

TKinterDesigner tutorial

Author	Honghaier
Ver	1.2.0
Date	2020-03-25

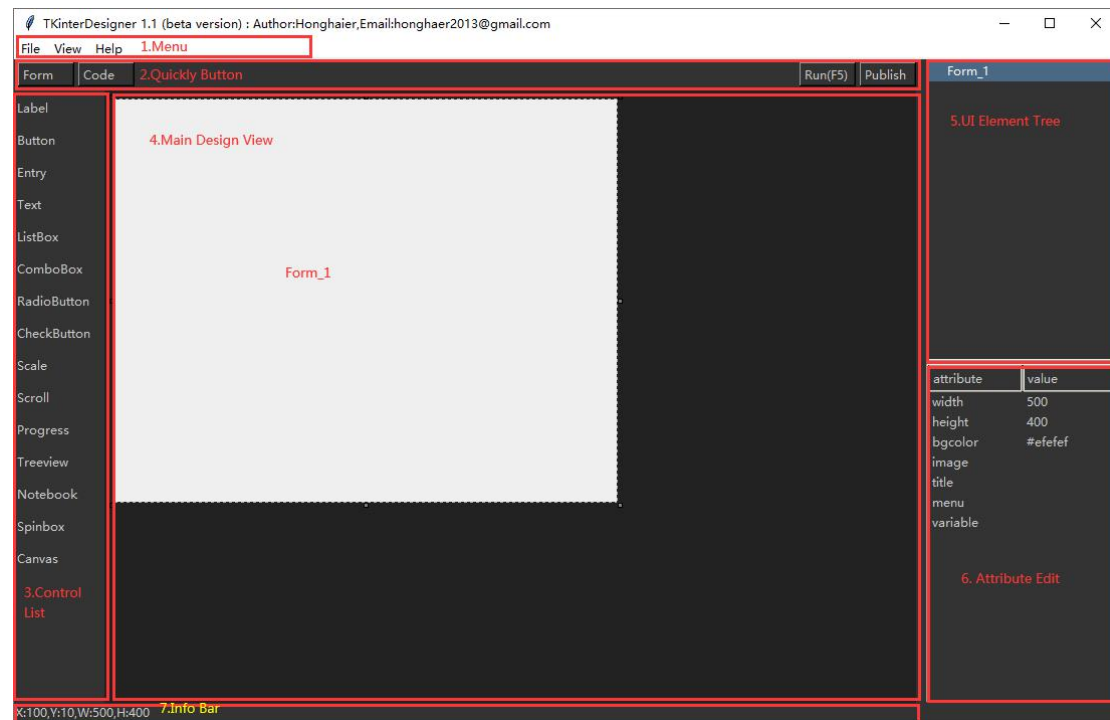
Hello everyone, I am Honghaier. A programmer come from China, I hope to help everyone improve the development efficiency in Python development. So I made a Tool software, It's named "TKinterDesigner", here are some simple instructions. If you have any questions or suggestions, you can send me an email: honghaier2013@gmail.com. I will continue to improve and release updated versions and tutorials. This software is completely free. If you can use it, you can recommend him to your friends and colleagues.

What is "TKinterDesigner"?

TKinterDesigner is a small Python-based TKinter interface editor for software interface design and development when prototyping small and fast Python functions.

Instructions

一. Interface description



The editor consists of seven areas:

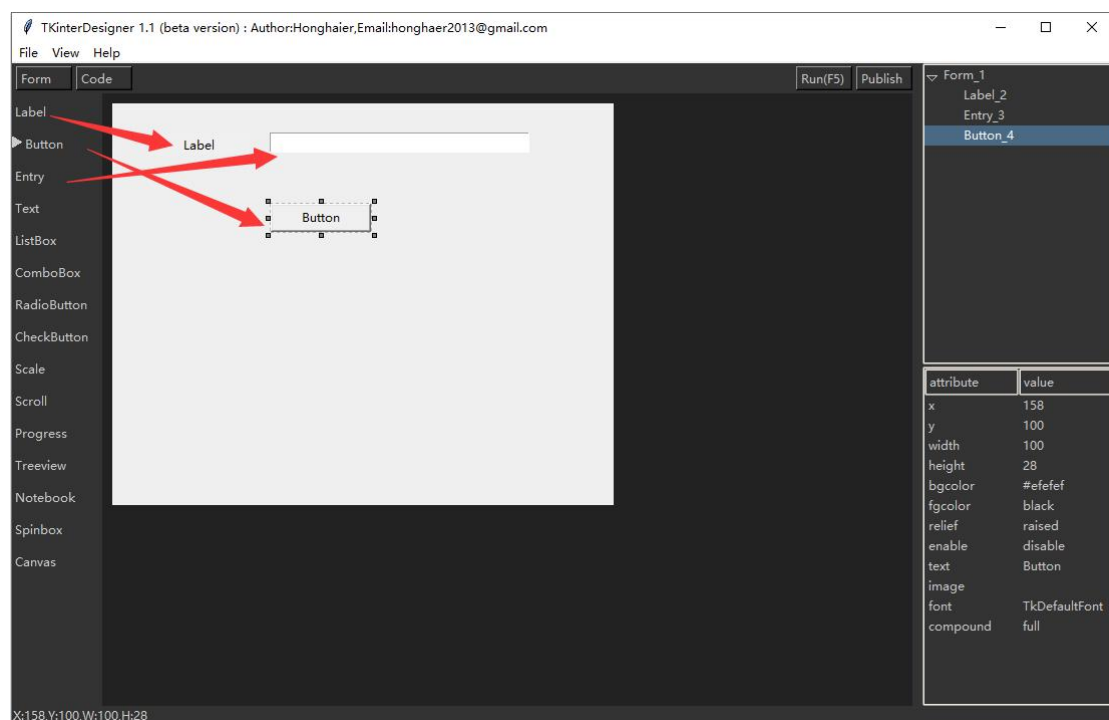
- 1. Main menu:** perform basic file, view, help and other menu item operations.
- 2. Shortcut button:** Quickly switch the form and code view, and view the running status of the interface and publish the EXE.
- 3. List of controls:** commonly used interface controls, used to drag to the interface design area.
- 4. Interface design area:** The main view area for the interface.
- 5. Interface control tree:** a list of all controls in the current interface for quick access and viewing.
- 6. Interface property area:** Display and modify the properties of the corresponding control.
- 7. Bottom information area:** display the position and size of the corresponding control.

二. Features description

1. Interface design

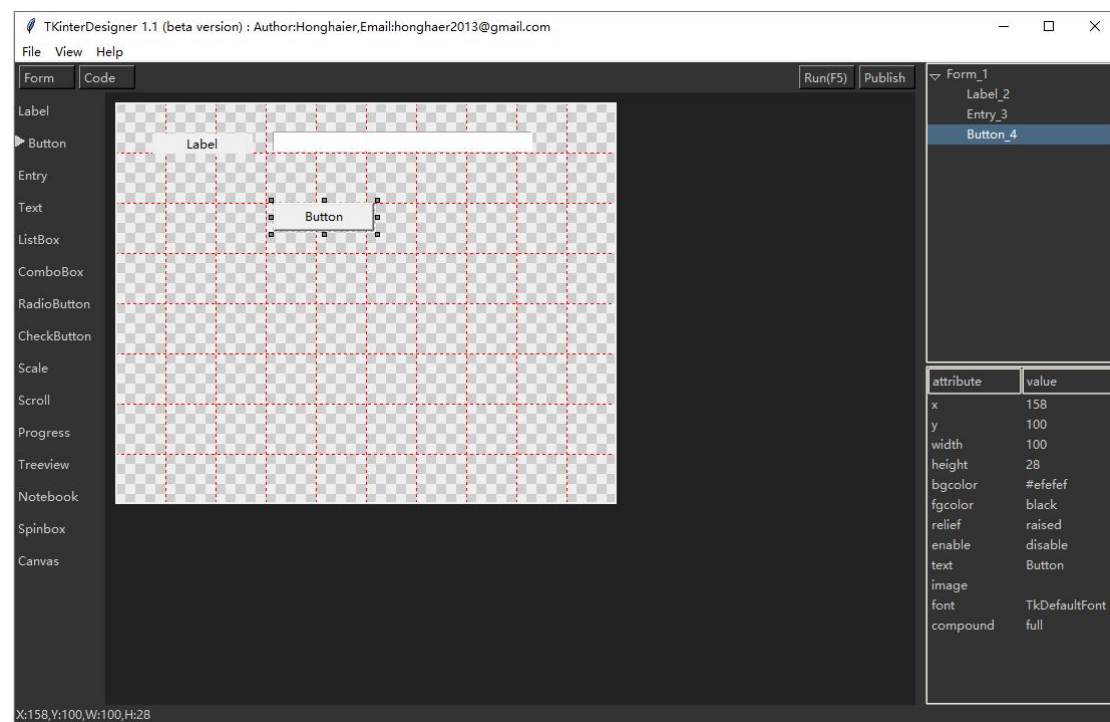
(1) Interface control creation

Use the left mouse button in the control list area to select the corresponding control and drag it to Form in the interface design area to complete the basic control creation. Then you can use the mouse to drag the control to the corresponding position.



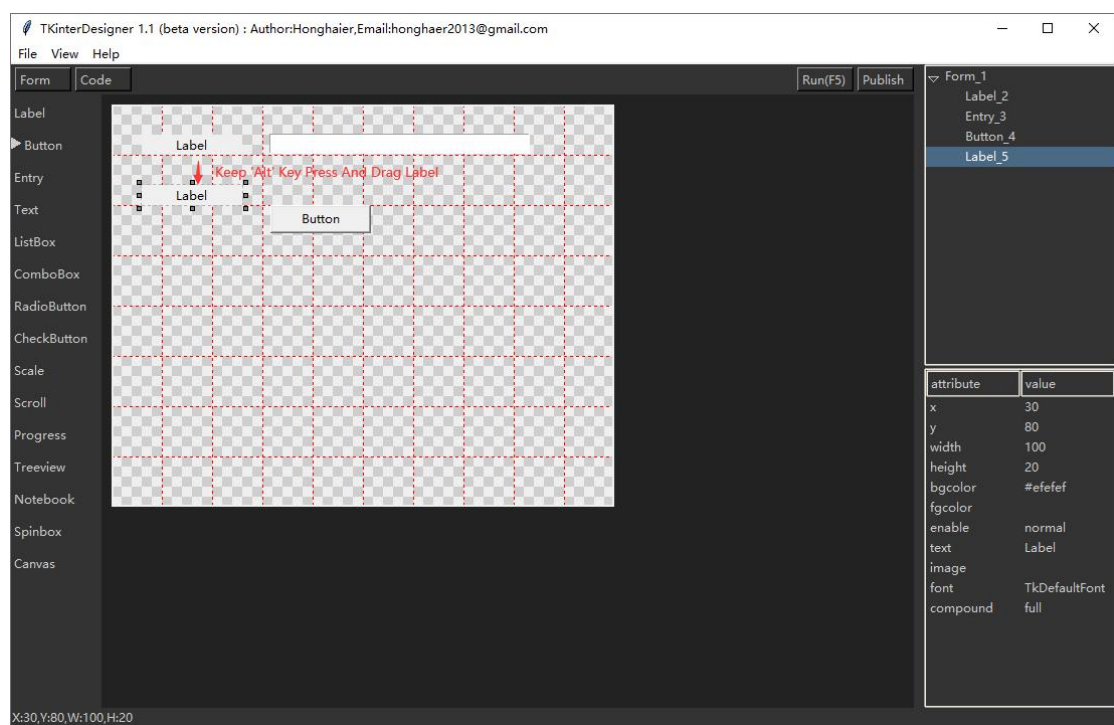
(2) Interface control placement

If you need to place a simple interface, first you can find "Grid" under the "View" item in the main menu. You can also use Ctrl+G to quickly open or close the grid. This will allow you to make basic size observations. Then, you can find "Adsorption" under the "View" item in the main menu, or use Ctrl+D to quickly turn adsorption on or off. After turning on the "grid" and "adsorption", you can drag and drop controls directly to quickly place and align them.

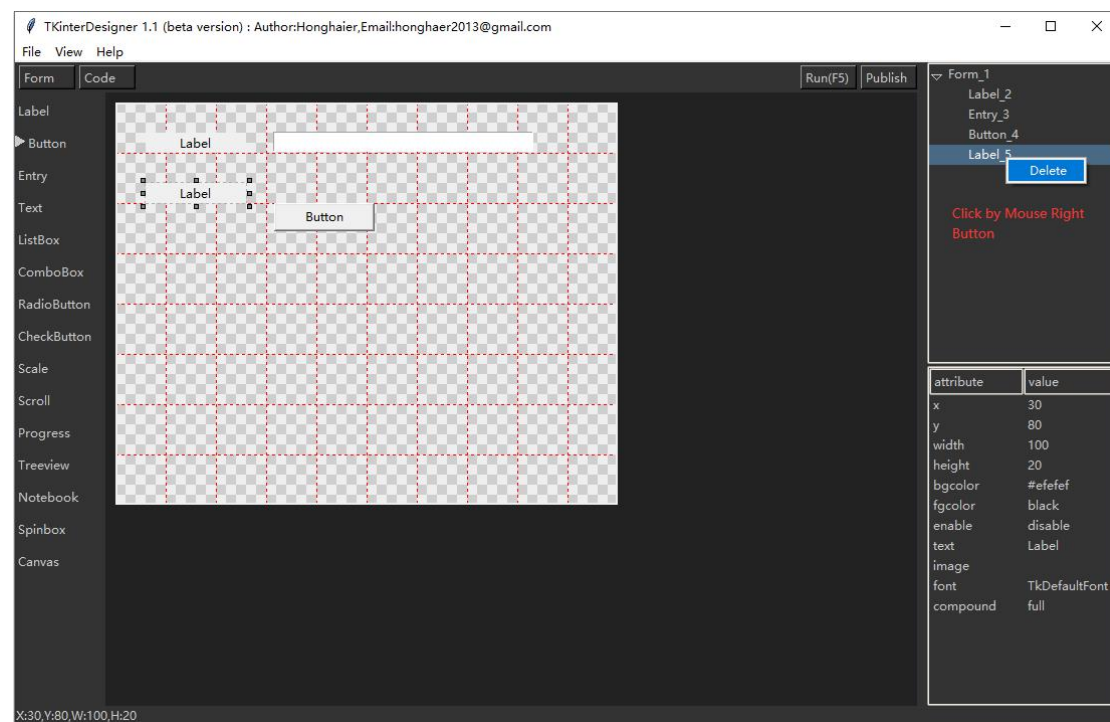


(3) Control copy and delete

Copy control: After selecting a control, hold down the Alt key and drag the control with the mouse to copy a new control.



Deleting a control: You can delete it by using the shortcut key delete or right-clicking a control in the control tree in the upper right corner, and in the pop-up menu. .

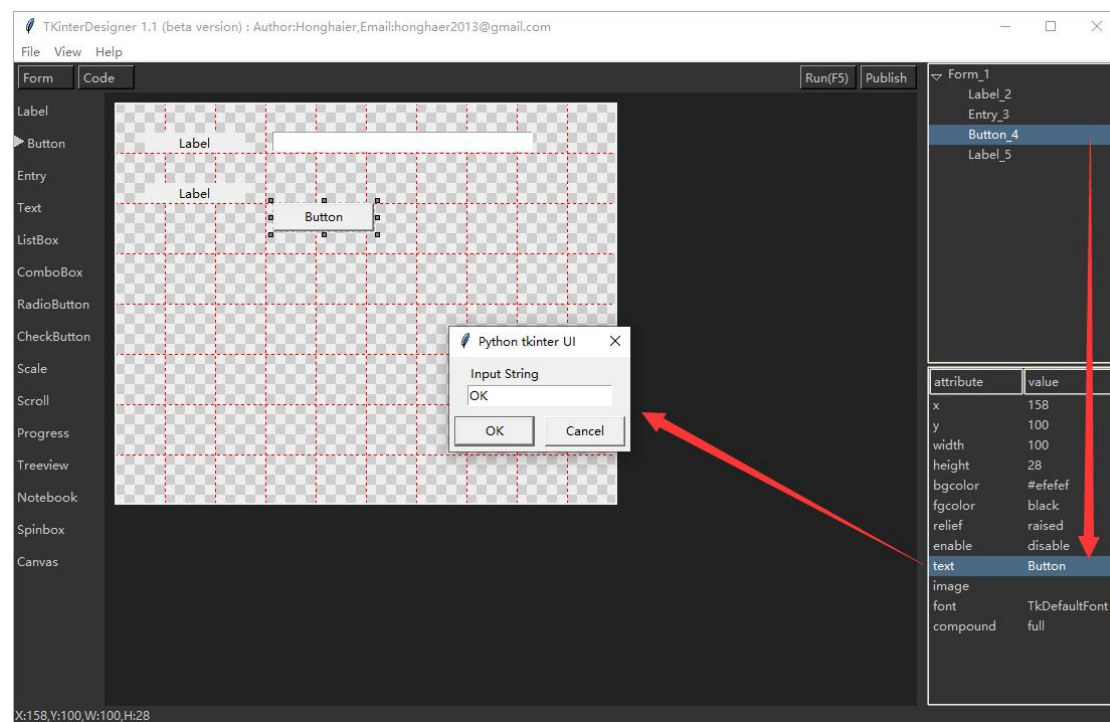


(4) Control property editing

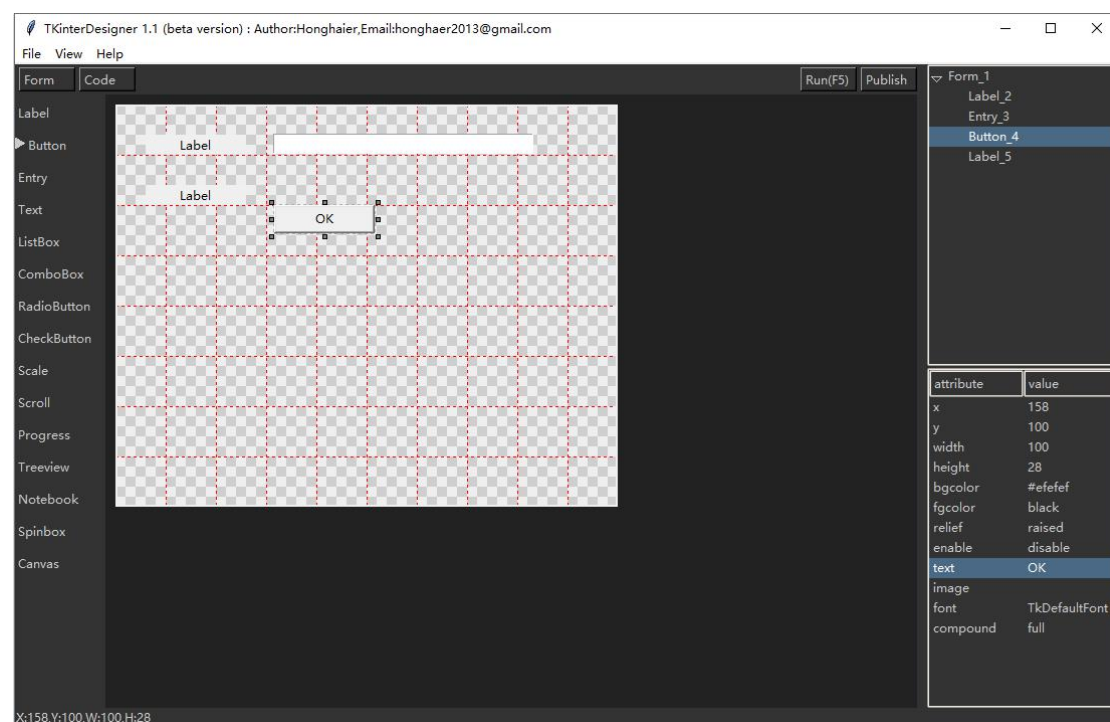
After selecting the corresponding control, in the control list tree in the upper right corner, the corresponding tree item of the corresponding control will be highlighted. At this time, we can modify the corresponding item in the attribute editing area in the lower right corner.

I take the control button as an example, and take a screenshot to show how to modify the text. When we double-click the property "Text" item, a dialog box will pop up, prompting to enter new text.

TKinterDesigner tutorial



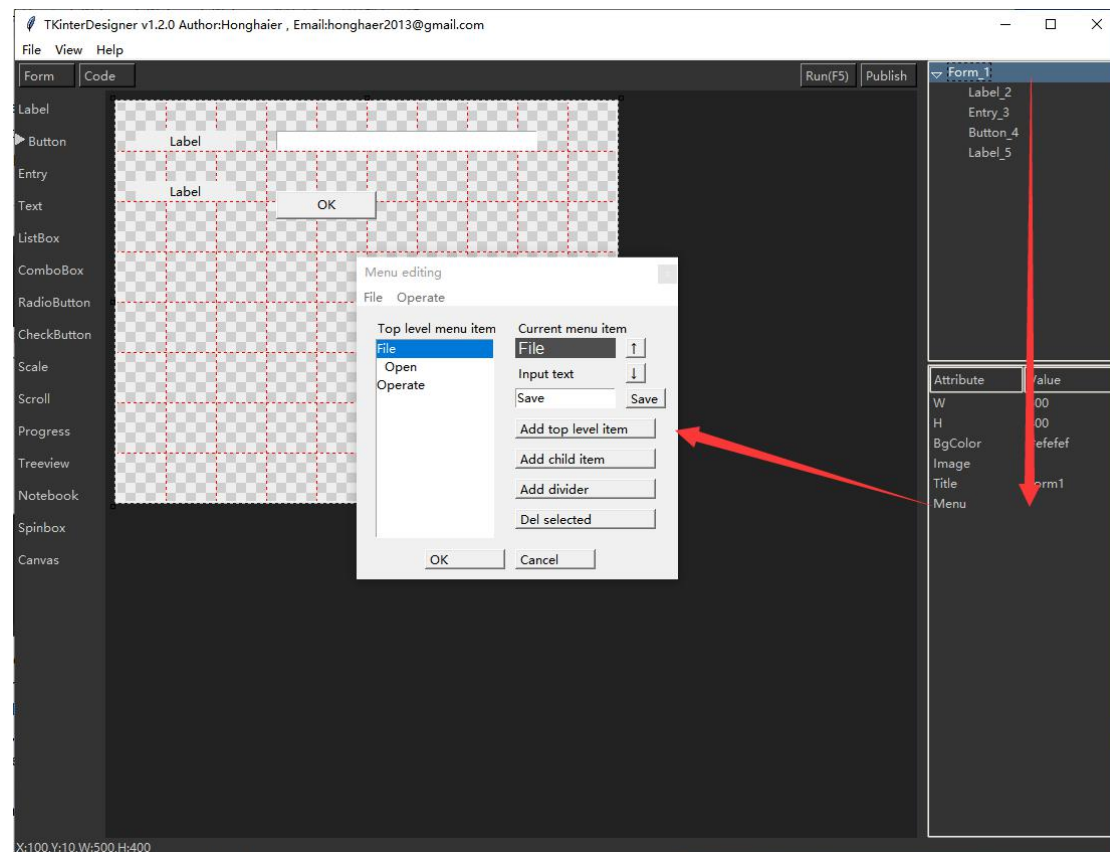
After inputting, click "OK", you can see that the corresponding control is changed to the currently entered text.



Regarding other attributes, we will not introduce them one by one here, and you can try it yourself.

(5) Menu editing

If you need to add a menu to the program, you can select "Form_1" in the control tree list in the upper right corner. At this time, you can see that there are "menu" items in its properties. Double-click the menu edit dialog box to pop up.

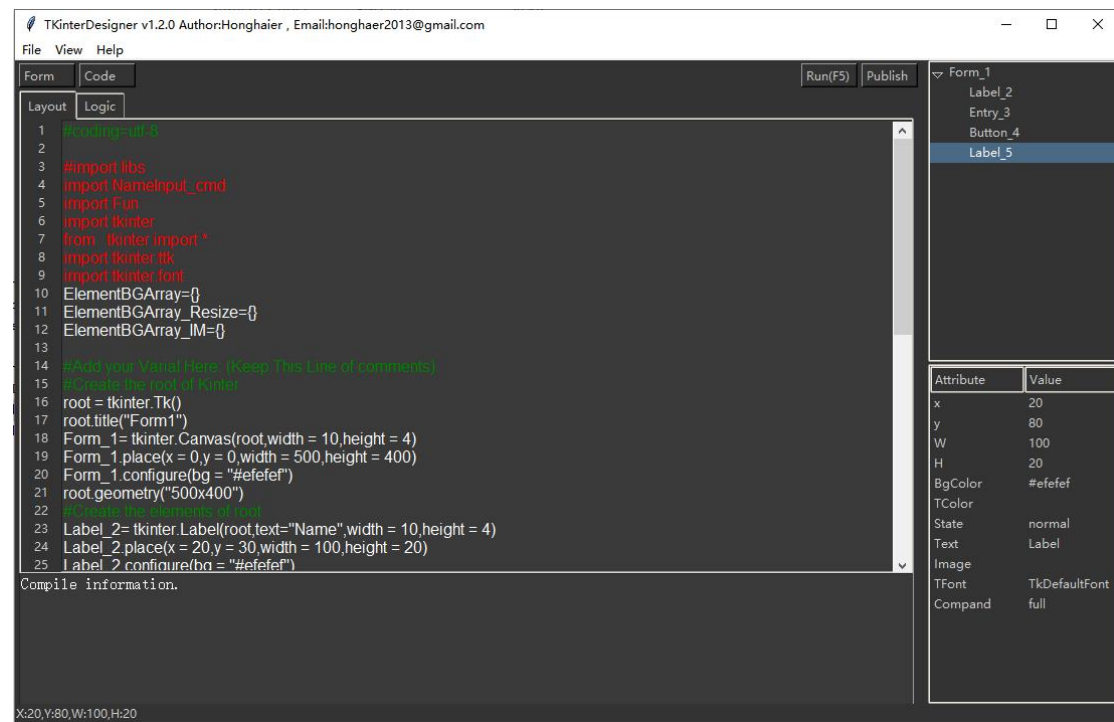


For example, In the menu editing dialog, we first enter "File" in the edit box for entering text, and then click "Add top level item". At this time, we can see that the "File" item appears in the list box of "Top menu items" on the left. , And added the menu and "File" menu items in the dialog box, it can facilitate you to preview the menu, yes, this is just a preview. After we add the top-level menu, click the corresponding menu item in the "Top-level menu item" list on the left, and you can add new sub-items to it by clicking the "Add child item" button on the right. And use the up and down arrow buttons to modify the order of menu items at the same level.

The processing of this part is slightly complicated, but I try to make the operation buttons simple and clear, I hope you can understand.

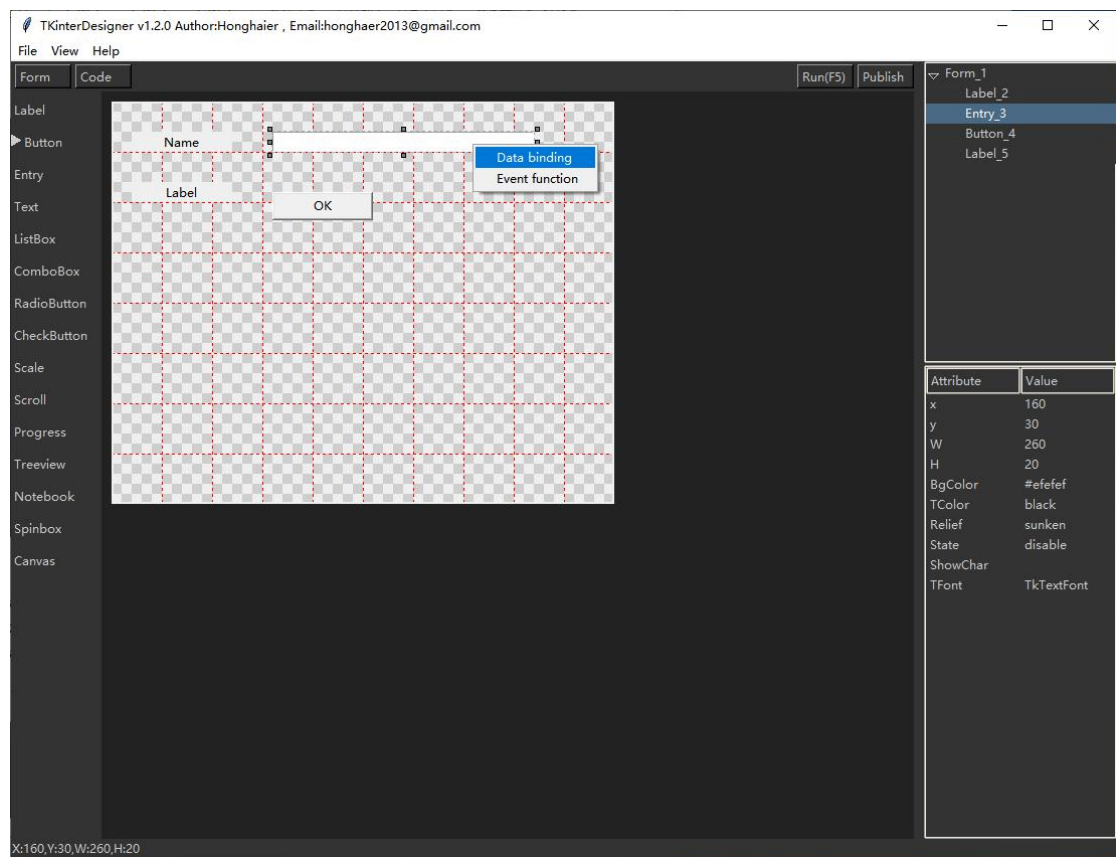
2. Form and code

If you want to view the code, you can click "Code" in the shortcut button area. At this time, we can see the interface code display. Although it cannot be edited directly at the moment, it can make you understand the code.



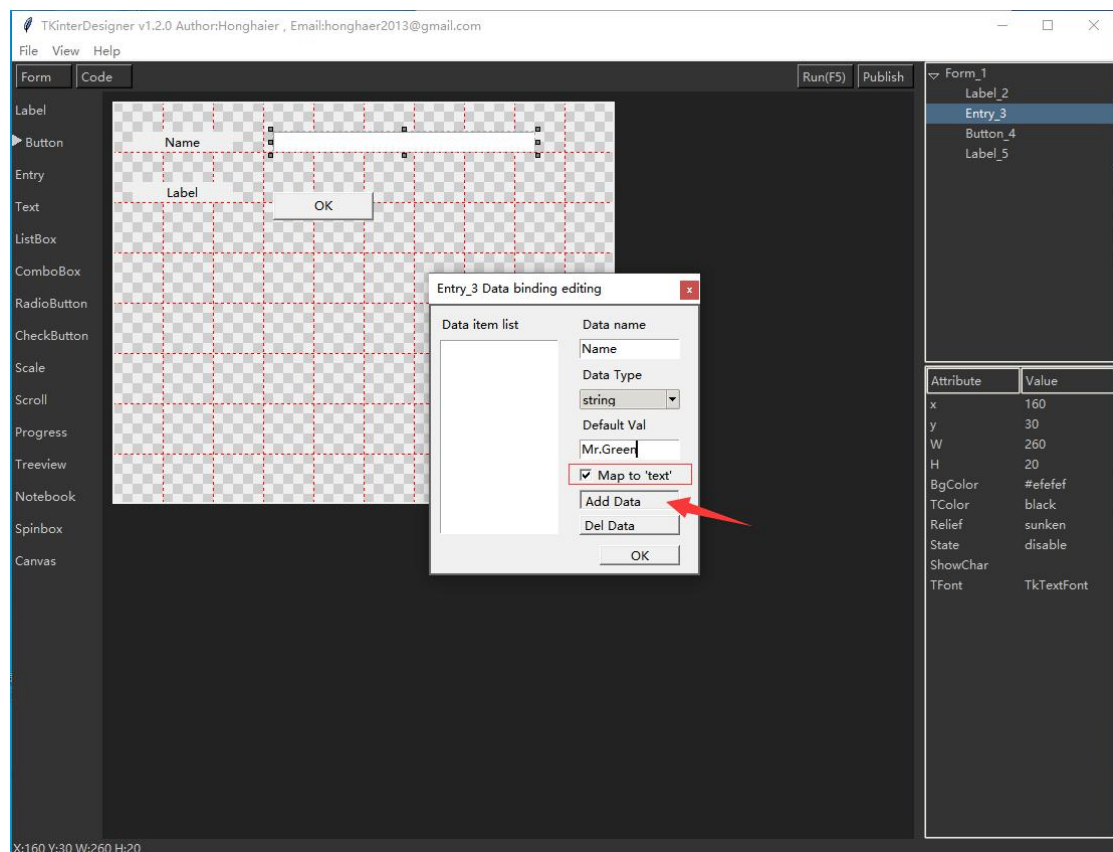
3. Data binding for controls

We can bind a dat to the controls in the interface, and access and modify it. We change the text of Label_2 to "Name" in the property box, and then we want to modify the text in the input box when we click the Button. We can right click on the edit box and select "Data binding" on the popup menu.

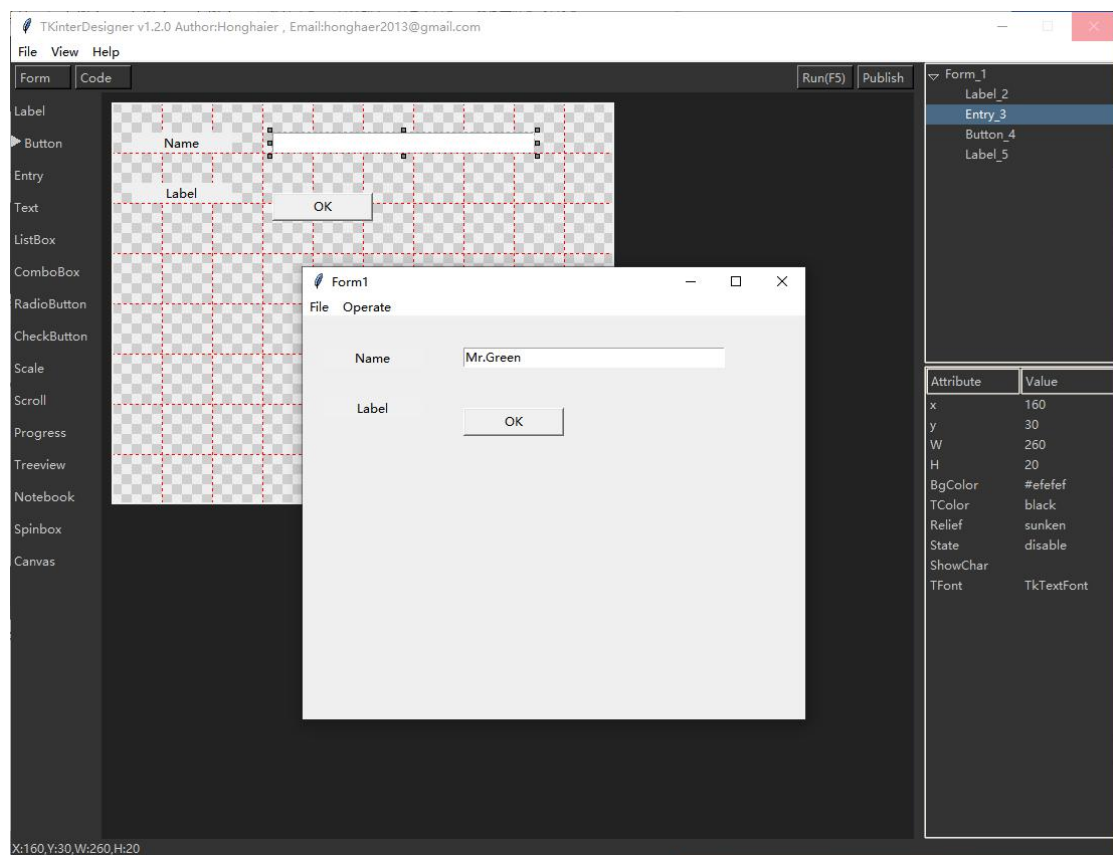


In the pop-up dialog box, add a binding data Name for Entry_3, select "String" for the data item type, enter "Mr.Green" for the default value of the data, and select "Map to 'text'". This option represents whether the data is directly synchronized to Entry_3's 'text' attribute or 'textvariable', click "Add Data ", you can add a string of data for Entry_3.

TKinterDesigner tutorial



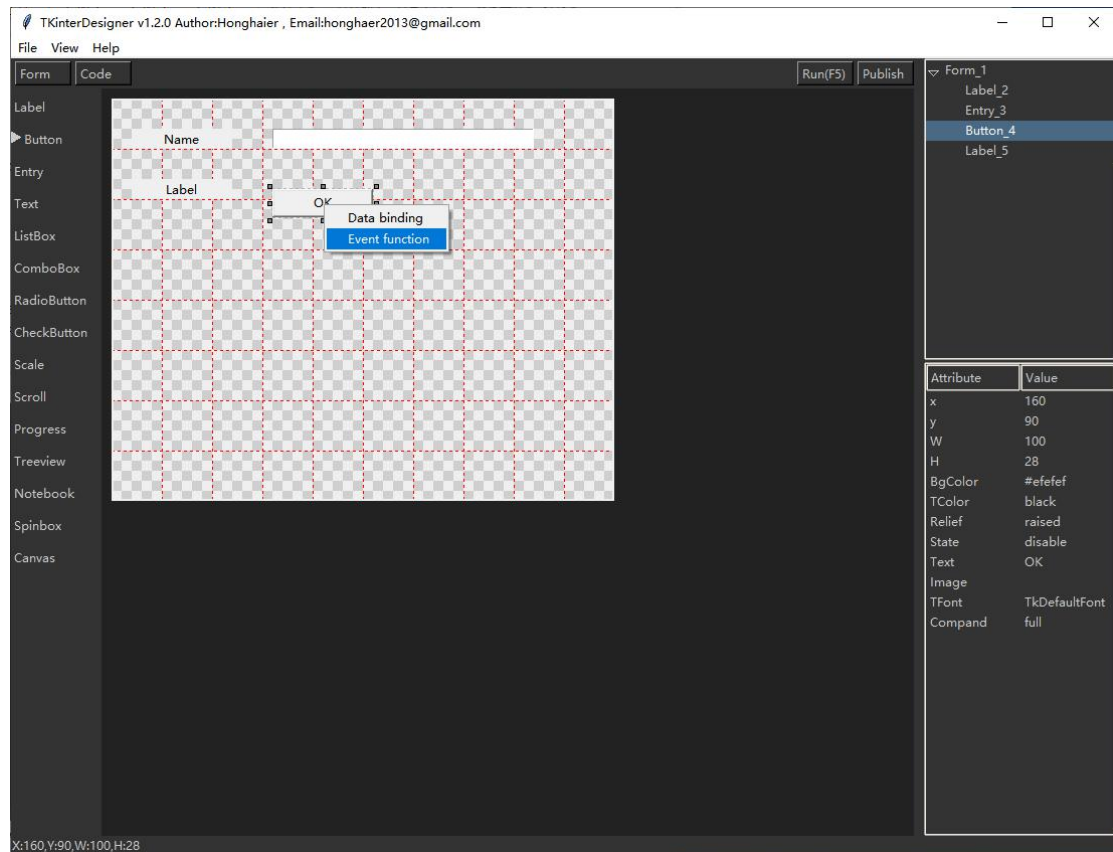
We try to run it by pressing F5. You can see that the Label in the window is displayed as "Mr.Green".



We can add multiple binding variables to a control and access them through the functions “setUIData” and “getUIData”.

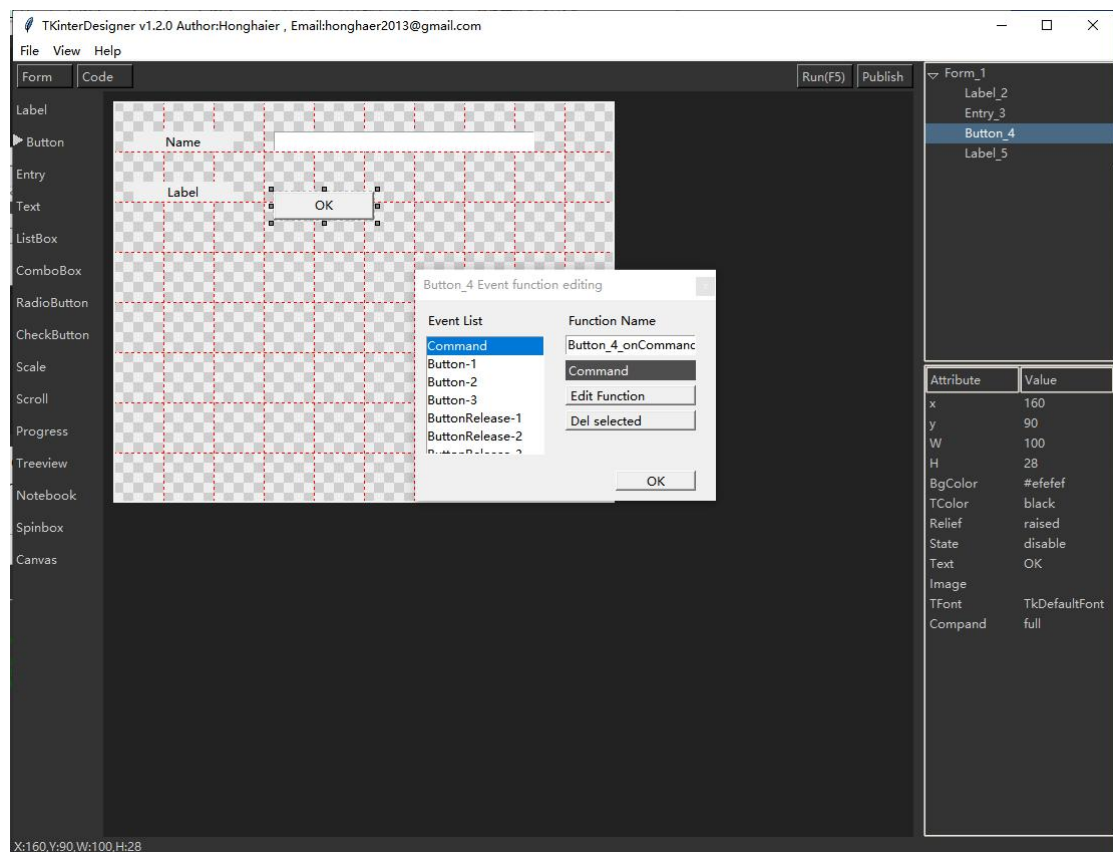
4. Control's event processing

I have added a wizard for mapping event functions for each control. For the example above, we can right-click on the button and select "Event Function" from the popup menu.

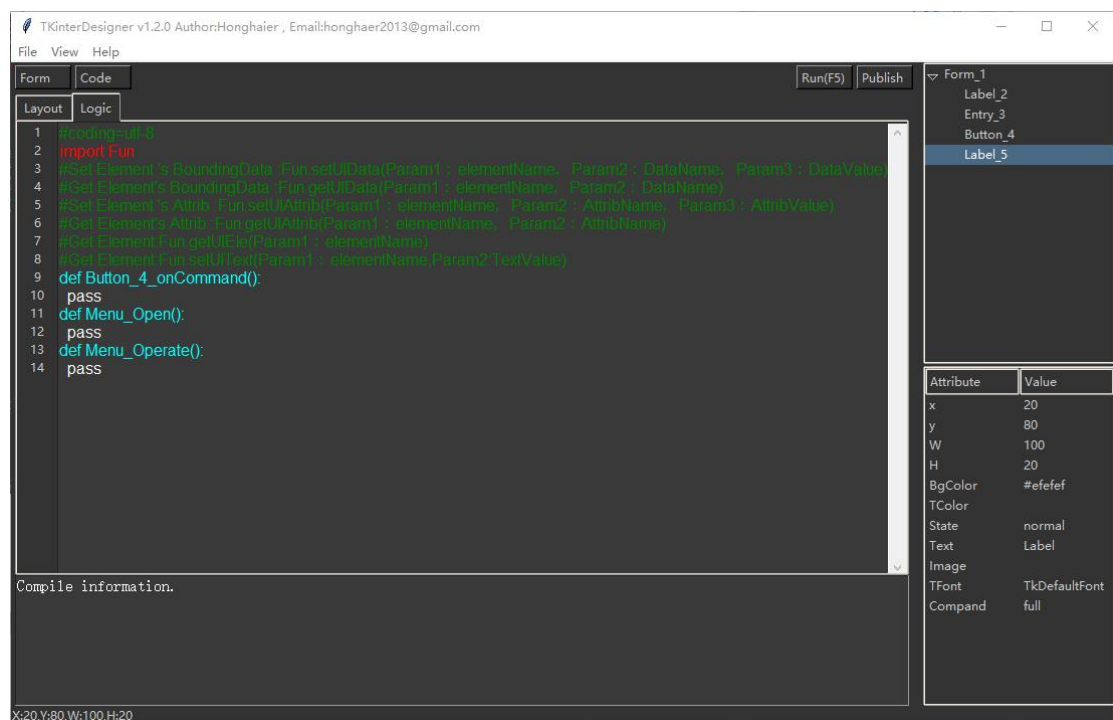


In the pop-up event function editing area, we can see the list of events involved in related controls.

TKinterDesigner tutorial

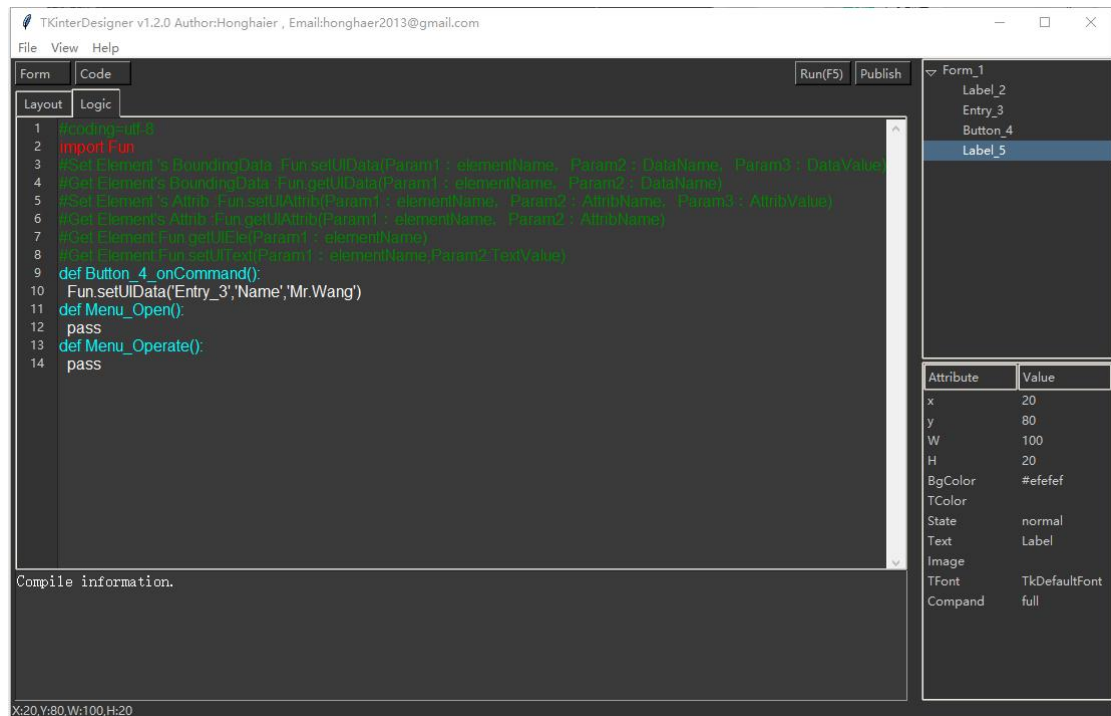


Double-click the function in the control event list or click "Edit Function " to map an event.
Here we edit the button click event, which is Command, to enter the code editing area:

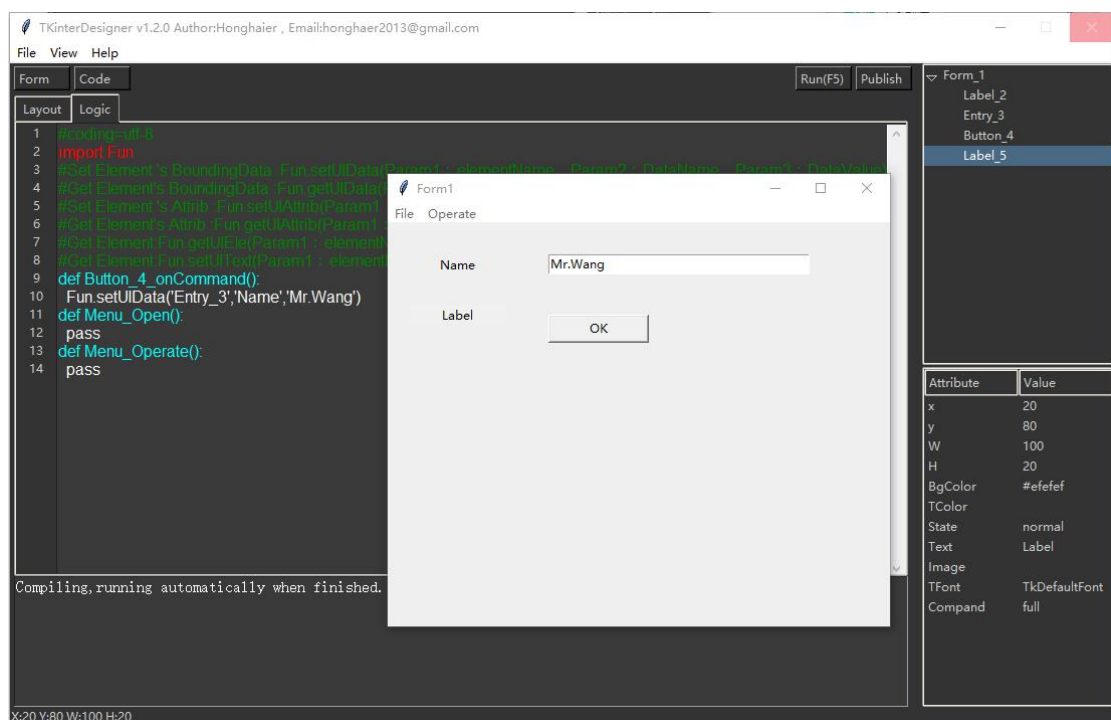


You can modify the code, such as this:

TKinterDesigner tutorial



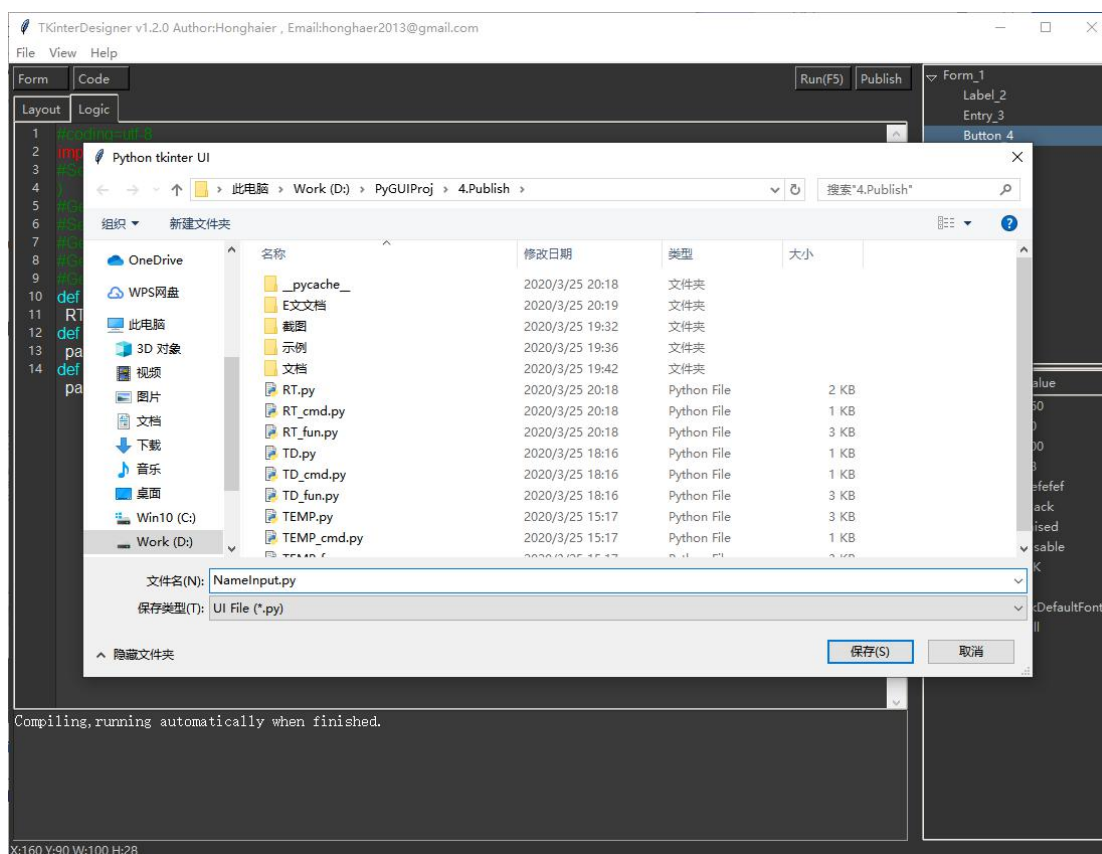
Then, you can click "Run" or press F5, you can run in the visible state, click the button, then you will find that the text of Label_2 becomes 'happy sheep'.



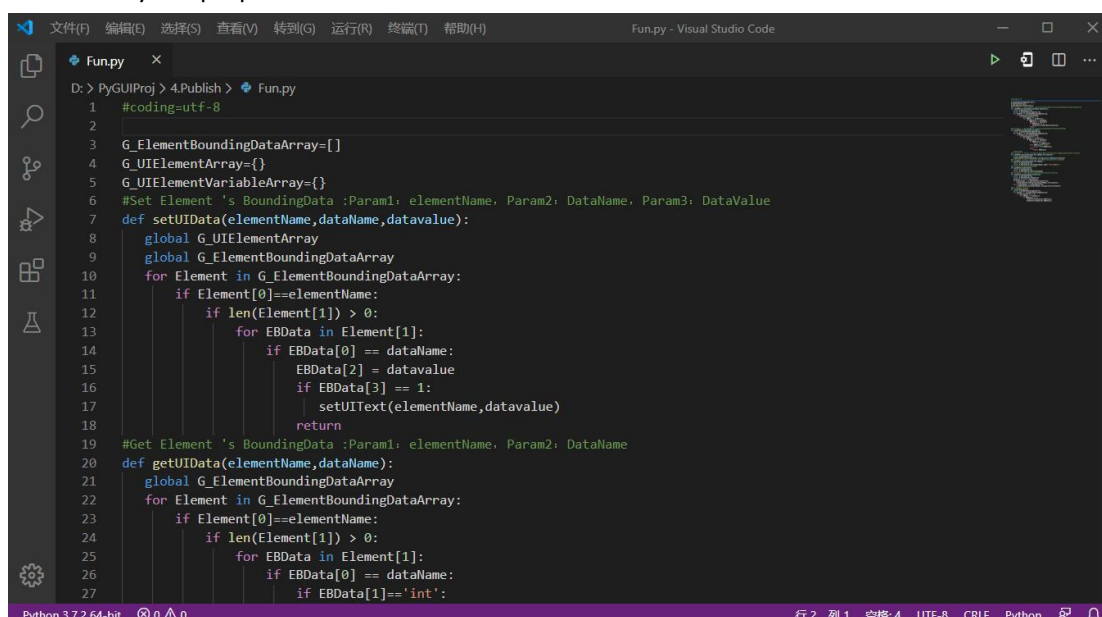
5. Saving and opening files

OK, everything is done, we just need to find "Save" under the "File" menu item in the main menu, or Ctrl + S to save the file.

TKinterDesigner tutorial

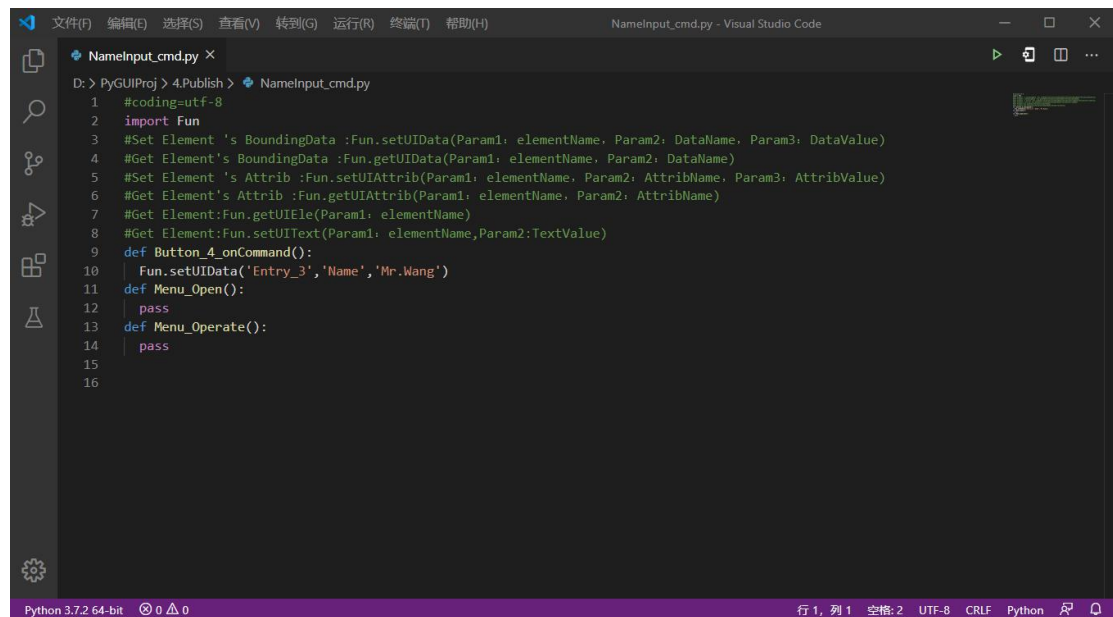


Here we define the file name as "NameInput", click "Save", if successful, a "Save Successful" prompt will pop up. We will see the generated "NameInput.py", "NameInput_cmd.py", "Fun.py" three files in the corresponding directory, where "NameInput.py" is the interface layout file, "NameInput_cmd.py" is the corresponding logical processing file, and "Fun.py" is I provide some functions for quick access to the interface. You can call it in "NameInput_cmd.py" to get the control or modify the properties of the control.



6. Logical processing

Open “NameInput_cmd.py” with a text editor, we will see the following screenshot:

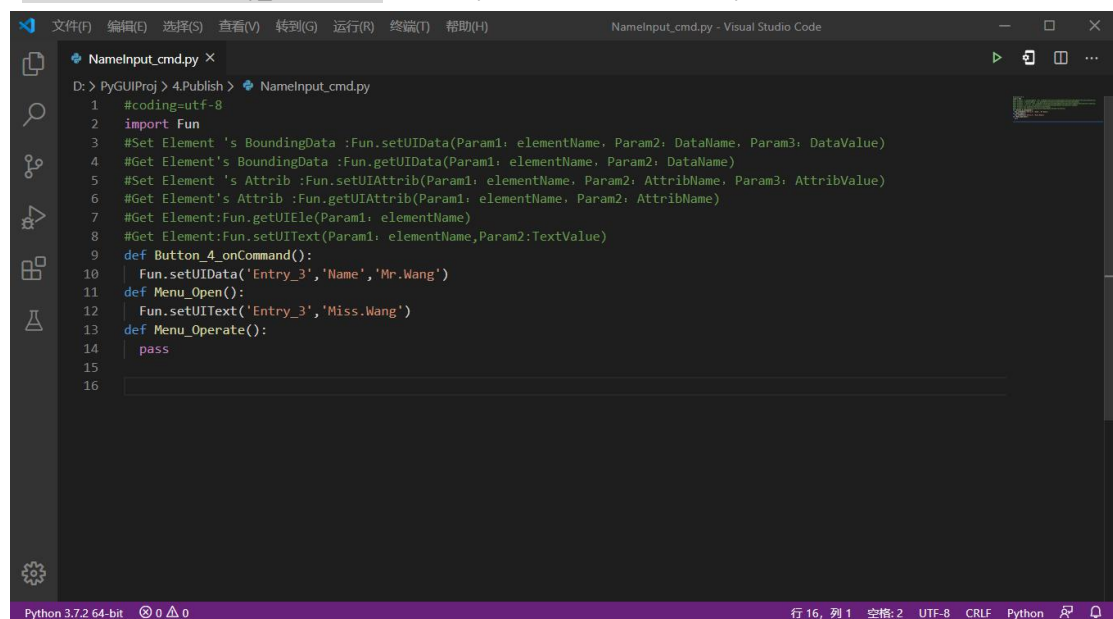


```

D:\> PyGUIProj > 4.Publish > NameInput_cmd.py
1  #coding=utf-8
2  import Fun
3  #Set Element 's BoundingData :Fun.setUIData(Param1: elementName, Param2: DataName, Param3: DataValue)
4  #Get Element's BoundingData :Fun.getUIData(Param1: elementName, Param2: DataName)
5  #Set Element 's Attrib :Fun.setUIAttrib(Param1: elementName, Param2: AttribName, Param3: AttribValue)
6  #Get Element's Attrib :Fun.getUIAttrib(Param1: elementName, Param2: AttribName)
7  #Get Element:Fun.getUIEle(Param1: elementName)
8  #Get Element:Fun.setUIText(Param1: elementName,Param2:TextValue)
9  def Button_4_onCommand():
10     Fun.setUIData('Entry_3','Name','Mr.Wang')
11     def Menu_Open():
12         pass
13     def Menu_Operate():
14         pass
15
16
  
```

There are some function annotations in “Fun”, you can access it directly through “Fun.FunctionName” For example, we change the code:

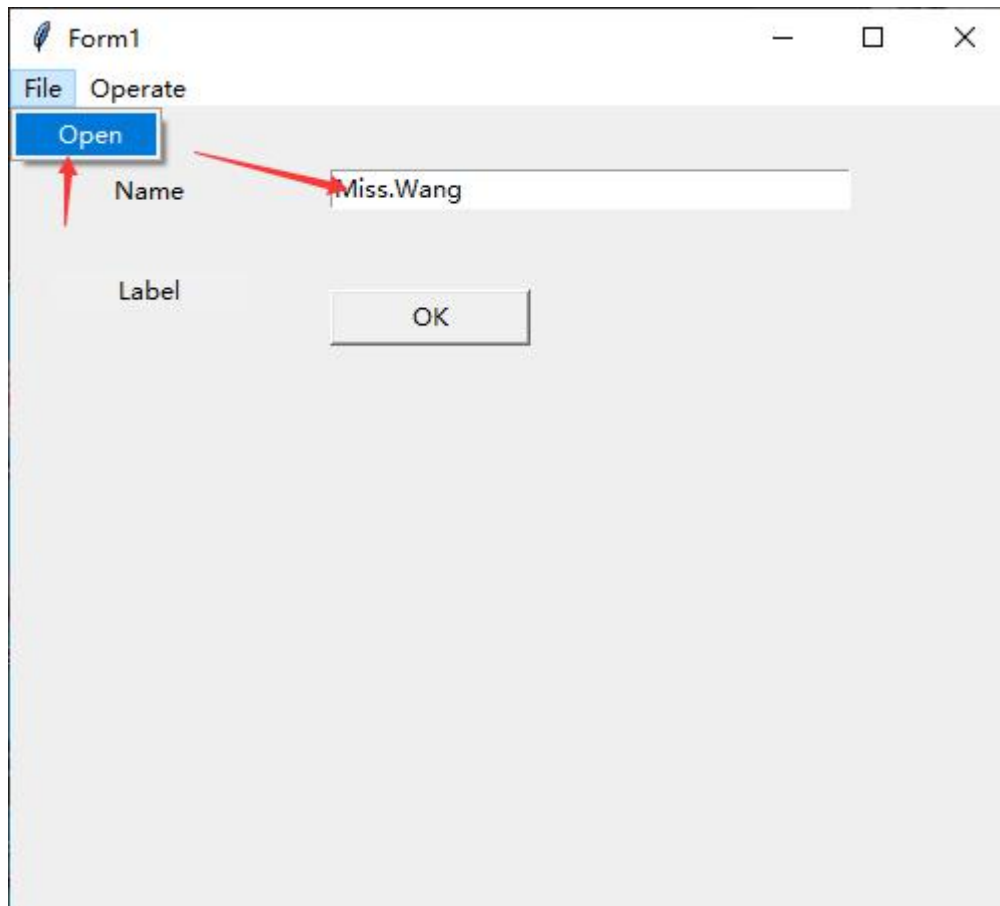
“Fun.setUIText (‘Entry_3’, ‘Mr.Li’)” to the predefined menu “Open File” item



```

D:\> PyGUIProj > 4.Publish > NameInput_cmd.py
1  #coding=utf-8
2  import Fun
3  #Set Element 's BoundingData :Fun.setUIData(Param1: elementName, Param2: DataName, Param3: DataValue)
4  #Get Element's BoundingData :Fun.getUIData(Param1: elementName, Param2: DataName)
5  #Set Element 's Attrib :Fun.setUIAttrib(Param1: elementName, Param2: AttribName, Param3: AttribValue)
6  #Get Element's Attrib :Fun.getUIAttrib(Param1: elementName, Param2: AttribName)
7  #Get Element:Fun.getUIEle(Param1: elementName)
8  #Get Element:Fun.setUIText(Param1: elementName,Param2:TextValue)
9  def Button_4_onCommand():
10     Fun.setUIData('Entry_3','Name','Mr.Wang')
11     def Menu_Open():
12         Fun.setUIText('Entry_3','Miss.Wang')
13     def Menu_Operate():
14         pass
15
16
  
```

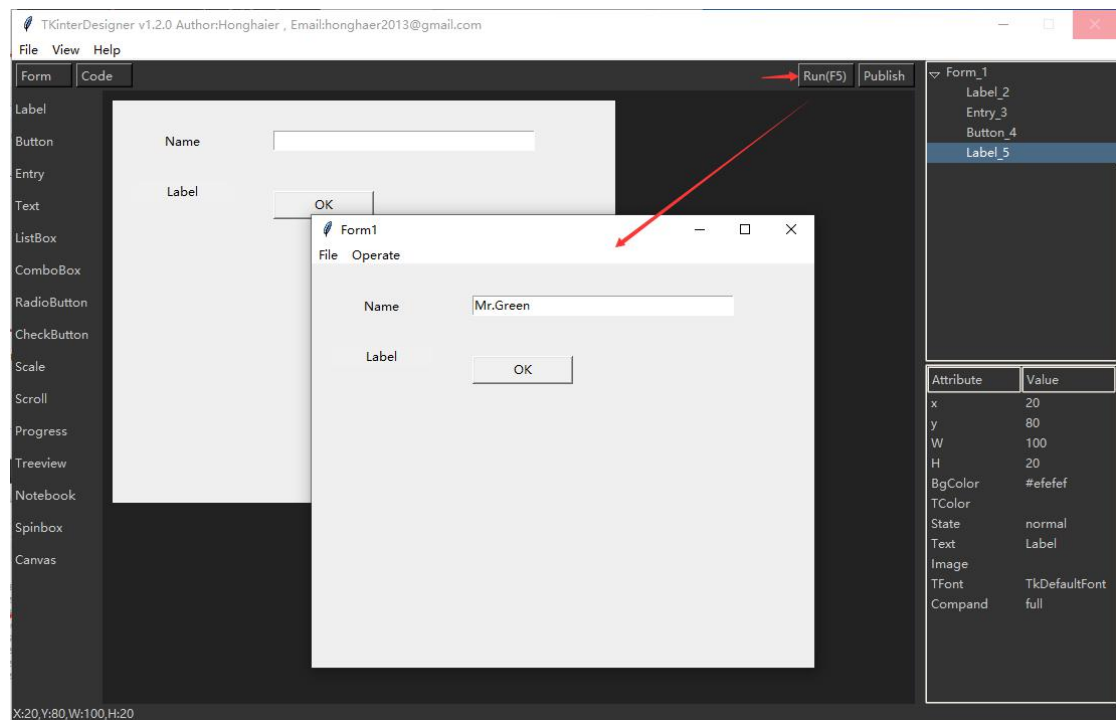
After saving, you can execute Test.py, click the menu item "Open File", and you will find that the value of the edit box becomes ‘Miss.Wang’.



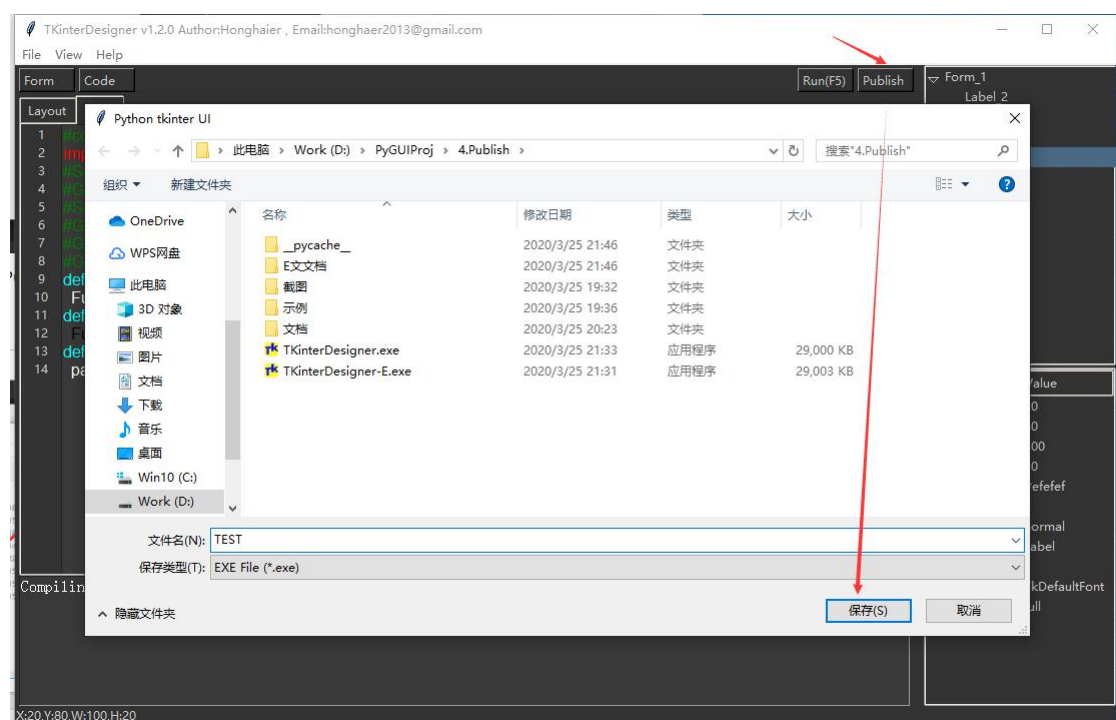
7. Run and Publish

we have designed an interface, and we want to quickly see the interface execution status. At this time, we only need to click the "Run" button in the shortcut button area to quickly see the results.

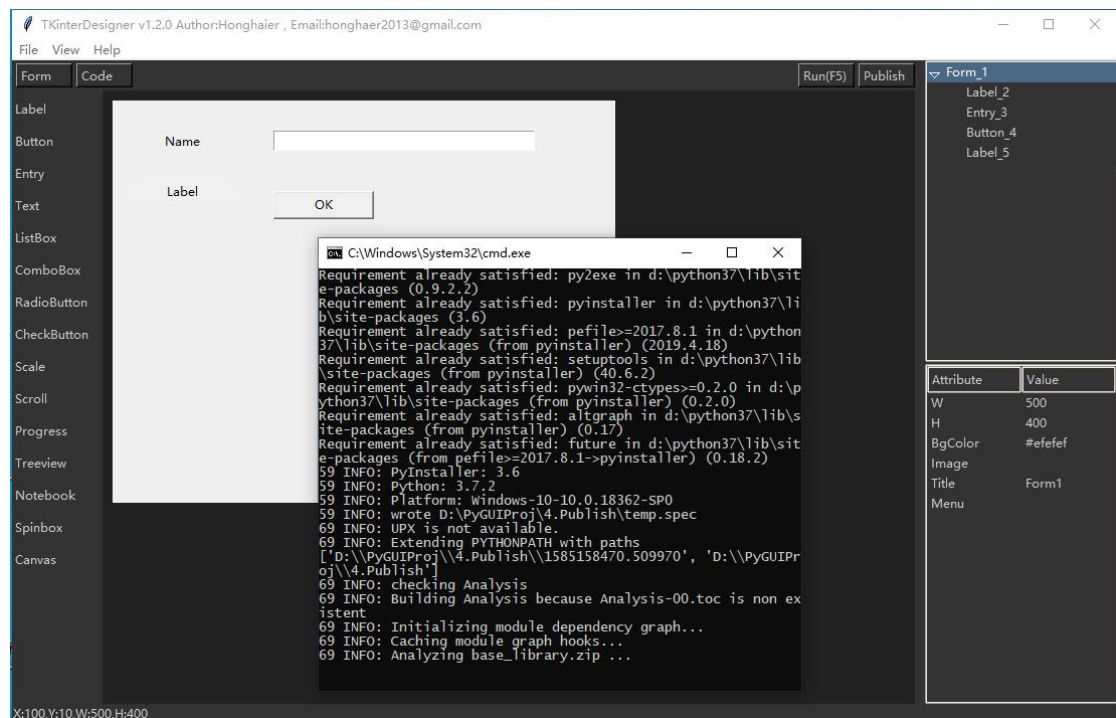
TKinterDesigner tutorial



If you have completed it, you can also click "Publish" to generate the corresponding execution file.



After clicking "OK", pyinstaller will be called to package the EXE.



Finally you can see a generated executable program:

__pycache__	2020/3/13 18:21	文件夹	
E文文档	2020/3/13 15:16	文件夹	
截图	2020/3/13 18:22	文件夹	
示例	2020/3/10 1:04	文件夹	
图片	2020/3/10 22:07	文件夹	
文档	2020/3/13 15:59	文件夹	
3.7说明.txt	2020/3/9 16:43	文本文档	1 KB
Python相关网站.txt	2020/3/13 2:11	文本文档	1 KB
run_temp_cmd.py	2020/3/13 18:21	Python File	1 KB
temp.spec	2020/3/13 18:22	SPEC 文件	1 KB
TEST.exe	2020/3/13 18:22	应用程序	8,651 KB
Test.py	2020/3/13 18:18	Python File	3 KB
Test_cmd.py	2020/3/13 18:18	Python File	1 KB
TKinterDesigner.exe	2020/3/13 15:54	应用程序	28,986 KB
TKinterDesigner-En.exe	2020/3/13 15:50	应用程序	28,986 KB

8. Custom modules and module libraries

Custom module is to provide developers with a basic method for loading and using custom package classes. I just started developing in this area, I believe that with more and more developers joining, our plug-in library will be further enriched. Please update from GitHub in time.

(1) Preparation of modules

A module is a Python class. Its rules are very simple. You only need to design some properties and interfaces for it. The only trick is that if you determine that you need to set basic properties in TKinterDesigner, then you need to set Set and get functions are written for attributes. I will introduce it with a simple Socket class:

```
#coding=utf-8
import socket
import tkinter
import threading
import time

def handle_socket(ServerSocket, ListBox):
    conn, addr = ServerSocket.accept()
    text = "Accept:"+str(addr)
    ListBox.insert(tkinter.END,text)
    while True:
        data = conn.recv(1024)
        text = "Receive Data:" + data.decode('utf-8')
        ListBox.insert(tkinter.END,text)
        time.sleep(3)
        send_data = 'got it'
        conn.send(send_data.encode('utf-8'))
        conn.close()

class MySocket:
    def __init__(self):
        self.s = None
        self.HOST = '127.0.0.1'
        self.PORT = 8888

    #SET HOST
    def set_HOST(self,host):
        self.HOST = host

    #GET HOST
    def get_HOST(self):
        return self.HOST

    #SET PORT
    def set_PORT(self,port):
        self.PORT = port

    #GET PORT
    def get_PORT(self):
        return self.PORT

    #Create Server
    def createServer(self,IPAddr,Port,ListBox):
        self.s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

```

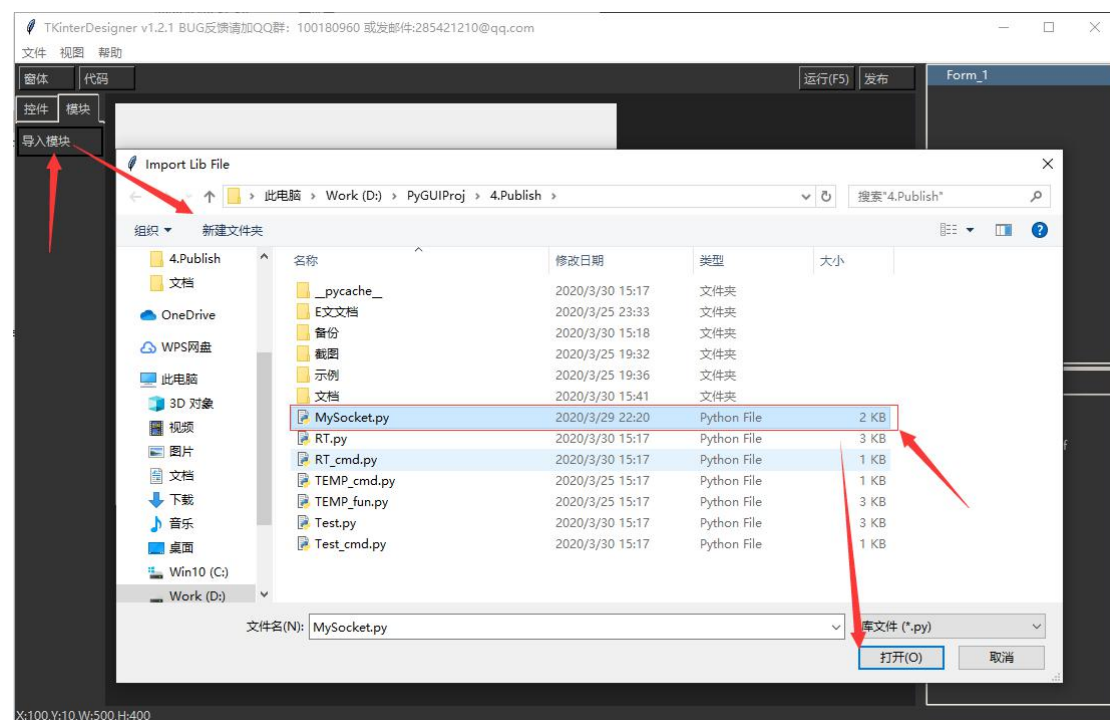
self.s.bind((self.HOST,int(self.PORT)))
self.s.listen(5)
ListBox.insert(tkinter.END,"Socket Created Succeed!")
client_thread = threading.Thread(target=handle_socket, args=[self.s, ListBox])
client_thread.start()

#Conn Server
def connServer(self,IPAddr,Port):
    self.s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    self.s.connect((self.HOST,int(self.PORT)))

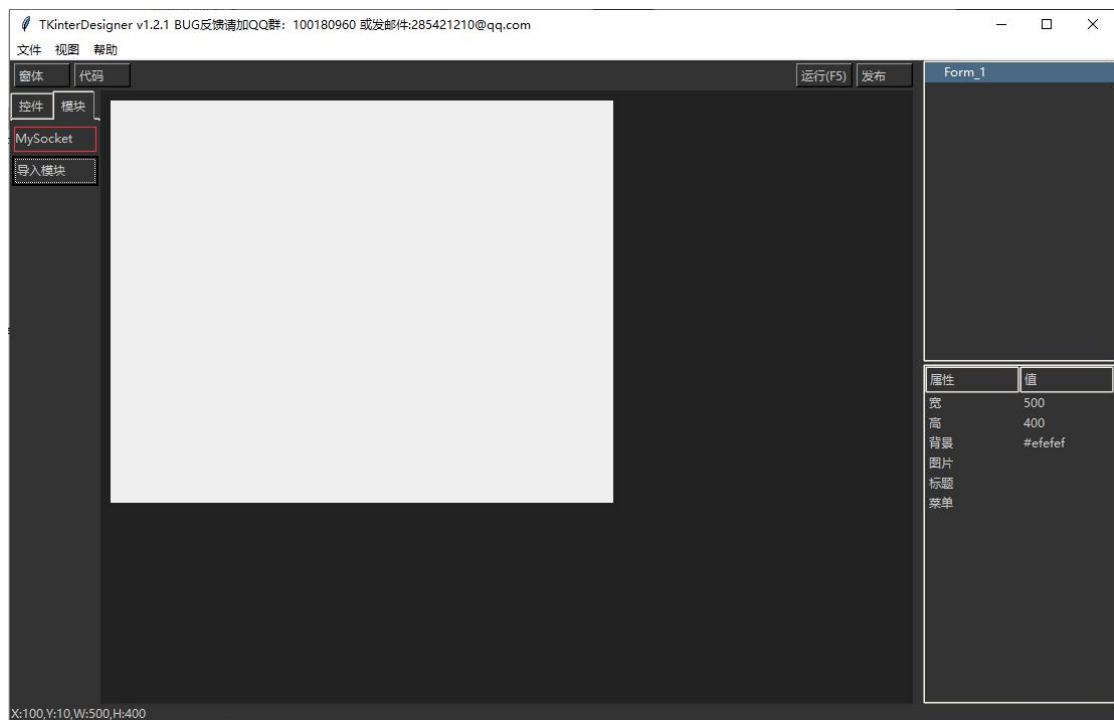
#Send Message
def sendMessage(self,text):
    msg = text.encode('utf-8')
    print("SendMsg:"+text)
    self.s.send(msg)

```

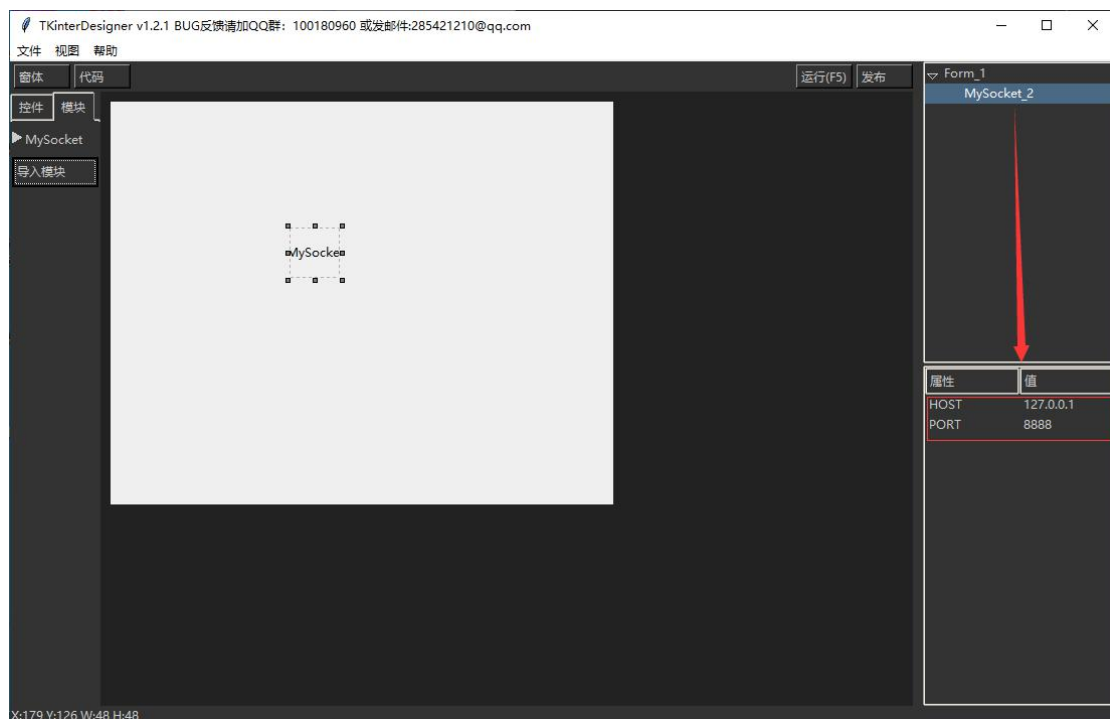
This is a test code I wrote by hand, let's call it "MySocket" for the time being. After saving MySocket.py, we open TKinterDesigner.



In the control selection area on the left, first switch to "Module", and then click "Import Module". In the pop-up file selection box, find the "MySocket.py" we just wrote. After opening the file, we will find that in the module selection One more module "MySocket" in the area:

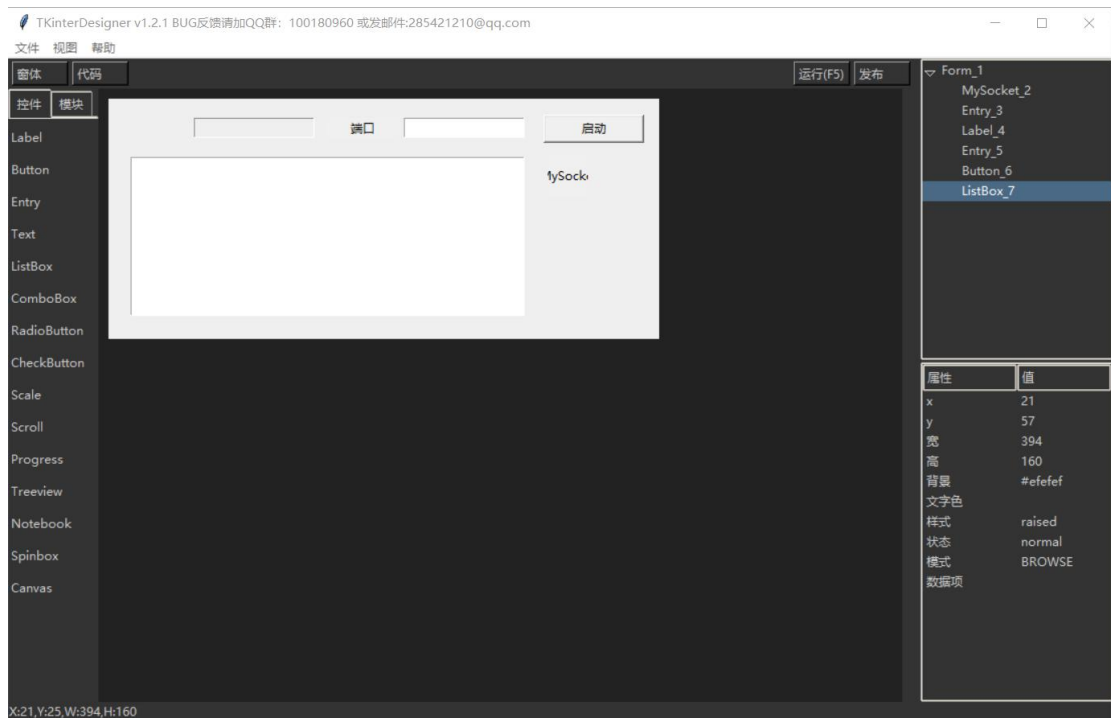


We can drag it to the Form in the design area like a drag-and-drop control. At this time, we can see that it is added to the control element tree list and displays the properties we want to expose at the property list for us to modify the initial value.



If we need to access it in code, we can get this instantiated object through "Fun.getUIEle (' MySocket_2')".

For example: we created such an UI:



This UI has two input boxes, which are IP and port respectively, and a list box for displaying messages. After clicking the right to start, the server creation function of the MySocket instance is called. I put the specific code in the Server directory of examples .

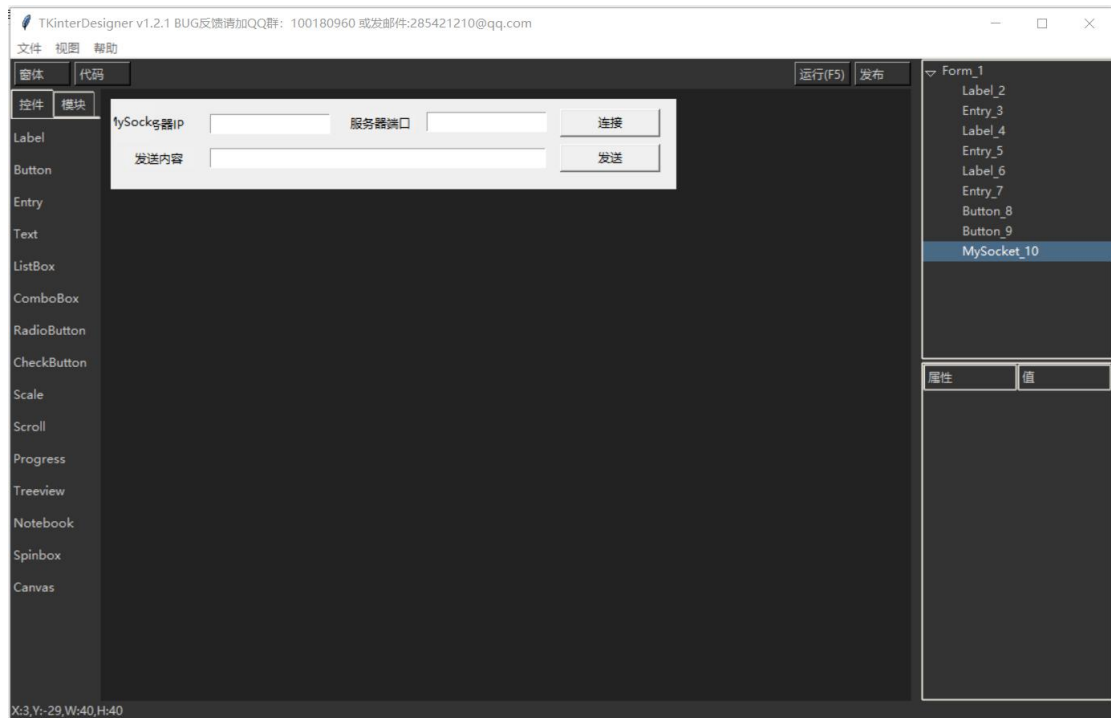
```

MySocket.py  Server_cmd.py X
D:\> PyGUIProj > 3.Compile-Exe > Server > Server_cmd.py > ...
1  #coding=utf-8
2  import Fun
3  #Set Element 's BoundingData :Fun.setUIData(Param1: elementName, Param2: DataName, Param3: DataValue)
4  #Get Element's BoundingData :Fun.getUIData(Param1: elementName, Param2: DataName)
5  #Set Element 's Attrib :Fun.setUIAttrib(Param1: elementName, Param2: AttribName, Param3: AttribValue)
6  #Get Element's Attrib :Fun.getUIAttrib(Param1: elementName, Param2: AttribName)
7  #Get Element:Fun.getUIEle(Param1: elementName)
8  #Set Element's Text :Fun.setUIText(Param1: elementName,Param2:TextValue)
9  #Get Element's Text :Fun.getUIText(Param1: elementName)
10 def Button_6_onCommand():
11     ListBox = Fun.getUIEle('ListBox_7')
12     MySocket = Fun.getUIEle('MySocket_2')
13     IPAddr = Fun.getUIData('Entry_3','IPAddr')
14     PORT = Fun.getUIData('Entry_5','Port')
15     MySocket.createServer(IPAddr,PORT,ListBox)
16
17
18

```

Make another client interface to send messages to the server:

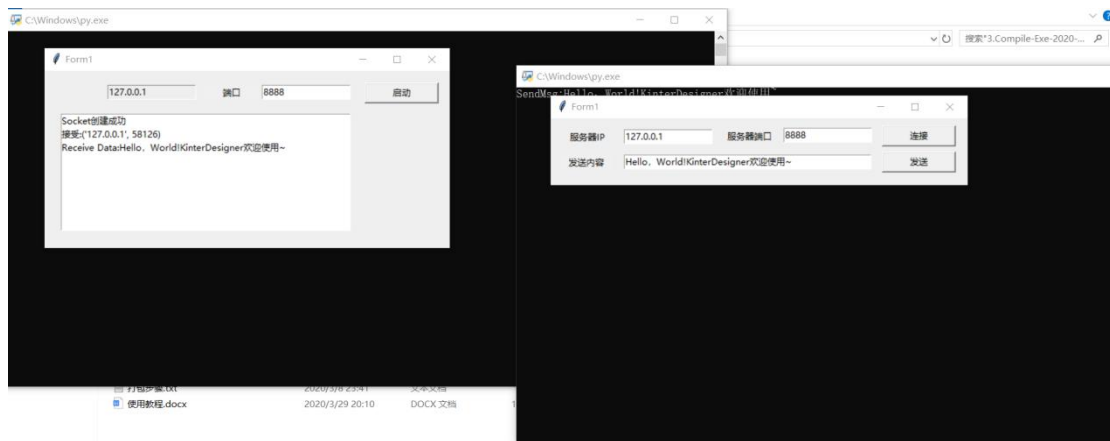
TKinterDesigner tutorial



We design the onCommand message for the corresponding button and implement the part that calls the MySocket instance:

```
MySocket.py  Server_cmd.py  Client_cmd.py X
D: > PyGUIProj > 3.Compile-Exe > Server > Client_cmd.py > ...
1  #coding=utf-8
2  import Fun
3  #Set Element 's BoundingData :Fun.setUIData(Param1: elementName, Param2: DataName, Param3: DataVa
4  #Get Element's BoundingData :Fun.getUIData(Param1: elementName, Param2: DataName)
5  #Set Element 's Attrib :Fun.setUIAttrib(Param1: elementName, Param2: AttribName, Param3: AttribVa
6  #Get Element's Attrib :Fun.getUIAttrib(Param1: elementName, Param2: AttribName)
7  #Get Element:Fun.getUIEle(Param1: elementName)
8  #Set Element's Text :Fun.setUIText(Param1: elementName,Param2:TextValue)
9  #Get Element's Text :Fun.getUIText(Param1: elementName)
10 def Button_8_onCommand():
11     MySocket = Fun.getUIEle('MySocket_10')
12     IPAddr = Fun.getUIData('Entry_3','IPAddr')
13     PORT = Fun.getUIData('Entry_5','Port')
14     MySocket.connServer(IPAddr,PORT)
15 def Button_9_onCommand():
16     MySocket = Fun.getUIEle('MySocket_10')
17     MsgText = Fun.getUIText('Entry_7')
18     MySocket.sendMessage(MsgText)
19
20
21
```

Finally they can run as follows:



If you still don't understand something, you can update the code on Github.

Finally, the meaning of the custom module is to hope that we can continuously explore the potential of everyone and develop more practical class libraries for the tool. I will not just import it through the interface so boring. Some interesting ideas, you can wait for the next edition.

三. About

First of all, because it's just a little motivation to learn Python intermittently for two months in my spare time, it has a lot of problems, but I will continue to improve, and I hope that interested Python enthusiasts will communicate with me, because I'm writing before I haven't finished a Python book, there are too many not getting started.

In addition, I am very optimistic about Python's interface tool requirements for rapid prototyping, and hope that Python is getting better and better.

Finally, I wish you every success in your work and good health ~

Honghaier

2020/03/31