STM_ICT Project 2021 SP2 #28 Simulate Sensor Data and Dashboard

Configure UI Code Documentation

Team member:
YuxuanLi, YifeiRan, Magura, Munyaradzi

CONTEXT

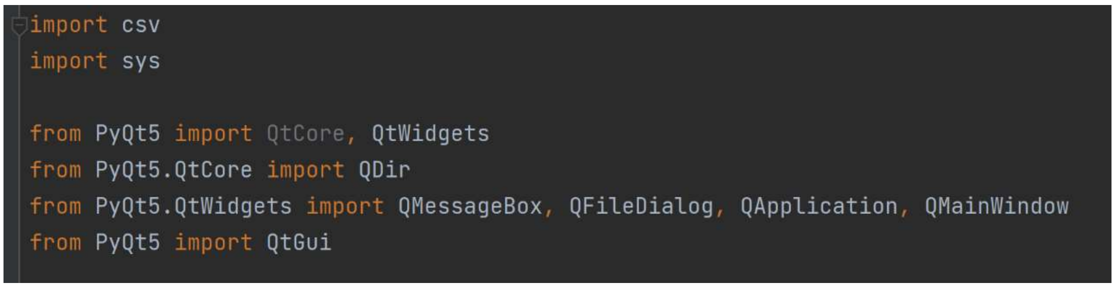
Overview

This document is used to describe the information about configuring the UI module in the project and instructions about its operation.
This document is divided into three parts, resource, user interaction and implementation method

Resource & Configuration requirement

This module contains two classes: window class (configurationUI.py) and parameter class (Configuration_parameter.py). Users need to place the files in the same project file directory.

Before running, users need to install or import the following libraries:
1. pyqt5
2. csv
3. sys



```python
import csv
import sys

from PyQt5 import QtCore, QtWidgets
from PyQt5.QtCore import QDir
from PyQt5.QtWidgets import QMessageBox, QFileDialog, QApplication, QMainWindow
from PyQt5 import QtGui
```

About pyqt5
Library website: https://pypi.org/project/PyQt5/

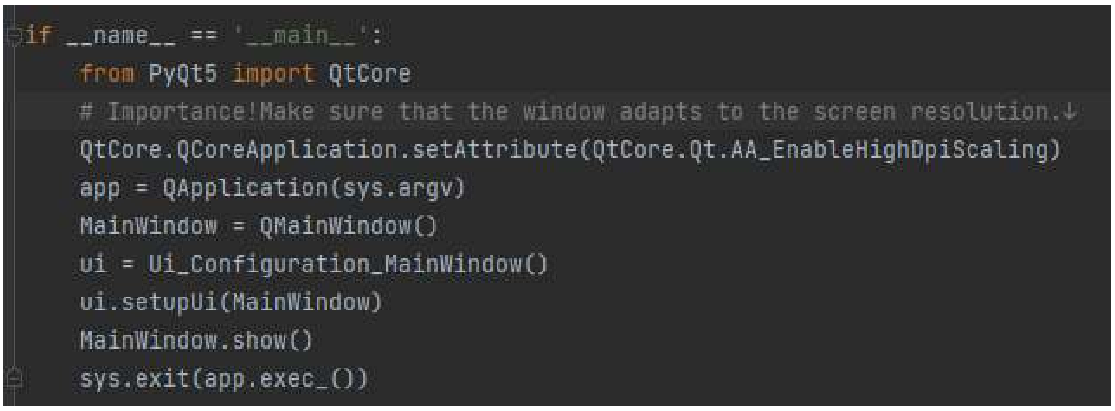
Installation Notes:
The GPL version of PyQt5 can be installed from PyPI:

```
pip install PyQt5
```

User Interaction

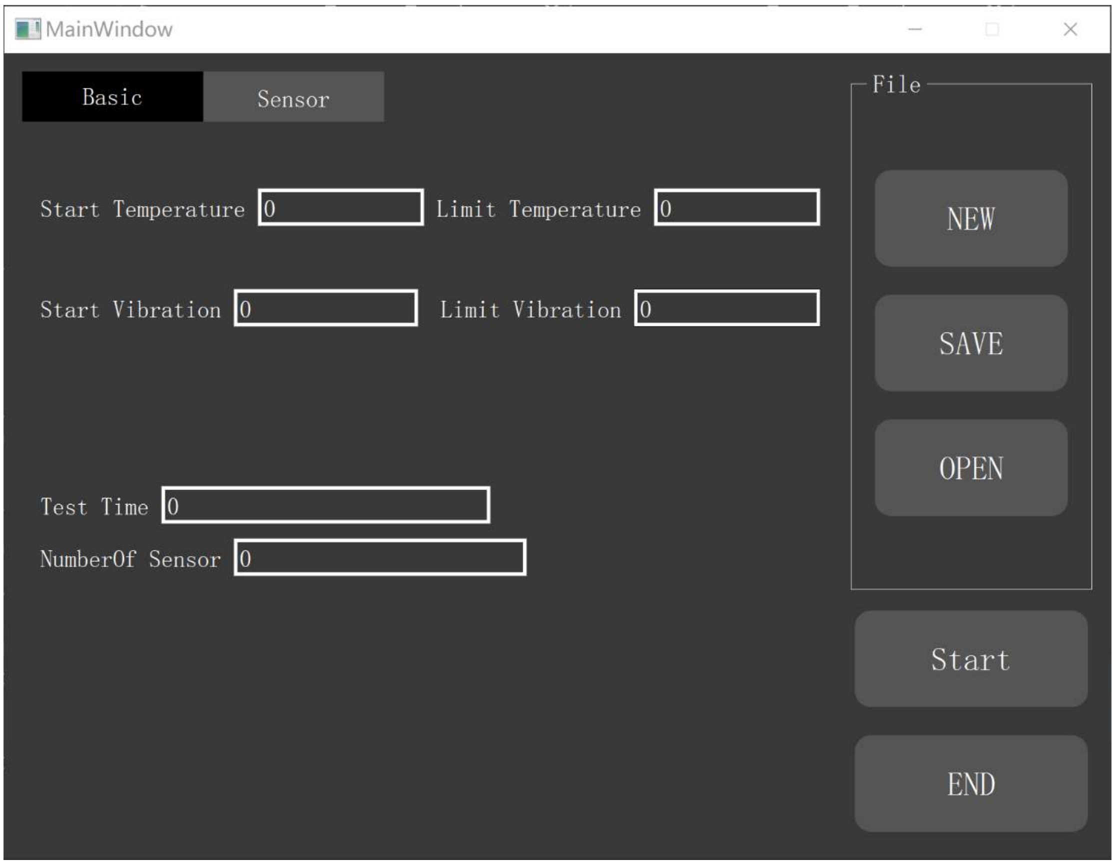
The UI provides a window for the user to enter the simulation configuration and control the simulation process.

Users can use this interface to run through the following methods:



```
if __name__ == '__main__':
    from PyQt5 import QtCore
    # Importance!Make sure that the window adapts to the screen resolution.
    QtCore.QCoreApplication.setAttribute(QtCore.Qt.AA_EnableHighDpiScaling)
    app = QApplication(sys.argv)
    MainWindow = QMainWindow()
    ui = Ui_Configuration_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

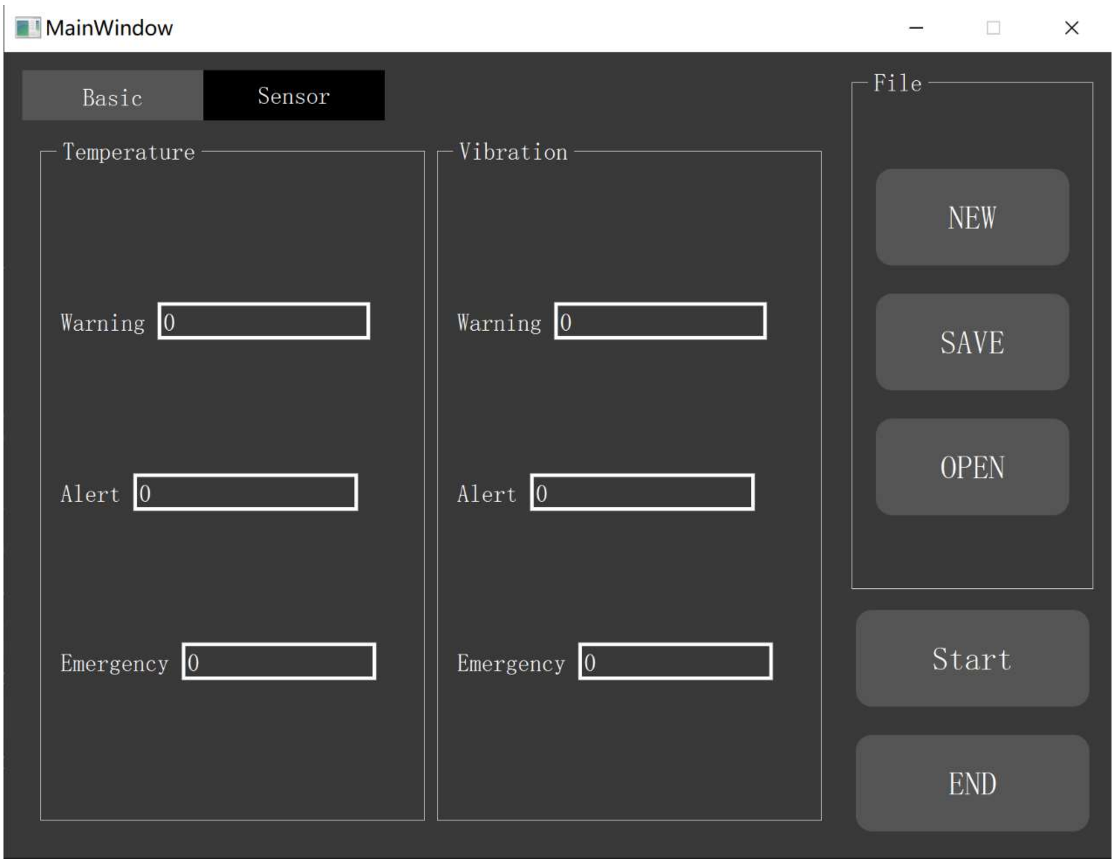
After the user runs the code, the following window will be displayed:
In the interface, users can enter the parameters they want to import to the DES module in the input box. The user can switch the input page "Basic" and "Sensor" through the TAG button on the interface.



MainWindow

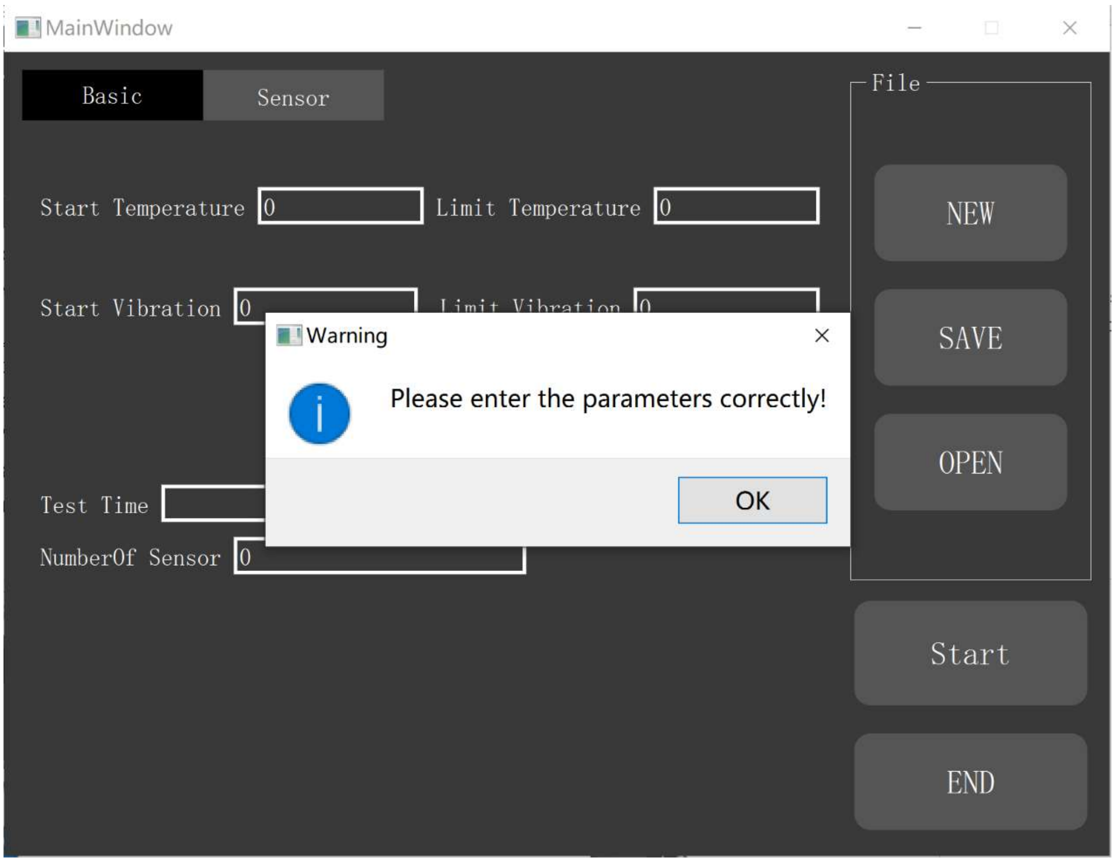
Basic | Sensor

Start Temperature: 0 | Limit Temperature: 0
Start Vibration: 0 | Limit Vibration: 0
Test Time: 0
NumberOf Sensor: 0



Only numeric type parameters can be entered in the input box. Other type parameters cannot be entered in the box (that is, the module function prevents the user from entering characters other than numbers). Refer to the following text for specific implementation methods.

In addition, the input box cannot be empty, otherwise it will be judged as illegal input and will not continue to the next step.



Start Temperature 0 Limit Temperature 0

Start Vibration 0 Limit Vibration 0

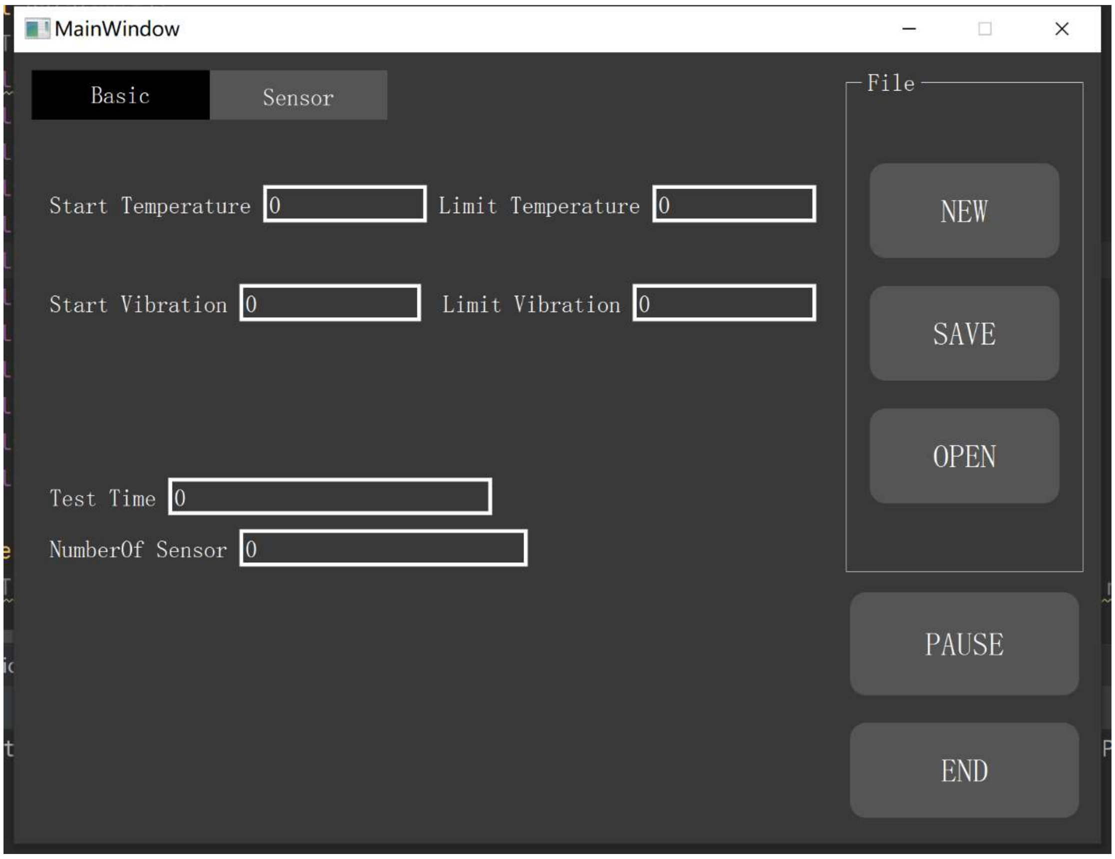
Test Time

NumberOf Sensor 0

Warning

Please enter the parameters correctly!

After clicking the button "START", in order to ensure that the configuration parameters will not be modified by mistake, all input box input permissions will be closed.
When the user clicks "PAUSE", the input permission of the input box will be opened again.
At this time, users can choose to modify their parameters.



Start Temperature 0 Limit Temperature 0

Start Vibration 0 Limit Vibration 0

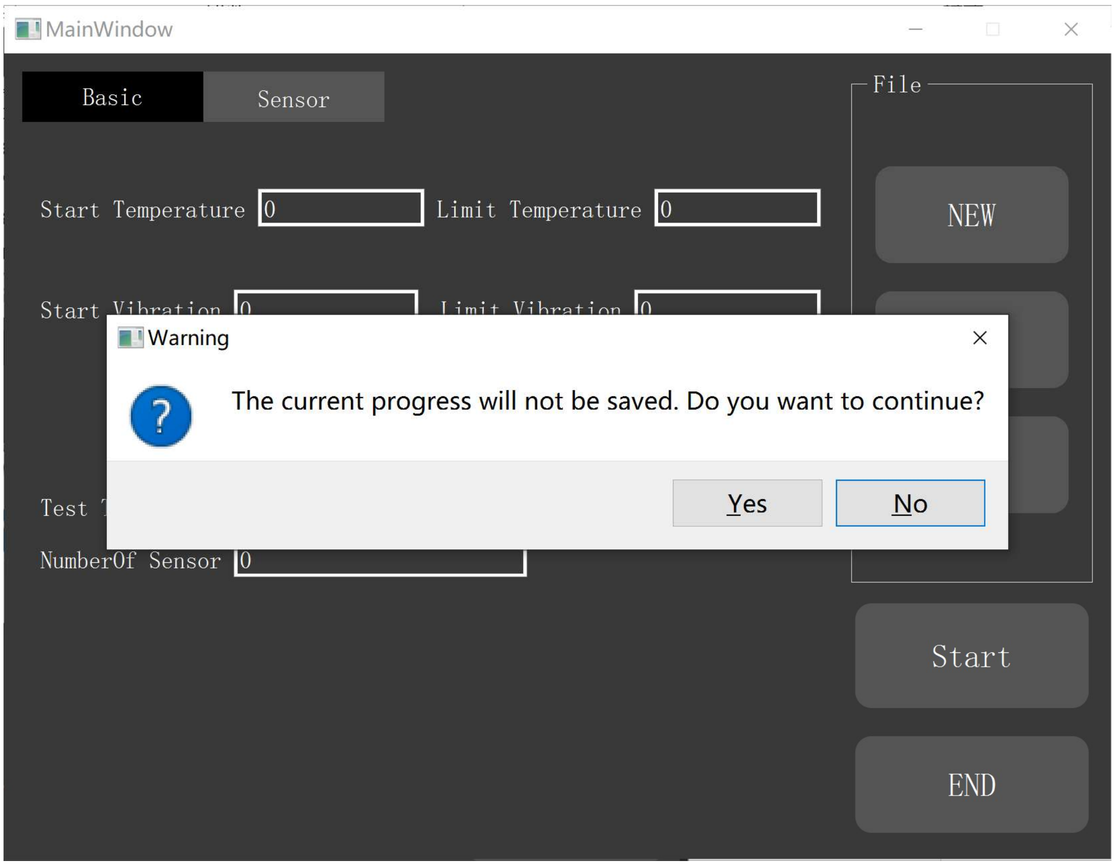
Test Time 0

NumberOf Sensor 0

For configuration parameters, users can perform three operations on the file:

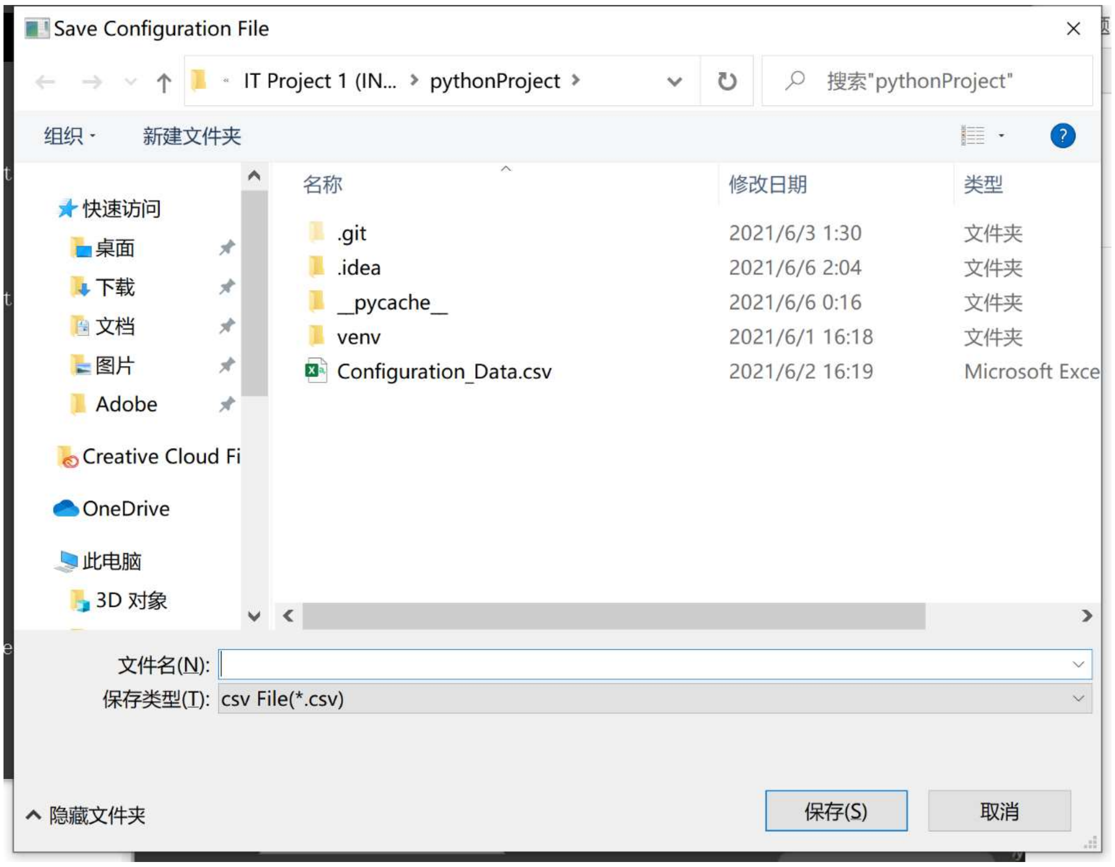
1、 **Create a new DES**

New DES New DES input means to reset all interface parameters. There will be an information box for inquiry. Users should carefully select "Yes" to avoid unsaved processes.



The current progress will not be saved. Do you want to continue?

2、Save current configuration parameters



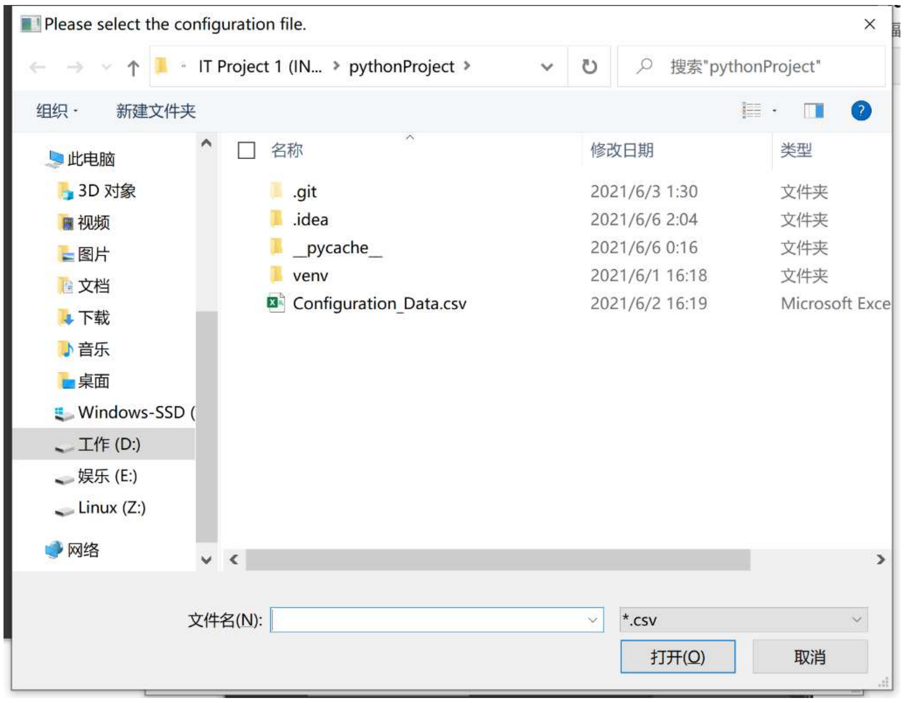
名称	修改日期	类型
.git	2021/6/3 1:30	文件夹
.idea	2021/6/6 2:04	文件夹
__pycache__	2021/6/6 0:16	文件夹
venv	2021/6/1 16:18	文件夹
Configuration_Data.csv	2021/6/2 16:19	Microsoft Exce

文件名(N):
保存类型(T): csv File(*.csv)

Click Save to open a window for saving operation. This operation will generate a configuration parameter csv in a specific format(csv).

3、**Open the configuration file**

After clicking "Open", the user can select the saved configuration file through this window.


Please select the configuration file.

IT Project 1 (IN... › pythonProject ›

搜索"pythonProject"

组织 · 新建文件夹

名称	修改日期	类型
.git	2021/6/3 1:30	文件夹
.idea	2021/6/6 2:04	文件夹
__pycache__	2021/6/6 0:16	文件夹
venv	2021/6/1 16:18	文件夹
Configuration_Data.csv	2021/6/2 16:19	Microsoft Exce

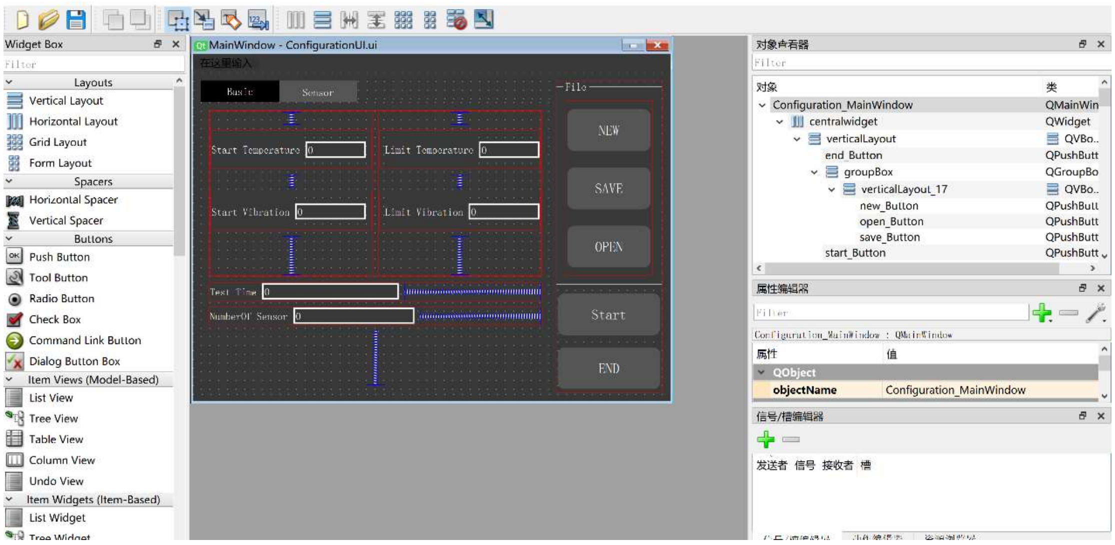
此电脑
3D 对象
视频
图片
文档
下载
音乐
桌面
Windows-SSD (
工作 (D:)
娱乐 (E:)
Linux (Z:)
网络

文件名(N):
*.csv
打开(O) 取消

*Note that the configuration file should be imported in a fixed format, otherwise it cannot be imported.

Code Implementation Components

The front end of this interface is based on QTdesigner for layout and settings.
In this window, the layout mainly involves the following controls:


MainWindow - ConfigurationUI.ui

Widget Box
Layouts: Vertical Layout, Horizontal Layout, Grid Layout, Form Layout
Spacers: Horizontal Spacer, Vertical Spacer
Buttons: Push Button, Tool Button, Radio Button, Check Box, Command Link Button, Dialog Button Box
Item Views (Model-Based): List View, Tree View, Table View, Column View, Undo View
Item Widgets (Item-Based): List Widget, Tree Widget

在这里输入
Basic | Sensor
Start Temperature 0 | Limit Temperature 0
Start Vibration 0 | Limit Vibration 0
Test Time 0
NumberOf Sensor 0
File: NEW, SAVE, OPEN
Start
END

对象查看器

对象	类
Configuration_MainWindow	QMainWin
centralwidget	QWidget
verticalLayout	QVBo..
end_Button	QPushButt
groupBox	QGroupBo
verticalLayout_17	QVBo..
new_Button	QPushButt
open_Button	QPushButt
save_Button	QPushButt
start_Button	QPushButt

属性编辑器
Configuration_MainWindow : QMainWindow

属性	值
QObject	
objectName	Configuration_MainWindow

信号/槽编辑器
发送者 信号 接收者 槽

Layout

Through the built-in layoutStretch parameter of horizontal layout and vertical layout, the designer can assign the proportion of the control in the layout, and the proportion can be adaptive with the change of the window size.

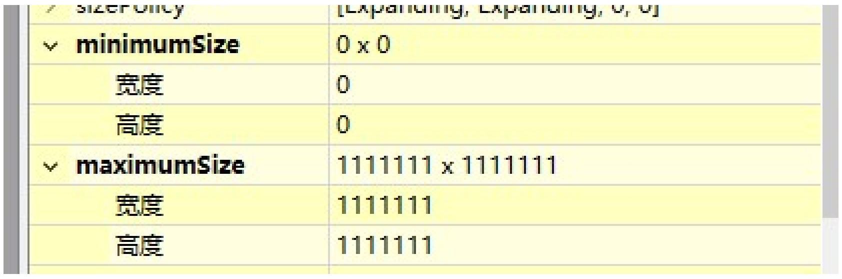
Groupbox, Tagwidget

The containers provide docked classification groups for the control.

The designer can also preset the following parameters in qtdesigner:

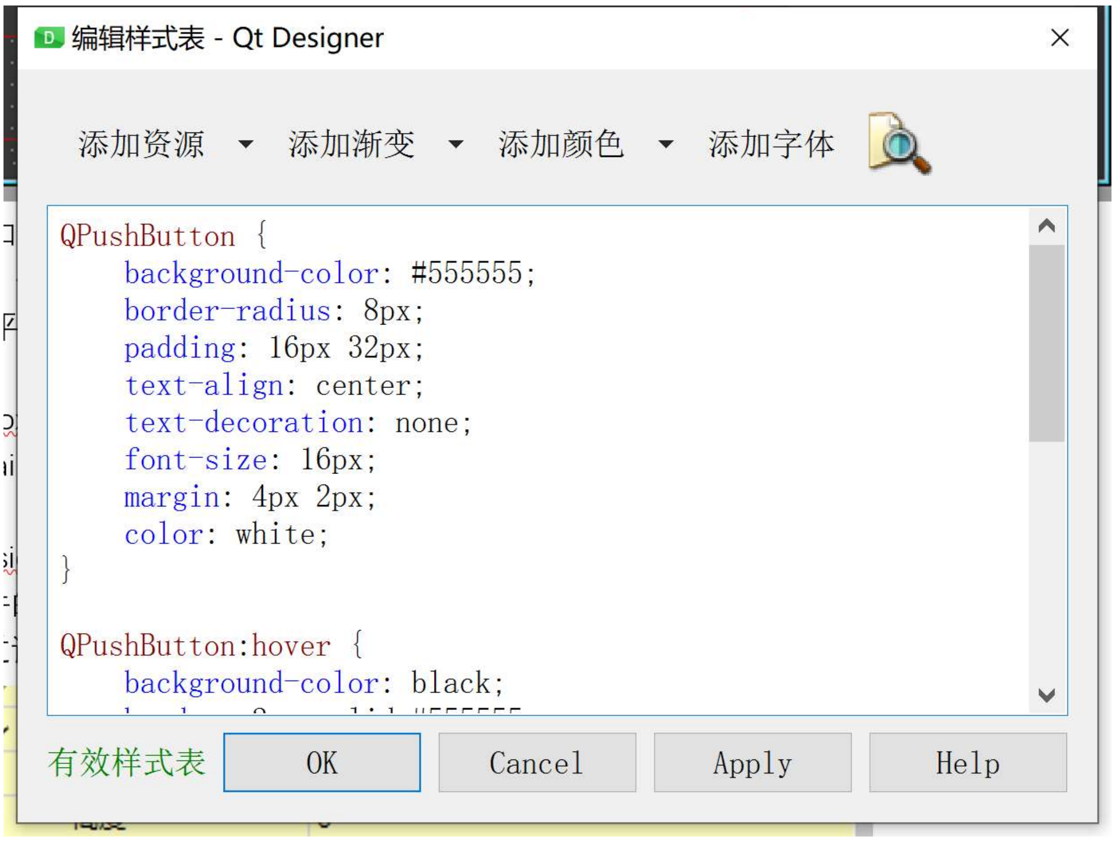
1. The maximum and minimum size of the control

Through this parameter, the designer can set the maximum and minimum size of the window or each control separately.



minimumSize	0 x 0
宽度	0
高度	0
maximumSize	1111111 x 1111111
宽度	1111111
高度	1111111

2. Style sheet



```
QPushButton {
    background-color: #555555;
    border-radius: 8px;
    padding: 16px 32px;
    text-align: center;
    text-decoration: none;
    font-size: 16px;
    margin: 4px 2px;
    color: white;
}

QPushButton:hover {
    background-color: black;
```

Through the style sheet, the designer can change the style of different groups of controls. Including: background color, font, size. The designer can also set the change of the floating style of the button.

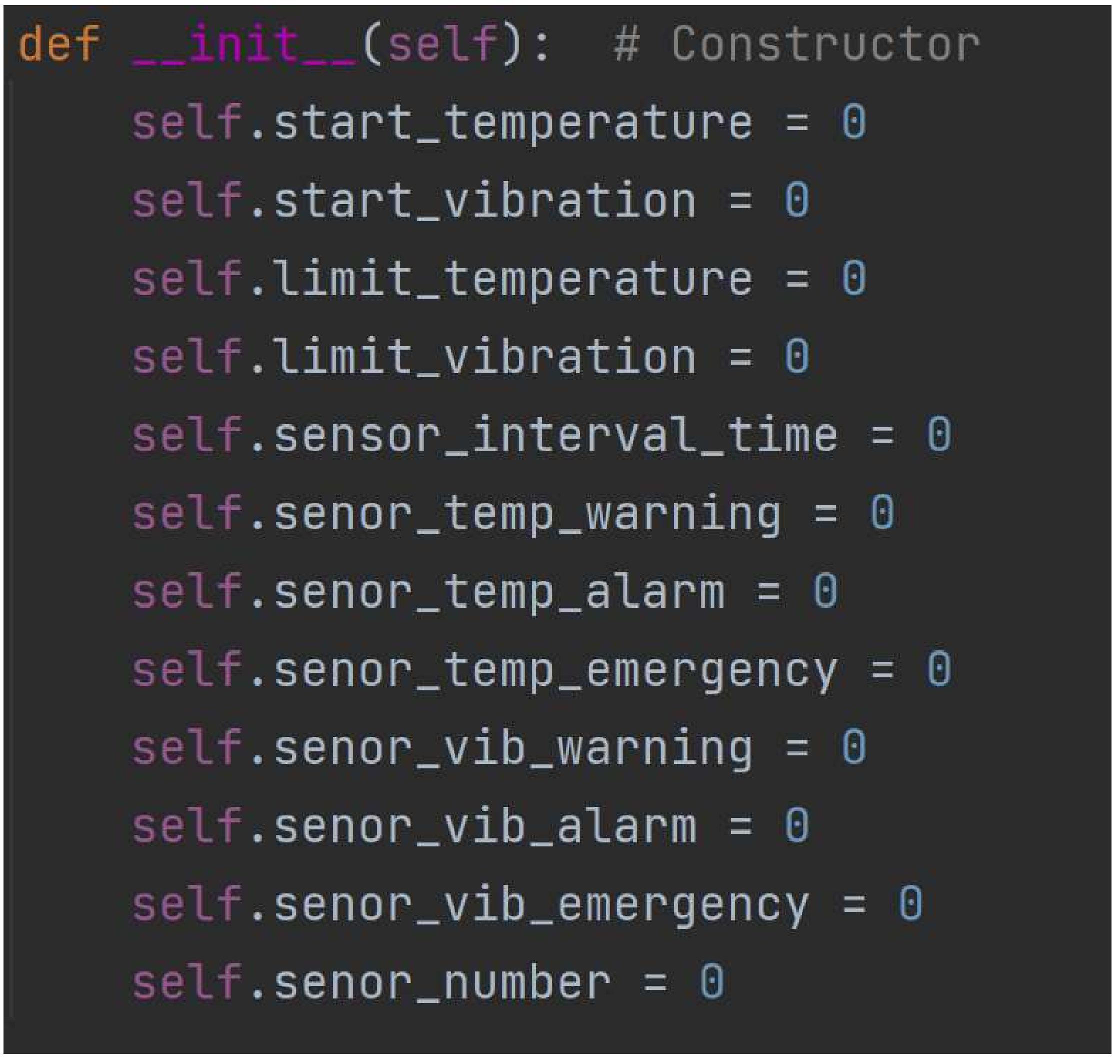
The interface designed in qtdesigner will be saved in .ui format, which is C++ style. The designer can use the external tool PYUIC to convert the ui file into a .py file.
The backend method will be implemented in the py file

Backend code

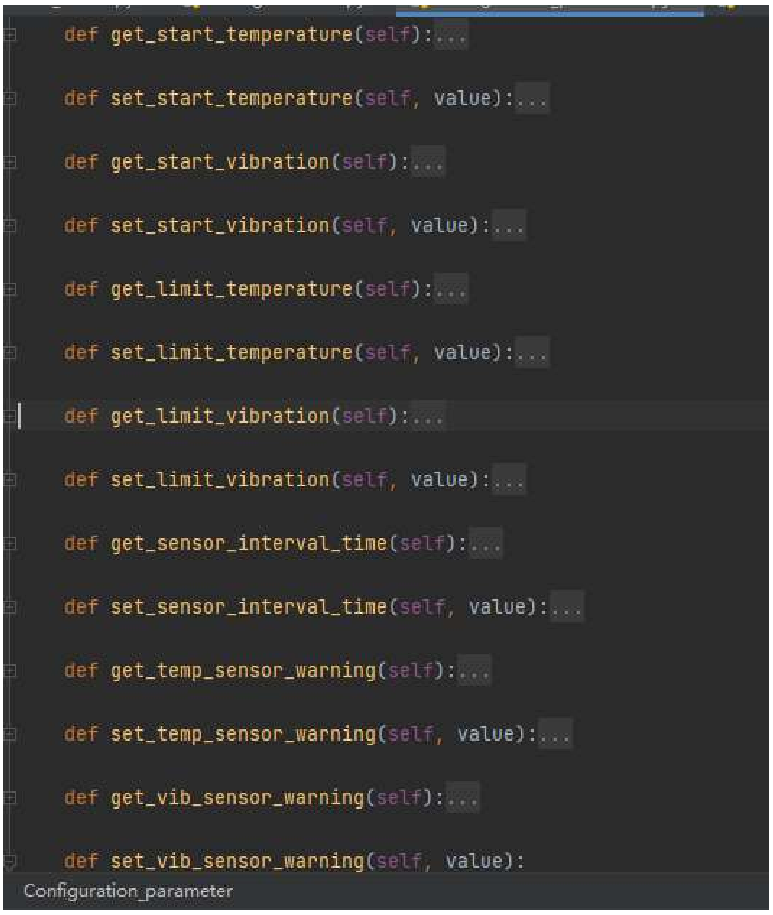
Back-end functions are implemented using code. The file is divided into the window file ConfigurationUI.py and the parameter class Configuration_paramer.py.

Parameter Class

The parameter class is mainly used to store the configuration parameters of the current interface, so that other modules or classes can call and input.



```python
def __init__(self):  # Constructor
    self.start_temperature = 0
    self.start_vibration = 0
    self.limit_temperature = 0
    self.limit_vibration = 0
    self.sensor_interval_time = 0
    self.senor_temp_warning = 0
    self.senor_temp_alarm = 0
    self.senor_temp_emergency = 0
    self.senor_vib_warning = 0
    self.senor_vib_alarm = 0
    self.senor_vib_emergency = 0
    self.senor_number = 0
```

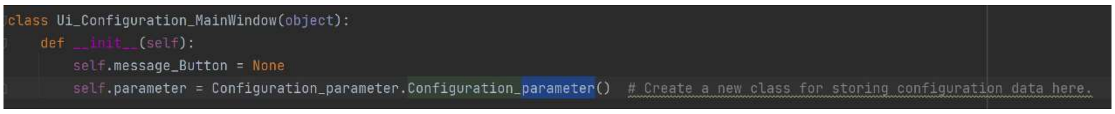


```python
def get_start_temperature(self):...

def set_start_temperature(self, value):...

def get_start_vibration(self):...

def set_start_vibration(self, value):...

def get_limit_temperature(self):...

def set_limit_temperature(self, value):...

def get_limit_vibration(self):...

def set_limit_vibration(self, value):...

def get_sensor_interval_time(self):...

def set_sensor_interval_time(self, value):...

def get_temp_sensor_warning(self):...

def set_temp_sensor_warning(self, value):...

def get_vib_sensor_warning(self):...

def set_vib_sensor_warning(self, value):
```

The methods in the class are the class constructor and the setter and getter of each parameter.

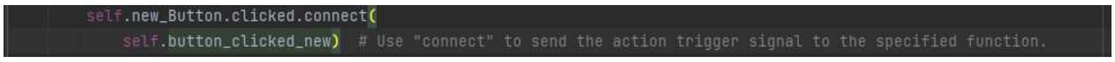
Window Class

The window class is converted from the qtdesigner file .ui.
Import the packages and parameter classes used. Initialize the parameter class and the button of the message box in the constructor. The reason is that the controls used in the message box are not defined in qtdesigner, so they need to be defined in the constructor.



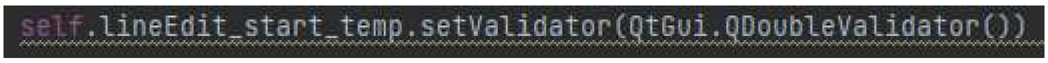
```python
class Ui_Configuration_MainWindow(object):
    def __init__(self):
        self.message_Button = None
        self.parameter = Configuration_parameter.Configuration_parameter()  # Create a new class for storing configuration data here.
```

Use the button method clicked to trigger the connect corresponding method for the button.



```python
        self.new_Button.clicked.connect(
            self.button_clicked_new)  # Use "connect" to send the action trigger signal to the specified function.
```

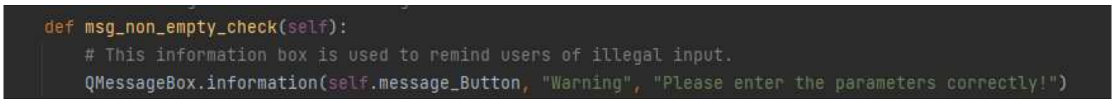
Limit the input type of the input box to avoid entering illegal types (such as: string)



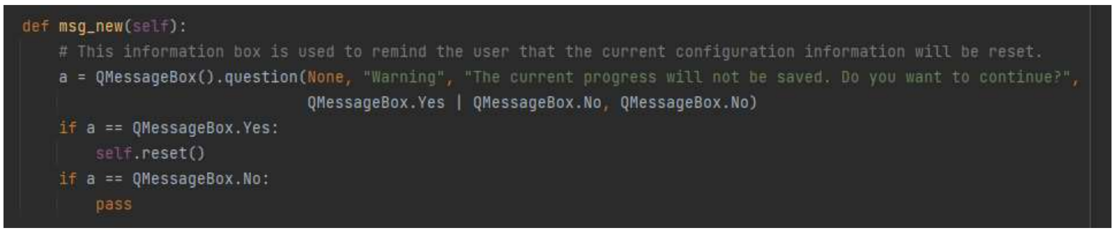
```python
self.lineEdit_start_temp.setValidator(QtGui.QDoubleValidator())
```

Information box method:

The method creates a new information box. It contains buttons for triggering methods and reminder boxes for untriggered methods. Boxes that need a trigger method need to define a trigger button.

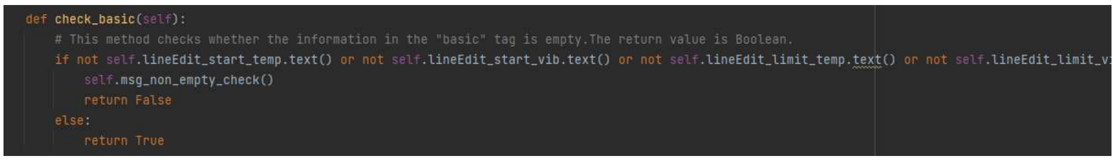


```
def msg_non_empty_check(self):
    # This information box is used to remind users of illegal input.
    QMessageBox.information(self.message_Button, "Warning", "Please enter the parameters correctly!")
```

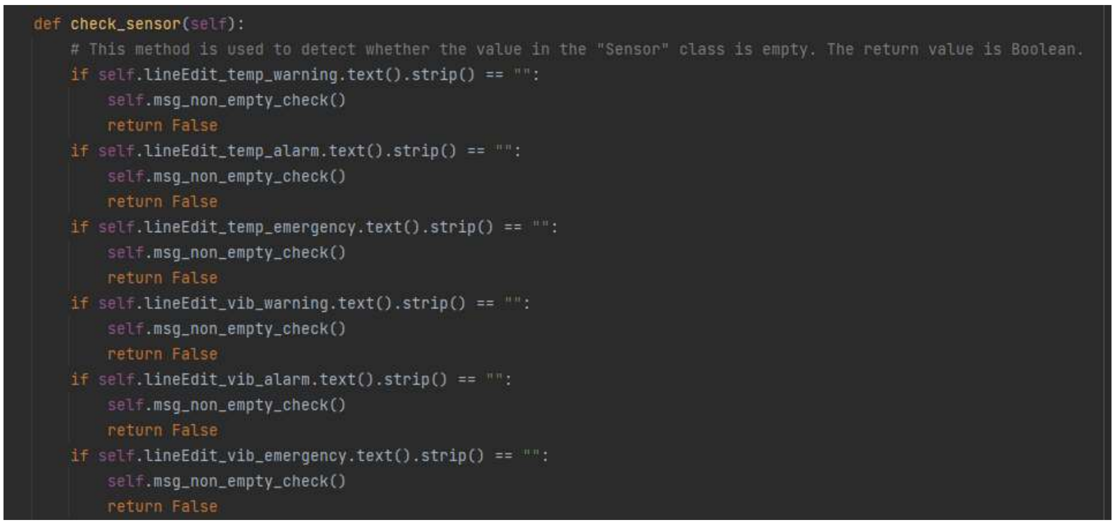


```
def msg_new(self):
    # This information box is used to remind the user that the current configuration information will be reset.
    a = QMessageBox().question(None, "Warning", "The current progress will not be saved. Do you want to continue?",
                               QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
    if a == QMessageBox.Yes:
        self.reset()
    if a == QMessageBox.No:
        pass
```

Check the input box to avoid illegal input values due to empty values in the input box.

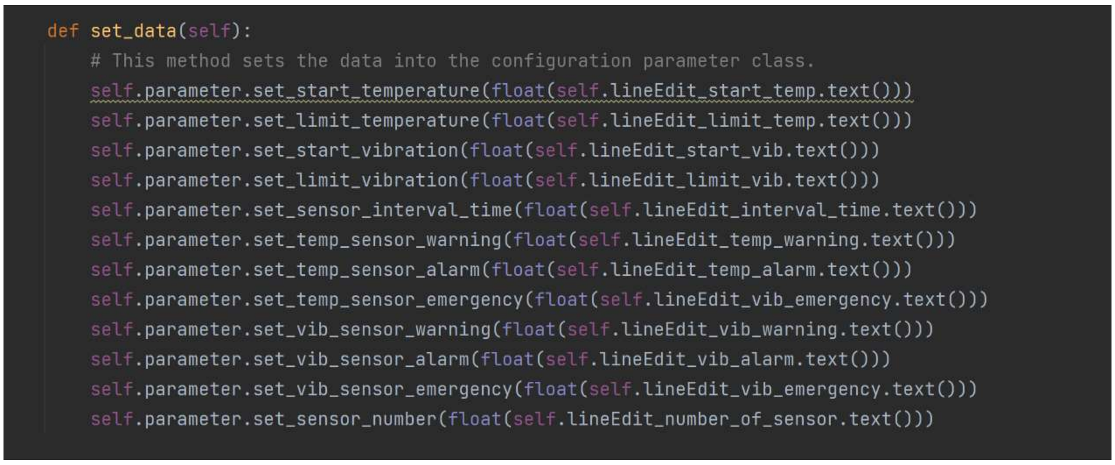


```
def check_basic(self):
    # This method checks whether the information in the "basic" tag is empty.The return value is Boolean.
    if not self.lineEdit_start_temp.text() or not self.lineEdit_start_vib.text() or not self.lineEdit_limit_temp.text() or not self.lineEdit_limit_v
        self.msg_non_empty_check()
        return False
    else:
        return True
```



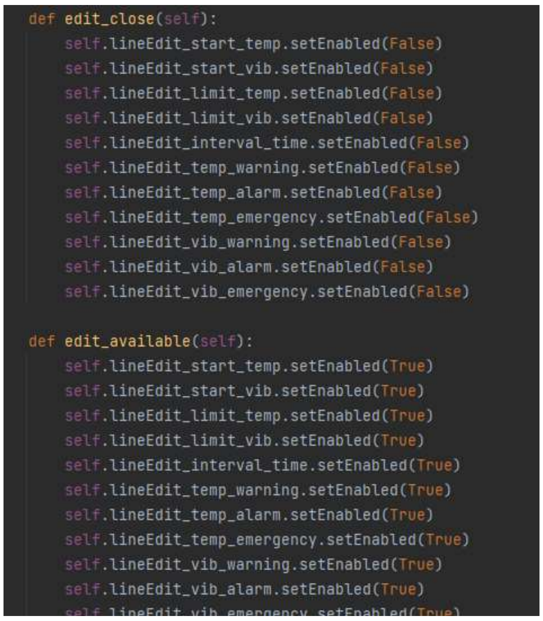
```
def check_sensor(self):
    # This method is used to detect whether the value in the "Sensor" class is empty. The return value is Boolean.
    if self.lineEdit_temp_warning.text().strip() == "":
        self.msg_non_empty_check()
        return False
    if self.lineEdit_temp_alarm.text().strip() == "":
        self.msg_non_empty_check()
        return False
    if self.lineEdit_temp_emergency.text().strip() == "":
        self.msg_non_empty_check()
        return False
    if self.lineEdit_vib_warning.text().strip() == "":
        self.msg_non_empty_check()
        return False
    if self.lineEdit_vib_alarm.text().strip() == "":
        self.msg_non_empty_check()
        return False
    if self.lineEdit_vib_emergency.text().strip() == "":
        self.msg_non_empty_check()
        return False
```

This method converts the characters in the input box to float type and saves it in the parameter class.



```
def set_data(self):
    # This method sets the data into the configuration parameter class.
    self.parameter.set_start_temperature(float(self.lineEdit_start_temp.text()))
    self.parameter.set_limit_temperature(float(self.lineEdit_limit_temp.text()))
    self.parameter.set_start_vibration(float(self.lineEdit_start_vib.text()))
    self.parameter.set_limit_vibration(float(self.lineEdit_limit_vib.text()))
    self.parameter.set_sensor_interval_time(float(self.lineEdit_interval_time.text()))
    self.parameter.set_temp_sensor_warning(float(self.lineEdit_temp_warning.text()))
    self.parameter.set_temp_sensor_alarm(float(self.lineEdit_temp_alarm.text()))
    self.parameter.set_temp_sensor_emergency(float(self.lineEdit_vib_emergency.text()))
    self.parameter.set_vib_sensor_warning(float(self.lineEdit_vib_warning.text()))
    self.parameter.set_vib_sensor_alarm(float(self.lineEdit_vib_alarm.text()))
    self.parameter.set_vib_sensor_emergency(float(self.lineEdit_vib_emergency.text()))
    self.parameter.set_sensor_number(float(self.lineEdit_number_of_sensor.text()))
```

The input permission of the input box can be set through the setEnabled method. By closing the input box, avoid misoperations during the simulation progress that will cause the simulation configuration to change.

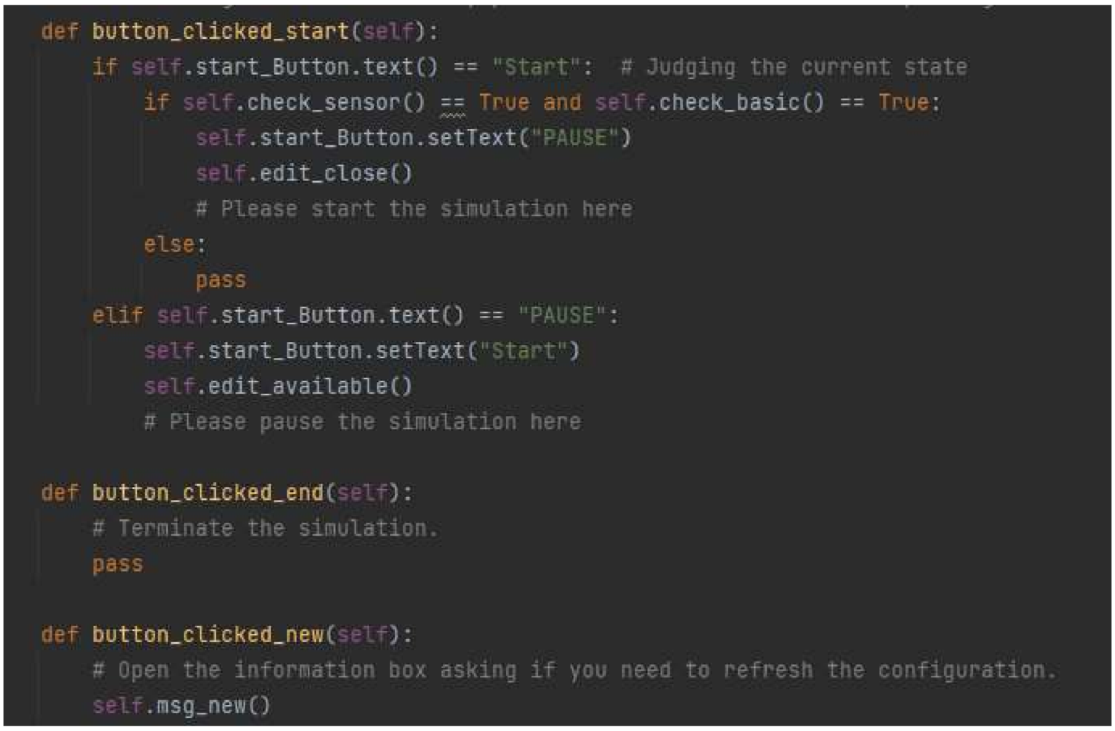


```python
def edit_close(self):
    self.lineEdit_start_temp.setEnabled(False)
    self.lineEdit_start_vib.setEnabled(False)
    self.lineEdit_limit_temp.setEnabled(False)
    self.lineEdit_limit_vib.setEnabled(False)
    self.lineEdit_interval_time.setEnabled(False)
    self.lineEdit_temp_warning.setEnabled(False)
    self.lineEdit_temp_alarm.setEnabled(False)
    self.lineEdit_temp_emergency.setEnabled(False)
    self.lineEdit_vib_warning.setEnabled(False)
    self.lineEdit_vib_alarm.setEnabled(False)
    self.lineEdit_vib_emergency.setEnabled(False)

def edit_available(self):
    self.lineEdit_start_temp.setEnabled(True)
    self.lineEdit_start_vib.setEnabled(True)
    self.lineEdit_limit_temp.setEnabled(True)
    self.lineEdit_limit_vib.setEnabled(True)
    self.lineEdit_interval_time.setEnabled(True)
    self.lineEdit_temp_warning.setEnabled(True)
    self.lineEdit_temp_alarm.setEnabled(True)
    self.lineEdit_temp_emergency.setEnabled(True)
    self.lineEdit_vib_warning.setEnabled(True)
    self.lineEdit_vib_alarm.setEnabled(True)
    self.lineEdit_vib_emergency.setEnabled(True)
```

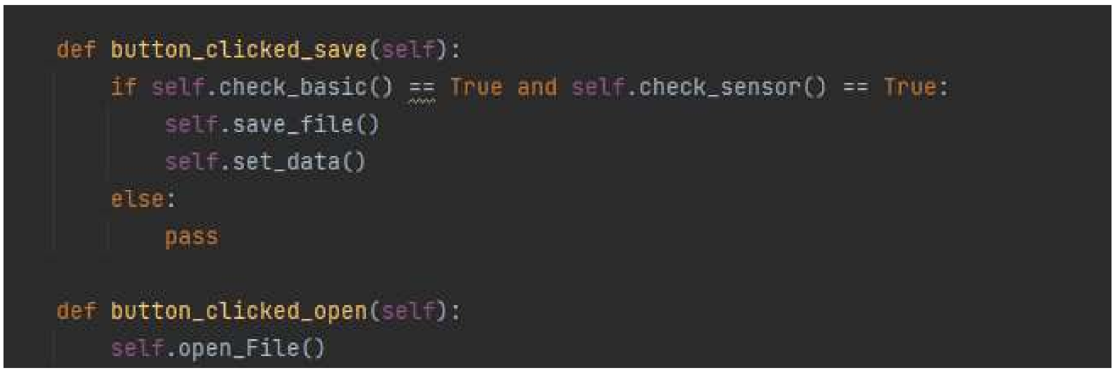
These five methods are all triggered by clicking the button.

Start button method:
This method will judge the current state based on the button text. The input box will be checked in the Start state. Under the premise that the values in the input box are all legal values, the input permission of the input box will be closed. The button text will change to "pause".
Pause:
Since the initial state of the window is "start" when the window is constructed, there is no need to check the state of the input box in this state. After clicking, the input permission of the input box will be opened. The button text will change to "start"

New, save and open will call methods for saving, opening files and resetting configuration parameters. The purpose of this design structure is to separate public methods so that methods can be called from other methods.



```
def button_clicked_start(self):
    if self.start_Button.text() == "Start":  # Judging the current state
        if self.check_sensor() == True and self.check_basic() == True:
            self.start_Button.setText("PAUSE")
            self.edit_close()
            # Please start the simulation here
        else:
            pass
    elif self.start_Button.text() == "PAUSE":
        self.start_Button.setText("Start")
        self.edit_available()
        # Please pause the simulation here

def button_clicked_end(self):
    # Terminate the simulation.
    pass

def button_clicked_new(self):
    # Open the information box asking if you need to refresh the configuration.
    self.msg_new()
```



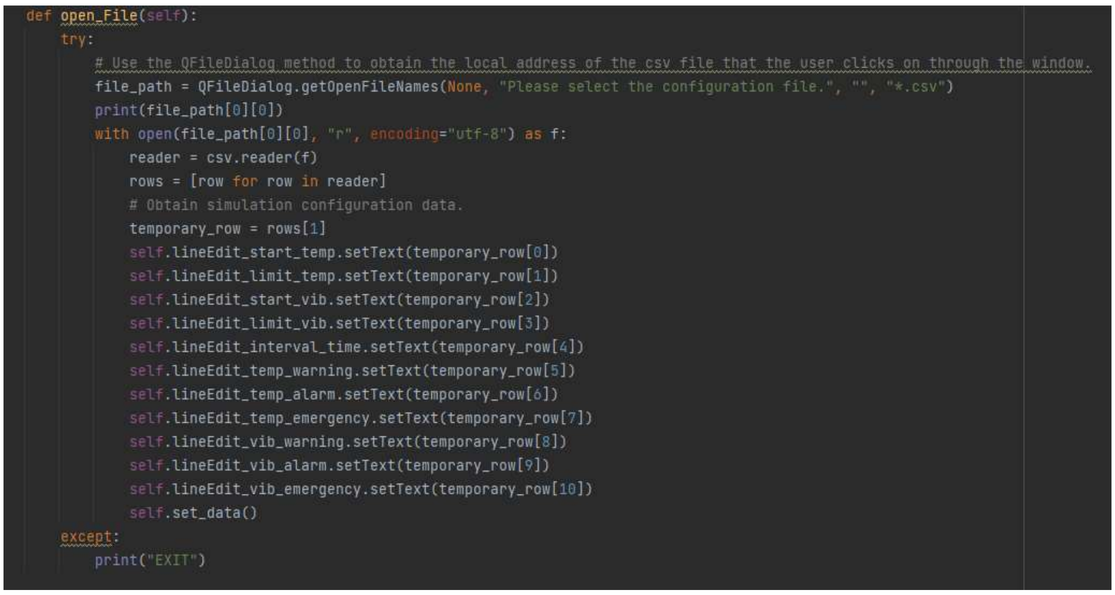
```
def button_clicked_save(self):
    if self.check_basic() == True and self.check_sensor() == True:
        self.save_file()
        self.set_data()
    else:
        pass

def button_clicked_open(self):
    self.open_File()
```

Open file method:

First get the file path through getOpenFileNames in QFileDialog. This method returns the path of the file selected by the user. The parameter "*.csv" restricts the user to only select csv files.

Open the path file through the open method and import it into csv.reader.

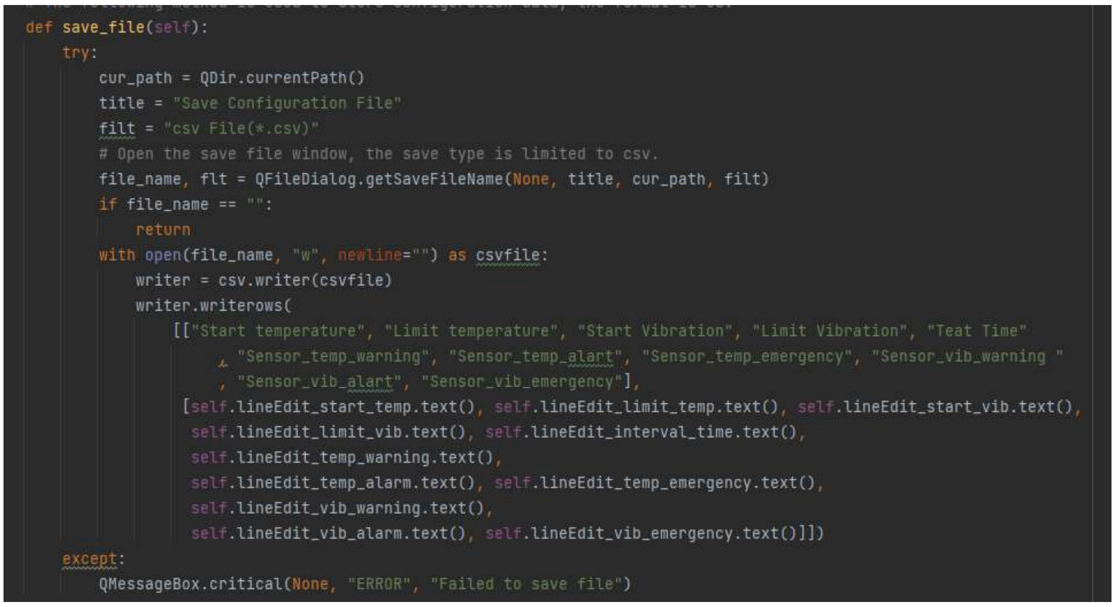
Read the second line of configuration parameters, import them into the input boxes, and import them into the parameter class.



```python
def open_File(self):
    try:
        # Use the QFileDialog method to obtain the local address of the csv file that the user clicks on through the window.
        file_path = QFileDialog.getOpenFileNames(None, "Please select the configuration file.", "", "*.csv")
        print(file_path[0][0])
        with open(file_path[0][0], "r", encoding="utf-8") as f:
            reader = csv.reader(f)
            rows = [row for row in reader]
            # Obtain simulation configuration data.
            temporary_row = rows[1]
            self.lineEdit_start_temp.setText(temporary_row[0])
            self.lineEdit_limit_temp.setText(temporary_row[1])
            self.lineEdit_start_vib.setText(temporary_row[2])
            self.lineEdit_limit_vib.setText(temporary_row[3])
            self.lineEdit_interval_time.setText(temporary_row[4])
            self.lineEdit_temp_warning.setText(temporary_row[5])
            self.lineEdit_temp_alarm.setText(temporary_row[6])
            self.lineEdit_temp_emergency.setText(temporary_row[7])
            self.lineEdit_vib_warning.setText(temporary_row[8])
            self.lineEdit_vib_alarm.setText(temporary_row[9])
            self.lineEdit_vib_emergency.setText(temporary_row[10])
            self.set_data()
    except:
        print("EXIT")
```

Save csv configuration method:

This method uses getSaveFileName to specify the save path and uses the open method to open the file. Import into the csv.writer method. And write to csv according to the format

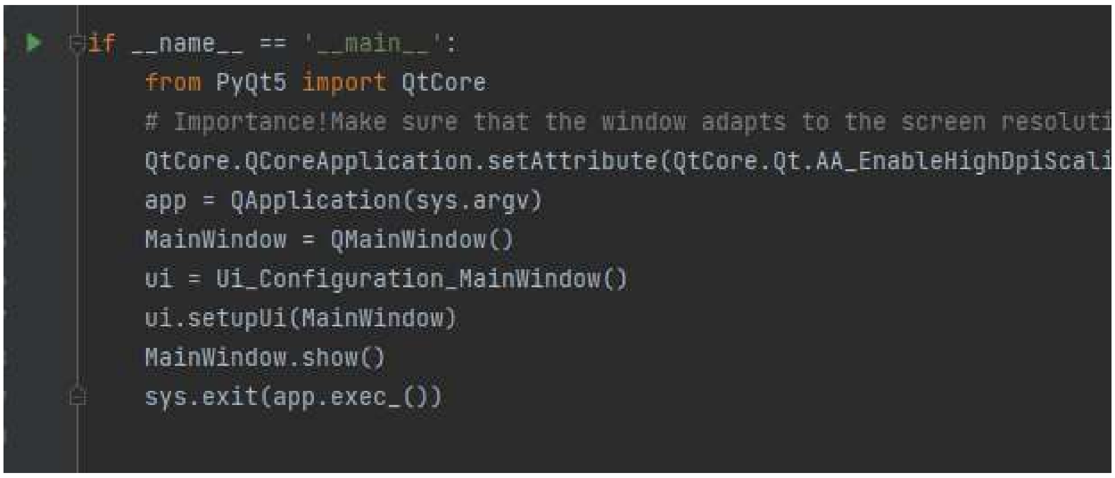


```python
def save_file(self):
    try:
        cur_path = QDir.currentPath()
        title = "Save Configuration File"
        filt = "csv File(*.csv)"
        # Open the save file window, the save type is limited to csv.
        file_name, flt = QFileDialog.getSaveFileName(None, title, cur_path, filt)
        if file_name == "":
            return
        with open(file_name, "w", newline="") as csvfile:
            writer = csv.writer(csvfile)
            writer.writerows(
                [["Start temperature", "Limit temperature", "Start Vibration", "Limit Vibration", "Teat Time"
                     , "Sensor_temp_warning", "Sensor_temp_alart", "Sensor_temp_emergency", "Sensor_vib_warning "
                     , "Sensor_vib_alart", "Sensor_vib_emergency"],
                 [self.lineEdit_start_temp.text(), self.lineEdit_limit_temp.text(), self.lineEdit_start_vib.text(),
                  self.lineEdit_limit_vib.text(), self.lineEdit_interval_time.text(),
                  self.lineEdit_temp_warning.text(),
                  self.lineEdit_temp_alarm.text(), self.lineEdit_temp_emergency.text(),
                  self.lineEdit_vib_warning.text(),
                  self.lineEdit_vib_alarm.text(), self.lineEdit_vib_emergency.text()]])
    except:
        QMessageBox.critical(None, "ERROR", "Failed to save file")
```

The user runs this method to create a new app window and display the window.

The following statement ensures that the window adapts to the resolution of the current display screen and avoids distortion of the control ratio due to different display screens.

```
QtCore.QCoreApplication.setAttribute(QtCore.Qt.AA_EnableHighDpiScalin
g)
```



```
if __name__ == '__main__':
    from PyQt5 import QtCore
    # Importance!Make sure that the window adapts to the screen resoluti
    QtCore.QCoreApplication.setAttribute(QtCore.Qt.AA_EnableHighDpiScali
    app = QApplication(sys.argv)
    MainWindow = QMainWindow()
    ui = Ui_Configuration_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Appendix

This module provides two different connection methods for other modules:

1. The user can use the configuration csv to import it into the DES module.
2. The configurationUI module reserves the control interface and parameter module for the DES module. The DES module can obtain configuration parameters or control signals (such as start and stop) in this module.