```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 csenergy as cs
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):
```

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

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

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

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

```
            subfields.append(sf)

        cfg['solarfield'].update({'subfields' : subfields})
        # cfg_settings['solarfield'].update(dict(cfg))
        f.write(json.dumps(cfg))
        f.close()

    def showTagsTable(self):

        self.msg = tk.Tk()
        #self.fr_tags = tk.Frame(self.msg, )
        self.strtags = tk.StringVar(self.msg)

        for row in self.tagslist:
            self.strtags.set(self.strtags.get() + '# ' +
                             str(row[0]) + ' ---> ' +
                             str(row[1]) + ' \n')

        self.lbtagslist = ttk.Label(self.msg,
                                    textvariable =self.strtags).pack()

        self.msg.mainloop()

    def openTagsWizard(self):

        tags_table = []

        for tag in self.tagslist:
            tags_table.append(['', tag])

        columns_names = []

        if self.varsimdatatype.get() == 2: # Data from field data file

            columns_names.append(['DNI','',''])
            columns_names.append(['Wspd','',''])
            columns_names.append(['DryBulb','',''])
            columns_names.append(['Pressure','',''])
            columns_names.append(['GrossPower','',''])
            columns_names.append(['AuxPower','',''])
            columns_names.append(['NetPower','',''])

        if self.varbenchmark.get():

            for row in self.solarfield_table.table_data:
                columns_names.append(['SB.'+row[0]+'.a.mf','',''])
                columns_names.append(['SB.'+row[0]+'.a.tin','',''])
                columns_names.append(['SB.'+row[0]+'.a.tout','',''])
                columns_names.append(['SB.'+row[0]+'.a.pin','',''])
                columns_names.append(['SB.'+row[0]+'.a.pout','',''])

        self.columns_table.table_data = columns_names

        self.showTagsTable()

    def loadSelectedTags(self):

        index = 0
        for row in self.columns_table.table_data:
            tag_index = self.to_number(row[1])
```

```
            if  isinstance(tag_index, int):
                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(
            row=7, column=0, columnspan =2, padx=5, pady=5)

        self.columns_table = table.Tk_Table(
            self.fr_solarfield,
            ['      COLUMN      ', 'TAG #', '       TAG       '],
            row_numbers=True,
            stripped_rows=('white', '#f2f2f2'),
            select_mode='none',
            cell_anchor='center',
            adjust_heading_to_content=True)
        self.columns_table.grid(
            row=7, column=2, columnspan=2, padx=5, pady=5)

        self.frame0 = ttk.Frame(self.fr_solarfield)
        self.frame0.grid(row=8, column=0, columnspan=2, padx=2, pady=5)

        self.btnewrow = ttk.Button(
            self.frame0,
            text='Insert',
            command=lambda: self.__insert_rows__(self.solarfield_table))
        self.btnewrow.pack(side=tk.LEFT)

        self.btdelrows = ttk.Button(
            self.frame0,
            text='Delete',
            command=lambda: self.__del_rows__(self.solarfield_table))
        self.btdelrows.pack(side=tk.LEFT)

        self.frame1 = ttk.Frame(self.fr_solarfield)
        self.frame1.grid(row=8, column=2, columnspan=2, padx=2, pady=5)

        self.bttagswizard = ttk.Button(
            self.frame1,
            text='Open TAGS wizard',
            command=lambda: self.openTagsWizard())
        self.bttagswizard.pack(side=tk.LEFT)

        self.btloadtags = ttk.Button(
            self.frame1,
            text='Load TAGs',
            command=lambda: self.loadSelectedTags())
        self.btloadtags.pack(side=tk.LEFT)

    #  Site & Weather contruction Tab
    def dataLoadDialog(self, title, labeltext=''):

        if self.varsimdatatype.get() == 1:
            path = askopenfilename(
                initialdir=self._DIR['weather_files'],
                title='choose your file',
                filetypes=(('TMY files', '*.tm2'),
                           ('TMY files', '*.tm3'),
                           ('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))
        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))
            df = pd.read_csv(path, sep=';',
                                            decimal= ',',
                                            dayfirst=True,
                                            index_col=0)
            count = 0
            for r in list(df):
                count += 1
                self.tagslist.append([count, r])
        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 = {}

        for item in self.fluid_list:

            if item['name'] == self.cmbfluidname.get():
                self.fluid_config = item

        datarow=[]

        for parameter in self.fluid_config.keys():
            if parameter in ['cp','mu','rho','kt','h','t']:
                datarow.append([parameter] + self.fluid_config[parameter])

        self.fluid_table.table_data = datarow

        self.varfluidname.set(self.fluid_config['name'])
        self.varfluidtmin.set(self.fluid_config['tmin'])
        self.varfluidtmax.set(self.fluid_config['tmax'])

    def fluid_load_dialog(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)

        cp_coefs = [[]]*10
        rho_coefs = [[]]*10
        mu_coefs = [[]]*10
        kt_coefs = [[]]*10
        h_coefs = [[]]*10
        t_coefs = [[]]*10

        temp_cp = ['cp']
        temp_rho = ['rho']
        temp_mu = ['mu']
        temp_kt = ['kt']
        temp_h = ['h']
        temp_t = ['t']

        grades = ['Factor']
        grades.extend([0, 1, 2, 3, 4, 5, 6, 7, 8])

        temp_cp.extend(list(cfg['hot_fluid']['cp']))
        temp_rho.extend(list(cfg['hot_fluid']['rho']))
        temp_mu.extend(list(cfg['hot_fluid']['mu']))
        temp_kt.extend(list(cfg['hot_fluid']['kt']))
        temp_h.extend(list(cfg['hot_fluid']['h']))
        temp_t.extend(list(cfg['hot_fluid']['t']))

```

```
        for index in range(len(temp_cp)):
            cp_coefs[index] = temp_cp[index]
        cp_coefs[1:] = [Decimal(s) for s in cp_coefs[1:]]
        for index in range(len(temp_rho)):
            rho_coefs[index] = temp_rho[index]
        rho_coefs[1:] = [Decimal(s) for s in rho_coefs[1:]]
        for index in range(len(temp_mu)):
            mu_coefs[index] = temp_mu[index]
        mu_coefs[1:] = [Decimal(s) for s in mu_coefs[1:]]
        for index in range(len(temp_kt)):
            kt_coefs[index] = temp_kt[index]
        kt_coefs[1:] = [Decimal(s) for s in kt_coefs[1:]]
        for index in range(len(temp_h)):
            h_coefs[index] = temp_h[index]
        h_coefs[1:] = [Decimal(s) for s in h_coefs[1:]]
        for index in range(len(temp_t)):
            t_coefs[index] = temp_t[index]
        t_coefs[1:] = [Decimal(s) for s in t_coefs[1:]]

        datarow = []
        datarow.append(cp_coefs)
        datarow.append(rho_coefs)
        datarow.append(mu_coefs)
        datarow.append(kt_coefs)
        datarow.append(h_coefs)
        datarow.append(t_coefs)

        self.fluid_table.table_data = datarow
        self.varfluidtmin.set(cfg['tmin'])
        self.varfluidtmax.set(cfg['tmax'])
        self.varfluidname.set(cfg['name'])

        self.fr_fluid.update()

    def checkfluid(self):

        var = self.varfluidtable.get()

        if var == 1:  # Fluid from table
            self.cmbcoolpropID.set('')
            self.cmbcoolpropID['state'] = 'disabled'
            self.enfluidname['state'] = 'normal'
            self.btloadfluidcfg['state'] = 'normal'
            self.enfluidtmin['state'] = 'normal'
            self.enfluidtmax['state'] = 'normal'
            self.varfluidtmax.set('')
            self.varfluidtmin.set('')
            self.varcoolproptmin.set('')
            self.varcoolproptmax.set('')
        elif var == 2:  # Fluid from CoolProp
            self.varcoolproptmax.set('')
            self.varcoolproptmin.set('')
            self.varfluidname.set('')
            self.cmbcoolpropID['state'] = 'readonly'
            self.enfluidname['state'] = 'disabled'
            self.btloadfluidcfg['state'] = 'disabled'
            self.enfluidtmin['state'] = 'disabled'
            self.enfluidtmax['state'] = 'disabled'
            self.varfluidtmax.set('')
            self.varfluidtmin.set('')
```

```
            self.varcoolproptmin.set('')
            self.varcoolproptmax.set('')
            self.fluid_table.table_data = []
        else:
            pass

    def get_coolprop_data(self):

        self.varcoolproptmin.set(CP.PropsSI("TMIN", self.cmbcoolpropID.get()))
        self.varcoolproptmax.set(CP.PropsSI("TMAX", self.cmbcoolpropID.get()))


    def buildFluidFrame(self):

        self.varfluidtmin = tk.DoubleVar(self.fr_fluid)
        self.varfluidtmax = tk.DoubleVar(self.fr_fluid)
        self.varcoolproptmin = tk.DoubleVar(self.fr_fluid)
        self.varcoolproptmax = tk.DoubleVar(self.fr_fluid)
        self.varfluidname = tk.StringVar(self.fr_fluid)
        self.fluid_list = []
        self.varfluidtable = tk.IntVar(self.fr_fluid)

        self.varfluidtable.set(1)  # 1 for Table, 2 for CoolProp Library

        #  RadioButton for fluid from library
        self.rbfluidlib = tk.Radiobutton(
            self.fr_fluid,
            padx=1,
            text='Fluid Data from CoolProp',
            variable=self.varfluidtable, value=2,
            command=lambda: self.checkfluid()).grid(
                row=0, column=0, sticky='W', padx=1, pady=5)

        self.lbcoolpropID = tk.Label(
            self.fr_fluid, text='CoolProp ID (INCOMP::xxxx)')
        self.lbcoolpropID.grid(row=0, column=1, sticky='W', padx=1, pady=5)

        self.cmbcoolpropID = ttk.Combobox(self.fr_fluid)
        self.cmbcoolpropID.bind(
            "<<ComboboxSelected>>", lambda event: self.get_coolprop_data())
        self.cmbcoolpropID['values'] = self._COOLPROP_FLUIDS
        self.cmbcoolpropID['state'] = 'readonly'
        self.cmbcoolpropID.current(0)
        self.cmbcoolpropID.grid(row=0, column=2, sticky='W', padx=1, pady=5)

        self.lbcoolproptmintext = tk.Label(
            self.fr_fluid,
            text='Tmin[K]')
        self.lbcoolproptmintext.grid(
            row=0, column=3, sticky='E', padx=1, pady=5)

        self.lbcoolproptmin = tk.Label(
            self.fr_fluid,
            textvariable=self.varcoolproptmin)
        self.lbcoolproptmin.grid(
            row=0, column=4, sticky='E', padx=1, pady=5)

        self.lbcoolproptmaxtext = tk.Label(
            self.fr_fluid,
            text='Tmax[K]')
```

```
        self.lbcoolproptmaxtext.grid(
            row=0, column=5, sticky='W', padx=1,pady=5)

        self.lbcoolproptmax = tk.Label(
            self.fr_fluid,
            textvariable=self.varcoolproptmax)
        self.lbcoolproptmax.grid(
            row=0, column=6, sticky='W', padx=1, pady=5)


        self.separator = ttk.Separator(self.fr_fluid).grid(
            row=1, column=0, columnspan=99, sticky=(tk.W, tk.E))

        #  RadioButton for fluid from table
        self.rbfluidtable = tk.Radiobutton(
            self.fr_fluid,
            padx=2,
            text = 'Fluid Data from table',
            variable=self.varfluidtable, value=1,
            command=lambda: self.checkfluid()).grid(
                row=2, column=0, columnspan=2,  sticky='W', padx=2, pady=5)

        self.lbfluidname = ttk.Label(self.fr_fluid, text='Name')
        self.lbfluidname.grid(row=3, column=0, sticky='W', padx=2, pady=5)
        self.enfluidname = ttk.Entry(self.fr_fluid, textvariable=self.varfluidname)
        self.enfluidname.grid(row=3, column=1, sticky='W', padx=2, pady=5)

        self.cmbfluidname = ttk.Combobox(self.fr_fluid)
        self.cmbfluidname.bind('<<ComboboxSelected>>',
                               lambda event: self.load_fluid_parameters())
        self.cmbfluidname['values'] = self.fluid_list
        self.cmbfluidname.grid(row=3, column=2, padx=5, pady=5)

        self.btloadfluidcfg = ttk.Button(
            self.fr_fluid, text='Load config',
            command=lambda: self.load_fluid_library())
        self.btloadfluidcfg.grid(
            row=3, column=3, sticky='W', padx=2, pady=5)

        self.lbfluidtmin = ttk.Label(self.fr_fluid, text='Tmin[K]')
        self.lbfluidtmin.grid(row=4, column=0, sticky='W', padx=2, pady=5)
        self.enfluidtmin = ttk.Entry(
            self.fr_fluid, textvariable=self.varfluidtmin)
        self.enfluidtmin.grid(row=4, column=1, sticky='W', padx=2, pady=5)

        self.lbfluidtmax = ttk.Label(self.fr_fluid, text='Tmax [K]')
        self.lbfluidtmax.grid(row=4,column=2, sticky='W', padx=2, pady=5)
        self.enfluidtmax = ttk.Entry(self.fr_fluid, textvariable=self.varfluidtmax)
        self.enfluidtmax.grid(row=4, column=3, sticky='W', padx=2, pady=5)

        self.fluid_table = table.Tk_Table(
                self.fr_fluid,
                ['Parameter',
                 '         A x^0', '         B x^1',
                 '         C x^2', '         D x^3',
                 '         E x^4', '         F x^5',
                 '         G x^6', '         H x^7',
                 '         I x^8'],
                row_numbers=True,
                stripped_rows=('white', '#f2f2f2'),
```

```
                select_mode='none',
                cell_anchor='e',
                adjust_heading_to_content=True)
        self.fluid_table.grid(
            row=5, column=0, columnspan=7, sticky='W', padx=2, pady=5)

        self.checkfluid()

    # SCA Construction tab
    def load_sca_library(self):

        path = askopenfilename(initialdir=self._DIR['sca_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.sca_list = cfg
        self.cmbscaname['values'] = [s['Name'] for s in cfg ]
        self.cmbscaname.current(0)
        self.load_sca_parameters()
        self.checkfluid()


    def load_sca_parameters(self):

        self.sca_config = {}

        for item in self.sca_list:
            if item['Name'] == self.cmbscaname.get():
                self.sca_config = item

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

    def buildSCAFrame(self):

        self.sca_list = []

        self.varscaname = tk.StringVar(self.fr_sca)
        self.varscalength = tk.DoubleVar(self.fr_sca)
        self.varscaaperture = tk.DoubleVar(self.fr_sca)
        self.varscafocallen = tk.DoubleVar(self.fr_sca)
        self.varscaIAMF0 = tk.DoubleVar(self.fr_sca)
        self.varscaIAMF1 = tk.DoubleVar(self.fr_sca)
        self.varscaIAMF2 = tk.DoubleVar(self.fr_sca)
        self.varscareflectance = tk.DoubleVar(self.fr_sca)
```

```
        self.varscageoaccuracy = tk.DoubleVar(self.fr_sca)
        self.varscatracktwist = tk.DoubleVar(self.fr_sca)
        self.varscacleanliness = tk.DoubleVar(self.fr_sca)
        self.varscafactor = tk.DoubleVar(self.fr_sca)
        self.varscaavailability = tk.DoubleVar(self.fr_sca)

        self.lbscaname = ttk.Label(
            self.fr_sca,
            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 factor 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 factor 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 factor 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()

        if var3 >= var4:
            self.varhceperscatext.set('Max. ' + str(var3) + ' HCE per SCA')
        else:
            self.varhceperscatext.set(
                'Error: ' + str(var4) +
                ' exceeded max. HCE per SCA =' + str(var3))

    def buildHCEFrame(self):

        self.hce_list = []
        self.varhcename = tk.StringVar(self.fr_hce)
        self.varhcedri = tk.DoubleVar(self.fr_hce)
        self.varhcedro = tk.DoubleVar(self.fr_hce)
        self.varhcedgi = tk.DoubleVar(self.fr_hce)
        self.varhcedgo = tk.DoubleVar(self.fr_hce)
        self.varhcelength = tk.DoubleVar(self.fr_hce)
        self.varhceinnerroughness = tk.DoubleVar(self.fr_hce)
        self.varhceminreynolds = tk.DoubleVar(self.fr_hce)
        self.varhcepersca = tk.DoubleVar(self.fr_hce)
        self.varhceperscatext = tk.StringVar(self.fr_hce)
        self.varhceabsorptance = tk.DoubleVar(self.fr_hce)
        self.varhcetransmittance = tk.DoubleVar(self.fr_hce)
        self.varhceemittanceA0 = tk.DoubleVar(self.fr_hce)
        self.varhceemittanceA1 = tk.DoubleVar(self.fr_hce)
        self.varbellowsratio = tk.DoubleVar(self.fr_hce)
        self.varshieldshading = tk.DoubleVar(self.fr_hce)
        self.varhcebrackets = tk.DoubleVar(self.fr_hce)

        self.lbhcename = ttk.Label(
            self.fr_hce,
            text='HCE base name').grid(
                row=0, column=0, sticky='W', padx=2, pady=5)
        self.enhcename = ttk.Entry(
            self.fr_hce,
            textvariable=self.varhcename).grid(
                row=0, column=1, sticky='W', padx=2, pady=5)

        self.cmbhcename = ttk.Combobox(self.fr_hce)
        self.cmbhcename.bind('<<ComboboxSelected>>', lambda event:
self.load_hce_parameters())
        self.cmbhcename['values'] = self.hce_list
        self.cmbhcename.grid(row=0, column=2, padx=5, pady=5)

        self.bthceload = ttk.Button(
            self.fr_hce, text='Load Library',
            command=lambda: self.load_hce_library())
        self.bthceload.grid(row=0, column=3, sticky='W', padx=2, pady=5)

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

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

        self.lbhcedro = ttk.Label(
            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)
```

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