']['Envelope transmittance'])
        self.varhceinnerroughness.set(cfg['HCE']['Inner surface roughness'])
        self.varhceminreynolds.set(cfg['HCE']['Min Reynolds'])
        self.varhcebrackets.set(cfg['HCE']['Brackets'])
        self.updateHCEperSCA()

        self.fr_fluid.update()
        self.fr_hce.update()
        self.fr_solarfield.update()
        self.fr_sca.update()
        self.fr_simulation.update()

    def simulation_save(self):

        self.save_as_JSON(self.generate_json(), self.filename)


    def simulation_save_as(self):

        self.save_as_JSON(self.generate_json())

    def generate_json(self):

        self.tagslist = []

        cfg = dict({'simulation': {},
                    'model': {},
                    'site': {},
                    'loop': {},
                    'SCA': {},
                    'HCE': {},
                    'HTF': {},
                    })

        cfg['simulation']['ID'] = self.varsimID.get()
        cfg['simulation']['datatype'] = self.varsimdatatype.get()
        cfg['simulation']['simulation'] = self.varsimulation.get()
        cfg['simulation']['benchmark'] = self.varbenchmark.get()
        cfg['simulation']['fastmode'] = self.varfastmode.get()
```

```
        cfg['simulation']['filename'] = self.vardatafilename.get()
        cfg['simulation']['filepath'] = self.vardatafilepath.get()
        cfg['simulation']['first_date'] = pd.to_datetime(
            self.varfirstdate.get()).strftime(
                '%Y/%m/%d %H:%M%z')
        cfg['simulation']['last_date'] = pd.to_datetime(
            self.varlastdate.get()).strftime(
                '%Y/%m/%d %H:%M%z')

        cfg['model']['name'] =  self.cmbmodelname.get()
        cfg['model']['max_err_t'] = self.varmodelmaxerrt.get()
        cfg['model']['max_err_tro'] = self.varmodelmaxerrtro.get()
        cfg['model']['max_err_pr'] = self.varmodelmaxerrpr.get()

        #  Site configuration
        cfg['site']['name'] = self.varsitename.get()
        cfg['site']['latitude'] = self.varsitelat.get()
        cfg['site']['longitude'] =  self.varsitelong.get()
        cfg['site']['altitude'] = self.varsitealt.get()

        #  HTF configuration
        if self.varfluidtable.get() == 1:
            cfg['HTF']['source'] = 'table'
            cfg['HTF']['name'] = self.varfluidname.get()

            parameters_table = list(self.fluid_table.table_data)

            for r in parameters_table:
                param_name = r[0]
                param_values = r[1:]
                param_values = list(map(self.to_number, r[1:]))
                cfg['HTF'].update(dict({param_name : param_values}))

            cfg['HTF'].update({'tmax' : float(self.entmax.get())})
            cfg['HTF'].update({'tmin' : float(self.entmin.get())})

        elif self.varfluidtable.get() == 2:
            cfg['HTF']['source'] = 'CoolProp'
            cfg['HTF']['CoolPropID'] = self.cmbcoolpropID.get()
            cfg['HTF']['tmax'] = float(self.varcoolproptmax.get())
            cfg['HTF']['tmin'] = float(self.varcoolproptmin.get())


        #  SCA configuration
        cfg['SCA']['Name'] = self.varscaname.get()
        cfg['SCA']['SCA Length'] = self.varscalength.get()
        cfg['SCA']['Aperture'] = self.varscaaperture.get()
        cfg['SCA']['Focal Len'] = self.varscafocallen.get()
        cfg['SCA']['IAM Coefficient F0'] = self.varscaIAMF0.get()
        cfg['SCA']['IAM Coefficient F1'] = self.varscaIAMF1.get()
        cfg['SCA']['IAM Coefficient F2'] = self.varscaIAMF2.get()
        cfg['SCA']['Track Twist'] = self.varscatracktwist.get()
        cfg['SCA']['Geom.Accuracy'] = self.varscageoaccuracy.get()
        cfg['SCA']['Reflectance'] = self.varscareflectance.get()
        cfg['SCA']['Cleanliness'] = self.varscacleanliness.get()
        cfg['SCA']['Factor'] = self.varscafactor.get()
        cfg['SCA']['Availability'] = self.varscaavailability.get()

        # HCE configuration
        cfg['HCE']['Name'] = self.varhcename.get()
```

```
        cfg['HCE']['Length'] = self.varhcelength.get()
        cfg['HCE']['Bellows ratio'] = self.varbellowsratio.get()
        cfg['HCE']['Shield shading'] = self.varshieldshading.get()
        cfg['HCE']['Absorber tube inner diameter'] = self.varhcedri.get()
        cfg['HCE']['Absorber tube outer diameter'] = self.varhcedro.get()
        cfg['HCE']['Glass envelope inner diameter'] = self.varhcedgi.get()
        cfg['HCE']['Glass envelope outer diameter'] = self.varhcedgo.get()
        cfg['HCE']['Min Reynolds'] = self.varhceminreynolds.get()
        cfg['HCE']['Inner surface roughness'] = self.varhceinnerroughness.get()
        cfg['HCE']['Envelope transmittance'] = self.varhcetransmittance.get()
        cfg['HCE']['Absorber emittance factor A0'] = self.varhceemittanceA0.get()
        cfg['HCE']['Absorber emittance factor A1'] = self.varhceemittanceA1.get()
        cfg['HCE']['Absorber absorptance'] = self.varhceabsorptance.get()
        cfg['HCE']['Brackets'] = self.varhcebrackets.get()


        #  loop Configuration
        cfg['loop']['scas'] = self.varscas.get()
        cfg['loop']['hces'] = self.varhces.get()
        cfg['loop']['rated_tin'] = self.vartin.get()
        cfg['loop']['rated_tout'] = self.vartout.get()
        cfg['loop']['rated_pin'] = self.varpin.get()
        cfg['loop']['rated_pout'] = self.varpout.get()
        cfg['loop']['tmin'] = self.vartmin.get()
        cfg['loop']['tmax'] = self.vartmax.get()
        cfg['loop']['rated_massflow'] = self.varratedmassflow.get()
        cfg['loop']['min_massflow'] = self.varrecirculation.get()
        cfg['loop']['row_spacing'] = self.varrowspacing.get()
        cfg['loop']['Tracking Type'] = self.varscatrackingtype.get()


        datarow=list(self.solarfield_table.table_data)
        dictkeys =['name', 'loops']

        subfields = []

        for r in datarow:
            sf = {}
            index = 0
            for v in r:
                k = dictkeys[index]
                sf[k]= self.to_number(v)
                index += 1
            subfields.append(sf)

        cfg.update({'subfields': subfields})

        if self.varsimdatatype.get() == 2:

            cfg['tags'] = dict({})

            for r in self.columns_table.table_data:
                cfg['tags'][r[0]] = r[2]

        return cfg

    def save_as_JSON(self, cfg, filename=None):

        if filename is None:
            f = asksaveasfile(initialdir=self._DIR['saved_configurations'],
```

```
                                title='choose your file name',
                                filetypes=[('JSON files', '*.json')],
                                defaultextension='json')
        else:
            f = open(filename,'w')

        if f is not None:
            f.write(json.dumps(cfg,
                               indent= True,
                               ensure_ascii=False))
            f.close()
        else:
            pass


    def __insert_rows__(self, table):

        lst = []
        for c in table._multicolumn_listbox._columns:
            lst.append('')

        rows = table.number_of_rows
        table.insert_row(lst, index = rows)

    def __del_rows__(self, table):
        table.delete_all_selected_rows()

    def run_simulation(self):
        # import subprocess
        import threading
        try:
            cfg = self.generate_json()

            SIM = cs.SolarFieldSimulation(cfg)
            FLAG_00 = datetime.now()
            hilo =  threading.Thread(target=SIM.runSimulation())
            hilo.start()
            FLAG_01 = datetime.now()
            DELTA_01 = FLAG_01 - FLAG_00

        except Exception as e:
            print("serialization failed", e)

    def help_help():
        pass

    def help_about():
        pass

    def simulation_exit(self):

        self.root.destroy()

    def to_number(self, s):

        try:
          i = int(s)
          return(i)
        except ValueError:
            pass
```

```
        try:
            f = float(s)
            return(f)
        except ValueError:
            pass
        return s


    def buildNotebook(self):

        self.nb.add(self.fr_simulation, text='Simulation Configuration', padding=2)
        self.nb.add(self.fr_solarfield, text='  Solar Field Layout ', padding=2)
        self.nb.add(self.fr_fluid, text='         HTF         ', padding=2)
        self.nb.add(self.fr_sca, text='SCA: Solar Collector Assembly', padding=2)
        self.nb.add(self.fr_hce, text='HCE: Heat Collector Element', padding=2)
        self.nb.select(self.fr_simulation)
        self.nb.enable_traversal()
        self.nb.pack()

    #  Simulaton Configuration tab
    def simulationLoadDialog(self, title, labeltext=''):

        path = askopenfilename(initialdir = self._DIR['saved_configurations'],
                               title='choose your file',
                               filetypes=[('JSON files', '*.json')])

        with open(path) as cfg_file:
            cfg = json.load(cfg_file,
                                    parse_float= float,
                                    parse_int= int)

        self.varsimID.set(cfg['simulation']['ID'])
        self.varsimdatatype.set(cfg['simulation']['datatype'])
        self.varsimulation.set(cfg['simulation']['simulation'])
        self.varbenchmark.set(cfg['simulation']['benchmark'])
        self.varfastmode.set(cfg['simulation']['fastmode'])

    def checkoptions(self):

        var = self.varsimdatatype.get()

        if var == 1: #  Weather File
            self.varbenchmark.set(False)
            self.cbbenchmark['state'] ='disabled'
            self.cbloadsitedata['state'] = 'normal'
            self.bttagswizard['state']='disabled'
            self.btloadtags['state']='disabled'
            # self.tags_table.state('disabled')
        elif var == 2: #  Field Data File
            self.varloadsitedata.set(False)
            self.cbbenchmark['state']='normal'
            self.cbloadsitedata['state'] = 'disabled'
            self.bttagswizard['state']='normal'
            self.btloadtags['state']='normal'
            # self.tags_table.state(('active'))
        else:
            pass

    def checkfastmode(self):

```

```
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.varmodelmaxerrt = 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.varsitelong = 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.lbmodelmaxerrt = 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.enmodelmaxerrt = ttk.Entry(
677               self.frame2, textvariable=self.varmodelmaxerrt).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.rbweather = 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.rbfielddata = 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...')
```

```
        self.lbdatasourcepath = ttk.Label(
            self.fr_simulation, textvariable=self.vardatafileurl).grid(
                row=4, column= 1, columnspan=4, sticky='W', padx=2, pady=5)

        self.lbfirstdate = ttk.Label(
            self.fr_simulation, text='First Date').grid(
                row=5, column=0,sticky='W', padx=2, pady=5)
        self.enfirstdate = ttk.Entry(
              self.fr_simulation, textvariable=self.varfirstdate).grid(
                  row=5, column=1, sticky='W', padx=2, pady=5)

        self.lblastdate = ttk.Label(
            self.fr_simulation, text='Last Date').grid(
                row=5, column=2,sticky='E', padx=2, pady=5)
        self.enlastdate = ttk.Entry(
              self.fr_simulation, textvariable=self.varlastdate).grid(
                  row=5, column=3, sticky='W', padx=2, pady=5)

        #  Checkbox for Simulation
        self.lbsimulation = ttk.Label(
            self.fr_simulation, text='Run test type...').grid(
                row=6, column=0, sticky='W', padx=2, pady=5)
        self.cbsimulation = ttk.Checkbutton(
            self.fr_simulation,
            text='Simulation',
            variable=self.varsimulation)
        self.cbsimulation.grid(
            row=6, column=1, sticky='W', padx=2, pady=5)

        self.cbbenchmark = ttk.Checkbutton(
            self.fr_simulation,
            text='Benchmark',
            variable=self.varbenchmark)
        self.cbbenchmark.grid(
            row=6, column=2, sticky='W', padx=2, pady=5)

        self.lbfastmode = ttk.Label(
              self.fr_simulation, text='Fast mode').grid(
                  row=7, column=0, sticky='W', padx=2, pady=5)
        self.cbfastmode = ttk.Checkbutton(
            self.fr_simulation,
            textvariable=self.varfastmodetext,
            variable= self.varfastmode,
            command=lambda: self.checkfastmode()).grid(
                row=7, column=1, sticky='W', padx=2, pady=5)

        self.lbsitename = ttk.Label(
              self.fr_simulation, text='Site').grid(
                  row=8, column=0, sticky='W', padx=2, pady=5)
        self.ensitename = ttk.Entry(
              self.fr_simulation, textvariable=self.varsitename).grid(
                  row=8, column=1, sticky='W', padx=2, pady=5)

        self.cbloadsitedata = ttk.Checkbutton(
            self.fr_simulation,
            text='Load site data from weather file',
            variable=self.varloadsitedata)
        self.cbloadsitedata.grid(
            row=8, column=2, sticky='W', padx=2, pady=5)

```

```
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_masslflow' :
    self.enratedmassflow.get()}))
820         cfg['solarfield'].update(dict({'min_massflow' :
    self.enrecirculationmassflow.get()}))
821         cfg['solarfield'].update(dict({'tmin' : self.entmin.get()}))
822         cfg['solarfield'].update(dict({'tmin' : self.entmax.get()}))
823         cfg['solarfield'].update(dict({'loop': dict({'scas': self.enscas.get(),
824                                                      'hces': self.enhces.get()})}))
825 
826         datarow=list(self.solarfield_table.table_data)
827         dictkeys =['name', 'loops']
828 
829         subfields = []
830 
831         for r in datarow:
832             sf = {}
833             index = 0
834             for v in r:
835                 k = dictkeys[index]
836                 sf[k]= self.to_number(v)
837                 index += 1
838             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]]
```

```
                self.columns_table.update_row(
                    index,new_row)
            index += 1

    def buildSolarFieldFrame(self):

        self.varsolarfieldname = tk.StringVar(self.fr_solarfield)
        self.lbname = ttk.Label(self.fr_solarfield, text ='Solar Field Name' )
        self.lbname.grid(row=0, column=0, sticky='W', padx=5, pady=5)
        self.enname = ttk.Entry(self.fr_solarfield,
textvariable=self.varsolarfieldname)
        self.enname.grid(row=0, column=1, sticky='W', padx=5, pady=5)

        self.vartin = tk.DoubleVar(self.fr_solarfield)
        self.lbratedtin = ttk.Label(self.fr_solarfield, text='Rated Tin [K]')
        self.lbratedtin.grid(row=1, column=0, sticky='W', padx=5, pady=5)
        self.enratedtin = ttk.Entry(self.fr_solarfield, textvariable=self.vartin)
        self.enratedtin.grid(row=1, column=1, sticky='W', padx=5, pady=5)

        self.vartout = tk.DoubleVar(self.fr_solarfield)
        self.lbratedtout = ttk.Label(self.fr_solarfield, text='Rated Tout [K]')
        self.lbratedtout.grid(row=1, column=2, sticky='W', padx=5, pady=5)
        self.enratedtout = ttk.Entry(self.fr_solarfield, textvariable=self.vartout)
        self.enratedtout.grid(row=1, column=3, sticky='W', padx=5, pady=5)

        self.varpin = tk.DoubleVar(self.fr_solarfield)
        self.lbratedpin = ttk.Label(self.fr_solarfield, text='Rated Pin [Pa]')
        self.lbratedpin.grid(row=2, column=0, sticky='W', padx=5, pady=5)
        self.enratedpin = ttk.Entry(self.fr_solarfield, textvariable=self.varpin)
        self.enratedpin.grid(row=2, column=1, sticky='W', padx=5, pady=5)

        self.varpout = tk.DoubleVar(self.fr_solarfield)
        self.lbratedpout = ttk.Label(self.fr_solarfield, text='Rated Pout [Pa]')
        self.lbratedpout.grid(row=2, column=2, sticky='W', padx=5, pady=5)
        self.enratedpout = ttk.Entry(self.fr_solarfield, textvariable=self.varpout)
        self.enratedpout.grid(row=2, column=3, sticky='W', padx=5, pady=5)

        self.vartmin = tk.DoubleVar(self.fr_solarfield)
        self.lbtmin = ttk.Label(self.fr_solarfield, text='Tmin [K]')
        self.lbtmin.grid(row=3, column=0, sticky='W', padx=5, pady=5)
        self.entmin = ttk.Entry(self.fr_solarfield, textvariable=self.vartmin)
        self.entmin.grid(row=3, column=1, sticky='W', padx=5, pady=5)

        self.vartmax = tk.DoubleVar(self.fr_solarfield)
        self.lbtmax = ttk.Label(self.fr_solarfield, text='Tmax [K]')
        self.lbtmax.grid(row=3, column=2, sticky='W', padx=5, pady=5)
        self.entmax = ttk.Entry(self.fr_solarfield, textvariable=self.vartmax)
        self.entmax.grid(row=3, column=3, sticky='W', padx=5, pady=5)

        self.varratedmassflow = tk.DoubleVar(self.fr_solarfield)
        self.lbratedmassflow = ttk.Label(
            self.fr_solarfield, text='Loop Rated massflow [Kg/s]')
        self.lbratedmassflow.grid(row=4, column=0, sticky='W', padx=5, pady=5)
        self.enratedmassflow = ttk.Entry(
            self.fr_solarfield, textvariable=self.varratedmassflow)
        self.enratedmassflow.grid(row=4, column=1, sticky='W', padx=5, pady=5)

        self.varrecirculation = tk.DoubleVar(self.fr_solarfield)
        self.lbrecirculationmassflow = ttk.Label(
            self.fr_solarfield, text='Loop minimum massflow [Kg/s]')
```

```
        self.lbrecirculationmassflow.grid(
            row=4, column=2, sticky='W', padx=5, pady=5)
        self.enrecirculationmassflow = ttk.Entry(self.fr_solarfield,
textvariable=self.varrecirculation)
        self.enrecirculationmassflow.grid(
            row=4, column=3, sticky='W', padx=5, pady=5)

        self.varrowspacing = tk.DoubleVar(self.fr_solarfield)
        self.lbrowspacing = ttk.Label(
            self.fr_solarfield, text='Row Spacing [m]')
        self.lbrowspacing.grid(row=5, column=0, sticky='W', padx=5, pady=5)
        self.enrowspacing = ttk.Entry(
            self.fr_solarfield, textvariable=self.varrowspacing)
        self.enrowspacing.grid(row=5, column=1, sticky='W', padx=5, pady=5)

        self.varscatrackingtype = tk.IntVar(self.fr_solarfield)

        self.lbscatrackingtype = ttk.Label(
            self.fr_solarfield,
            text='Tracking Axis (1: N-S, 2: E-W)').grid(
                row=5, column=2, sticky='W', padx=2, pady=5)

        #  RadioButton for Tracking Type (Tracking Axis 1: N-S, 2: E-W)
        self.rbtrackingtype = tk.Radiobutton(
            self.fr_solarfield,
            padx=2,
            text = 'Tracking Axis N-S',
            variable= self.varscatrackingtype, value=1).grid(
                row=5, column=3, sticky='W', padx=2, pady=5)

        #  RadioButton for Tracking Type (Tracking Axis 1: N-S, 2: E-W)
        self.rbtrackingtype  = tk.Radiobutton(
            self.fr_solarfield,
            padx=2,
            text = 'Tracking Axis E-W',
            variable= self.varscatrackingtype, value=2).grid(
                row=5, column=4, sticky='W', padx=2, pady=5)

        self.varscas = tk.IntVar(self.fr_solarfield)
        self.lbscas = ttk.Label(self.fr_solarfield, text='SCAs per Loop')
        self.lbscas.grid(row=6, column=0, sticky='W', padx=5, pady=5)
        self.enscas = ttk.Entry(self.fr_solarfield, textvariable=self.varscas)
        self.enscas.grid(row=6, column=1, sticky='W', padx=5, pady=5)

        self.varhces = tk.IntVar(self.fr_solarfield)
        self.lbhces = ttk.Label(self.fr_solarfield, text='HCEs per SCA')
        self.lbhces.grid(row=6, column=2, sticky='W', padx=5, pady=5)
        self.enhces = ttk.Entry(self.fr_solarfield, textvariable=self.varhces)
        self.enhces.bind('<Key>', lambda event: self.updateHCEperSCA())
        self.enhces.grid(row=6, column=3, sticky='W', padx=5, pady=5)

        self.solarfield_table = table.Tk_Table(
            self.fr_solarfield,
            ['SUBFIELD NAME', 'NUMBER OF LOOPS'],
            row_numbers=True,
            stripped_rows=('white', '#f2f2f2'),
            select_mode='none',
            cell_anchor='center',
            adjust_heading_to_content=True)
        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:
```

```
            path = askopenfilename(
                initialdir=self._DIR['fielddata_files'],
                title='choose your file',
                filetypes=(('csv files', '*.csv'),
                           ('all files', '*.*')))

            self.vardatafilepath.set(os.path.dirname(path)+'/')
            self.vardatafilename.set(os.path.basename(path))
            self.vardatafileurl.set(os.path.dirname(path)+'/' +
                                    os.path.basename(path))

        else:
            tk.messagebox.showwarning(
                title='Warning',
                message='Check Data Source Selection')

        if self.varloadsitedata.get():

            strfilename, strext = os.path.splitext(path)
            if  strext == '.csv':
                weatherdata = pvlib.iotools.tmy.read_tmy3(path)
                file = path
            elif (strext == '.tm2' or strext == '.tmy'):
                weatherdata = pvlib.iotools.tmy.read_tmy2(path)
                file = path
            elif strext == '.xls':
                pass
            else:
                print('unknow extension ', strext)
                return

            self.varsitename.set(weatherdata[1]['City'])
            self.varsitelat.set(weatherdata[1]['latitude'])
            self.varsitelong.set(weatherdata[1]['longitude'])
            self.varsitealt.set(weatherdata[1]['altitude'])

    def select_fluid_from_csv(self):

        # abre una ventana de seleccón de csv.
        self.loadWindow= tk.Tk()
        self.loadWindow.attributes('-fullscreen', False)
        self.loadWindow.title('Selection Window')

        # Menu
        self.loadWindow.menubar = tk.Menu(
            self.loadWindow)
        self.loadWindow.load_menu = tk.Menu(
            self.loadWindow.menubar, tearoff=0)
        self.loadWindow.load_menu.add_command(
            label='Open', command=self.open_csv())
        self.loadWindow.load_menu.add_separator()
        self.loadWindow.load_menu.add_command(
            label='Exit', command=None)
        self.loadWindow.menubar.add_cascade(
            label='Select File', menu=self.loadWindow.load_menu)

        self.loadWindow.config(menu=self.loadWindow.menubar)

    def open_csv(self, path=None):

```

```
        df = pd.DataFrame()

        try:
            if path is None:
                root = Tk()
                root.withdraw()
                path = askopenfilename(initialdir = '.fielddata_files/',
                                       title='choose your file',
                                       filetypes=(('csv files','*.csv'),
                                                  ('all files','*.*')))
                root.update()
                root.destroy()

                if path is None:
                    return
                else:
                    strfilename, strext = os.path.splitext(path)

                    if  strext == '.csv':
                        print('csv........')

                        df = pd.read_csv(path, sep=';',
                                                  decimal= ',')
                        file = path
                    else:
                        print('unknow extension ', strext)
                        return
            else:
                strfilename, strext = os.path.splitext(path)

                if  strext == '.csv':
                    print('csv...')
                    df = pd.read_csv(path, sep=';',
                                                   decimal= ',')
                    file = path
                else:
                    print('unknow extension ', strext)
                    return

        except Exception:
            raise

    def load_fluid_library(self):

        path = askopenfilename(initialdir = self._DIR['fluid_files'],
                               title='choose your file',
                               filetypes=[('JSON files', '*.json')])

        with open(path) as cfg_file:
            cfg = json.load(cfg_file, parse_float= float, parse_int= int)

        self.fluid_list = cfg

        self.cmbfluidname['values'] = [s['name'] for s in cfg ]
        self.cmbfluidname.current(0)
        self.load_fluid_parameters()

    def load_fluid_parameters(self):

        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.rbfluidlib = 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.rbfluidtable = 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.varscaname.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.varscaIAMF0.set(self.sca_config['IAM Coefficient F0'])
1466        self.varscaIAMF1.set(self.sca_config['IAM Coefficient F1'])
1467        self.varscaIAMF2.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.varscaavailability.set(self.sca_config['Availability'])
1474        self.updateHCEperSCA()
1475
1476    def buildSCAFrame(self):
1477
1478        self.sca_list = []
1479
1480        self.varscaname = 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.varscaIAMF0 = tk.DoubleVar(self.fr_sca)
1485        self.varscaIAMF1 = tk.DoubleVar(self.fr_sca)
1486        self.varscaIAMF2 = 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.varscaavailability = tk.DoubleVar(self.fr_sca)
1493
1494        self.lbscaname = ttk.Label(
1495            self.fr_sca,
1496            text='SCA base name').grid(
```

```
                row=0, column=0, sticky='W', padx=2, pady=5)

        self.enscaname = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscaname).grid(
                row=0, column=1, sticky='W', padx=2, pady=5)

        self.btscaload = ttk.Button(
            self.fr_sca, text='Load Library',
            command=lambda: self.load_sca_library())
        self.btscaload.grid(row=0, column=7, sticky='W', padx=2, pady=5)

        self.cmbscaname = ttk.Combobox(self.fr_sca)
        self.cmbscaname.bind(
            '<<ComboboxSelected>>', lambda event: self.load_sca_parameters())
        self.cmbscaname['values'] = self.sca_list
        self.cmbscaname.grid(row=0, column=3, columnspan=4, padx=5, pady=5)

        self.lbscalength = ttk.Label(
            self.fr_sca,
            text='SCA Length [m]').grid(
                row=1, column=0,  sticky='W', padx=2, pady=5)
        self.enscalength = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscalength)
        self.enscalength.bind('<Key>', lambda event: self.updateHCEperSCA())
        self.enscalength.grid(
                row=1, column=1, sticky='W', padx=2, pady=5)

        self.lbscaaperture = ttk.Label(
            self.fr_sca,
            text='SCA aperture [m]').grid(
                row=2, column=0,  sticky='W', padx=2, pady=5)
        self.enscaaperture = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscaaperture).grid(
                row=2, column=1, sticky='W', padx=2, pady=5)

        self.lbscafocallen = ttk.Label(
            self.fr_sca,
            text='SCA focal length [m]').grid(
                row=3, column=0,  sticky='W', padx=2, pady=5)
        self.enscafocallen = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscafocallen).grid(
                row=3, column=1, sticky='W', padx=2, pady=5)

        self.lbscaIAMF0 = ttk.Label(
            self.fr_sca,
            text='SCA IAM facotr F0 []').grid(
                row=4, column=0,  sticky='W', padx=2, pady=5)
        self.enscaIAMF0 = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscaIAMF0).grid(
                row=4, column=1, sticky='W', padx=2, pady=5)

        self.lbscaIAMF1 = ttk.Label(
            self.fr_sca,
            text='SCA IAM facotr F1 []').grid(
                row=5, column=0,  sticky='W', padx=2, pady=5)
```

```
        self.enscaIAMF1 = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscaIAMF1).grid(
                row=5, column=1, sticky='W', padx=2, pady=5)

        self.lbscaIAMF2 = ttk.Label(
            self.fr_sca,
            text='SCA IAM facotr F2 []').grid(
                row=6, column=0,  sticky='W', padx=2, pady=5)
        self.enscaIAMF2 = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscaIAMF2).grid(
                row=6, column=1, sticky='W', padx=2, pady=5)

        self.lbscareflectance = ttk.Label(
            self.fr_sca,
            text='SCA mirror reflectance []').grid(
                row=7, column=0,  sticky='W', padx=2, pady=5)
        self.enscareflectance = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscareflectance).grid(
                row=7, column=1, sticky='W', padx=2, pady=5)

        self.lbscageoaccuracy = ttk.Label(
            self.fr_sca,
            text='SCA geometric accuracy []').grid(
                row=8, column=0,  sticky='W', padx=2, pady=5)
        self.enscageoaccuracy = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscageoaccuracy).grid(
                row=8, column=1, sticky='W', padx=2, pady=5)

        self.lbscatracktwist = ttk.Label(
            self.fr_sca,
            text='SCA Tracking Twist []').grid(
                row=9, column=0,  sticky='W', padx=2, pady=5)
        self.enscatracktwist = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscatracktwist).grid(
                row=9, column=1, sticky='W', padx=2, pady=5)

        self.lbscacleanliness = ttk.Label(
            self.fr_sca,
            text='SCA Cleanliness []').grid(
                row=10, column=0,  sticky='W', padx=2, pady=5)
        self.enscacleanliness = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscacleanliness).grid(
                row=10, column=1, sticky='W', padx=2, pady=5)

        self.lbscafactor = ttk.Label(
            self.fr_sca,
            text='SCA Factor []').grid(
                row=11, column=0,  sticky='W', padx=2, pady=5)
        self.enscafactor = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscafactor).grid(
                row=11, column=1, sticky='W', padx=2, pady=5)

        self.lbscaavailability = ttk.Label(
```

```
            self.fr_sca,
            text='SCA Availability []').grid(
                row=12, column=0,  sticky='W', padx=2, pady=5)
        self.enscaavailability = ttk.Entry(
            self.fr_sca,
            textvariable=self.varscaavailability).grid(
                row=12, column=1, sticky='W', padx=2, pady=5)

    def load_hce_parameters(self):

        self.hce_config = {}

        for item in self.hce_list:

            if item['Name'] == self.cmbhcename.get():
                self.hce_config = item

        self.varhcename.set(self.hce_config['Name'])
        self.varhcedri.set(self.hce_config['Absorber tube inner diameter'])
        self.varhcedro.set(self.hce_config['Absorber tube outer diameter'])
        self.varhcedgi.set(self.hce_config['Glass envelope inner diameter'])
        self.varhcedgo.set(self.hce_config['Glass envelope outer diameter'])
        self.varhcelength.set(self.hce_config['Length'])
        self.varhceinnerroughness.set(self.hce_config['Inner surface roughness'])
        self.varhceminreynolds.set(self.hce_config['Min Reynolds'])
        self.varhceemittanceA0.set(self.hce_config['Absorber emittance factor A0'])
        self.varhceemittanceA1.set(self.hce_config['Absorber emittance factor A1'])
        self.varhceabsorptance.set(self.hce_config['Absorber absorptance'])
        self.varhcetransmittance.set(self.hce_config['Envelope transmittance'])
        self.varhcebrackets.set(self.hce_config['Brackets'])
        self.updateHCEperSCA()

    def load_hce_library(self, path = None):

        if path is None:
            path = askopenfilename(initialdir = self._DIR['hce_files'],
                           title='choose your file',
                           filetypes=[('JSON files', '*.json')])

        with open(path) as cfg_file:
            cfg = json.load(cfg_file, parse_float= float, parse_int= int)

        self.hce_list = cfg
        self.cmbhcename['values'] = [s['Name'] for s in cfg ]
        self.cmbhcename.current(0)
        self.load_hce_parameters()


    def updateHCEperSCA(self):

        var1 = self.varscalength.get()
        var2 = self.varhcelength.get()

        if var2 != 0:
            self.varhcepersca.set(var1 // var2)
        else:
            self.varhcepersca.set(0.0)

        var3 = self.varhcepersca.get()
        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,
```

```
            text='HCE outer diameter [m]').grid(
                row=2, column=0,  sticky='W', padx=2, pady=5)
        self.enhcedro = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcedro).grid(
                row=2, column=1, sticky='W', padx=2, pady=5)

        self.lbhcedgi = ttk.Label(
            self.fr_hce,
            text='HCE glass inner diameter [m]').grid(
                row=3, column=0,  sticky='W', padx=2, pady=5)
        self.enhcedgi = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcedgi).grid(
                row=3, column=1, sticky='W', padx=2, pady=5)

        self.lbhcedgo = ttk.Label(
            self.fr_hce,
            text='HCE glass outer diameter [m]').grid(
                row=4, column=0,  sticky='W', padx=2, pady=5)
        self.enhcedgo = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcedgo).grid(
                row=4, column=1, sticky='W', padx=2, pady=5)

        self.lbhcelong = ttk.Label(
            self.fr_hce,
            text='HCE longitude [m]').grid(
                row=5, column=0,  sticky='W', padx=2, pady=5)
        self.enhcelong = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcelength)
        self.enhcelong.bind('<Key>', lambda event: self.updateHCEperSCA())
        self.enhcelong.grid(row=5, column=1, sticky='W', padx=2, pady=5)

        self.lbhcepersca = ttk.Label(
            self.fr_hce,
            textvariable=self.varhceperscatext).grid(
                row=5, column=2, sticky='W', padx=2, pady=5)

        self.lbbellowsratio = ttk.Label(
            self.fr_hce,
            text='Bellows ratio  [ ]').grid(
                row=6, column=0,  sticky='W', padx=2, pady=5)
        self.enbellowsratio = ttk.Entry(
            self.fr_hce,
            textvariable=self.varbellowsratio).grid(
                row=6, column=1, sticky='W', padx=2, pady=5)

        self.lbshieldshading = ttk.Label(
            self.fr_hce,
            text='Shield shading  [ ]').grid(
                row=7, column=0,  sticky='W', padx=2, pady=5)
        self.enshieldshading = ttk.Entry(
            self.fr_hce,
            textvariable=self.varshieldshading).grid(
                row=7, column=1, sticky='W', padx=2, pady=5)

        self.lbbrackets = ttk.Label(
            self.fr_hce,
```

```
            text='Brackets spacing').grid(
                row=8, column=0,  sticky='W', padx=2, pady=5)
        self.enbrackets = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcebrackets).grid(
                row=8, column=1, sticky='W', padx=2, pady=5)

        self.lbhceinnerroughness = ttk.Label(
            self.fr_hce,
            text='HCE inner roughness [ ]').grid(
                row=10, column=0,  sticky='W', padx=2, pady=5)
        self.enhceinnerroughness = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhceinnerroughness).grid(
                row=10, column=1, sticky='W', padx=2, pady=5)

        self.lbhceminreynolds = ttk.Label(
            self.fr_hce,
            text='HCE minimum Reynolds Number [ ]').grid(
                row=11, column=0,  sticky='W', padx=2, pady=5)
        self.enhceminreynolds = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhceminreynolds).grid(
                row=11, column=1, sticky='W', padx=2, pady=5)

        self.lbhceabsorptance = ttk.Label(
            self.fr_hce,
            text='Absorptance [ ]').grid(
                row=12, column=0,  sticky='W', padx=2, pady=5)
        self.enhceabsorptance = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhceabsorptance).grid(
                row=12, column=1, sticky='W', padx=2, pady=5)

        self.lbhcetransmittance = ttk.Label(
            self.fr_hce,
            text='Transmittance [ ]').grid(
                row=13, column=0,  sticky='W', padx=2, pady=5)
        self.enhcetransmittance = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcetransmittance).grid(
                row=13, column=1, sticky='W', padx=2, pady=5)

        self.lbemittanceA0 = ttk.Label(
            self.fr_hce,
            text='Emittance Factor A0 [ ]').grid(
                row=14, column=0,  sticky='W', padx=2, pady=5)
        self.enemittanceA0 = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhceemittanceA0).grid(
                row=14, column=1, sticky='W', padx=2, pady=5)

        self.lbemittanceA1 = ttk.Label(
            self.fr_hce,
            text='Emittance Factor A1  [ ]').grid(
                row=15, column=0,  sticky='W', padx=2, pady=5)
        self.enemittanceA1 = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhceemittanceA1).grid(
                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()
```