

Building a basic GUI application step-by-step in Python with Tkinter and wxPython

There is a french translation of this document:

Il existe une version française de ce document: http://sebsauvage.net/python/gui/index_fr.html

In this page you will learn to build a basic GUI application in Python **step by step**.

The aim is:

- Mastering most common GUI techniques (widgets layout, GUI constraints, event binding, etc.)
- **Understanding *each and every single method and parameter used here*.**
- See two different major GUI toolkit and learn their differences.
- Serve as a basis for building your own GUI applications.

You will learn basic tips:

- building a GUI application class,
- creating widgets,
- laying them in containers,
- attaching and handling events,
- manipulating widgets values,
- etc.

Our constraints for this document:

- Do the same thing with **Tkinter** (the standard GUI toolkit provided with Python) and **wxPython** (an advanced, portable, popular GUI toolkit).
- Program it the *right* way
- Use plain english
- Explicit is better than implicit ;-)

Tkinter wraps **Tcl/tk** so that it can be used it in Python. **wxPython** wraps **wxWidgets** so that it can be used it in Python.

We will refer to one of the other in this document.

Both GUI toolkits are portable: It means that - if properly used - your GUI will work on all systems (Windows, Linux, MacOS X...).

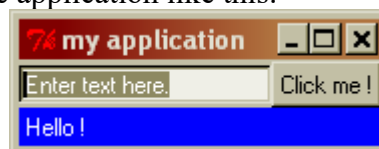
Table of contents

- [Our project](#)
- [STEP 1 : Import the GUI toolkit module](#)
- [STEP 2 : Create a class](#)
- [STEP 3 : The constructor](#)
- [STEP 4 : Keep track of our parent](#)

- [STEP 5 : Initialize our GUI](#)
- [STEP 6 : Creation of main](#)
- [STEP 7 : Loop it !](#)
- [STEP 8 : Layout manager](#)
- [STEP 9 : Adding the text entry](#)
- [STEP 10 : Adding the button](#)
- [STEP 11 : Adding the label](#)
- [STEP 12 : Enable resizing](#)
- [STEP 13 : Adding constraint](#)
- [STEP 14 : Adding event handlers](#)
- [STEP 15 : Changing the label](#)
- [STEP 16 : Display the value](#)
- [STEP 17 : Small refinement: auto-select the text field](#)
- [STEP 18 : Tkinter resize hiccup](#)
- [We're done !](#)
- [RAD tools and pixel coordinates](#)
- [Non blocking GUI ?](#)
- [Tkinter or wxPython ?](#)
- [Other GUI toolkit](#)
- [Packaging to an EXE](#)
- [About this document](#)

Our project

Our project is to create a very simple application like this:



A widget is a GUI element (a button, a checkbox, a label, a tab, a pane...).

Our application has 3 widgets:

- A text field ("Enter text here")
- A button ("Click me !")
- A blue label ("Hello !")

The behaviour we want:

- When ENTER is pressed in the text field, or the button is clicked, the blue label will display the text which was entered.
- We want to constraint window resize so that window can only be resized horizontally.
- If the window is resized, the text field and blue label will expand horizontally, like this:



In this document, we will show Tkinter and wxPython code side-by-side like this:

Tkinter source code is in this color, on the **left**.

wxPython equivalent code is in this color, on the **right**.

Lines added in each step will be highlighted **like this**.

Let's start !

STEP 1 : Import the GUI toolkit module

Let's import the required module.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program."
```

Tkinter is part of the standard Python distribution, so we expect it to be present. We just import it.

wxPython is not part of the standard Python distribution and has to be downloaded and installed separately. It's best to tell the user that wxPython is required but has not been found (hence the try/except ImportError).

STEP 2 : Create a class

It's best to put our application in a class.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter

class simpleapp_tk(Tkinter.Tk):
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
try:
    import wx
```

```
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
```

In **Tkinter**, we inherit from **Tkinter.Tk**, which is the base class for standard windows.
 In **wxPython**, we inherit from **wx.Frame**, which is the base class for standard windows.

STEP 3 : The constructor

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
```

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"
```

```
class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
```

simpleapp_tk derives from **Tkinter.Tk**, so we have to call the **Tkinter.Tk** constructor (**Tkinter.Tk.__init__()**).

simpleapp_wx inherits from **wx.Frame**, so we have to call the **wx.Frame** constructor (**wx.Frame.__init__()**).

A GUI is a hierarchy of objects: A button may be contained in a pane which is contained in a tab which is contained in a window, etc.

So each GUI element has a parent (its container, usually).

That's why both constructor have a **parent** parameter.

Keeping track of parents is usefull when we have to show/hide a group of widgets, repaint them on screen or simply destroy them when application exits.

wx.Frame object has two more parameters: **id** (an identifier of the widget) and **title** (the title of the window).

STEP 4 : Keep track of our parent

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
```

It is a good habit, when building any type of GUI component, to keep a reference to our parent.

STEP 5 : Initialize our GUI

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        pass
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
```

```
raise ImportError,"The wxPython module is required to run this program"
```

```
class simpleapp_wx(wx.Frame):  
    def __init__(self,parent,id,title):  
        wx.Frame.__init__(self,parent,id,title)  
        self.parent = parent  
        self.initialize()  
  
    def initialize(self):  
        self.Show(True)
```

It's usually best to have the portion of code which **creates all the GUI elements** (button, text fields...) separate from the **logic** of the program.

That's why we create the **initialize()** method. We will create all our widgets (buttons, text field, etc.) in this method.

For the moment, the Tkinter version contains nothing (hence the **pass** instruction which does nothing.) The **wxPython** version contains **self.Show(True)** to force the window to show up (otherwise it remains hidden) (This is not needed in Tkinter because Tkinter automatically shows created widgets.)

STEP 6 : Creation of main

```
#!/usr/bin/python  
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):  
    def __init__(self,parent):  
        Tkinter.Tk.__init__(self,parent)  
        self.parent = parent  
        self.initialize()
```

```
    def initialize(self):  
        pass
```

```
if __name__ == "__main__":  
    app = simpleapp_tk(None)  
    app.title('my application')
```

```
#!/usr/bin/python  
# -*- coding: iso-8859-1 -*-
```

```
try:  
    import wx  
except ImportError:  
    raise ImportError,"The wxPython module is required to run this program"
```

```

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')

```

Now we have a class, let's use it !

We create a **main** which is executed when the program is run from the command-line.

In **Tkinter**, we instantiate our class (**app=simpleapp_tk()**). We give it no parent (**None**), because it's the first GUI element we build.

We also give a title to our window (**app.title()**).

In **wxPython**, it is mandatory to instantiate a wx.App() object before creating GUI elements. That why we do **app=wx.App()**.

Then we instantiate our class (**frame=simpleapp_wx()**). We also give it no parent (**None**), because it's the first GUI element we build.

We use **-1** to let wxPython choose an identifier itself. And we give our window a title: '**my application**'.

STEP 7 : Loop it !

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

```

```

    def initialize(self):
        pass

```

```

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()

```

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()
```

Now, we tell each program to loop with **.mainloop()**

What's that ?

It means that each program will now **loop indefinitely**, waiting for events (user clicking a button, pressing keys, operating system asking our application to quit, etc.).

The Tkinter and wxPython main loops will receive these events and handle them accordingly.

This is call **event-driven programming** (Because the program will do nothing but wait for events, and only react when it receives an event.)

☼ *At this point, you can run both programs: They work !*

They will show an empty window.

Now close them, and let's get back to the code to add widgets.

STEP 8 : Layout manager

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
```



```

    self.parent = parent
    self.initialize()

def initialize(self):
    self.grid()

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()
        self.SetSizerAndFit(sizer)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

There are several ways to put widgets in a window (or another container): Add them side by side horizontally, vertically, in grid, etc.

So there are different classes (called **layout managers**) capable of placing the widgets in containers in different ways. Some are more flexible than others. Each container (window, pane, tab, dialog...) can have its own layout manager.

I recommend the **grid** layout manager. It's a simple grid where you put your widgets, just like you would place things in a spreadsheet (Excel, OpenOffice Calc...). For example: Put a button at column 2, row 1. Put a checkbox at column 5, row 3. etc. If you have to learn one, learn this one.

In **Tkinter**, calling **self.grid()** will simply create the grid layout manager, and tell our window to use it.

In **wxPython**, we create the grid layout manager with **sizer=wx.GridBagSizer()** and then ask our window to use it (**self.SetSizerAndFit(sizer)**).

Now let's add widgets.

STEP 9 : Adding the text entry

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
```

```
sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
```

```
self.SetSizerAndFit(sizer)  
self.Show(True)
```

```
if __name__ == "__main__":  
    app = wx.App()  
    frame = simpleapp_wx(None,-1,'my application')  
    app.MainLoop()
```

To add a widget, you always:

- first **create** the widget
- and **add it** to a layout manager

Frist, we **create the widget**:

In **Tkinter**, we create the **Entry** widget (**self.entry=Tkinter.Entry()**)

In **wxPython**, we create **TextCtrl** object (**self.entry=wx.TextCtrl()**)

In both cases we pass **self** as **parent** parameter, because our window will be the parent of this widgets:
They will appear in our window.

wxPython.**TextCtrl** has 2 more parameters: **-1** (so that wxPython automatically assigns an identifier.),
and the text itself (**u'Enter text here.'**).

(For the text in Tkinter.Entry, we will see this later.)

Note that we keep a referece to this widget in our class: **self.entry=...**

That's because we need to access it later, in other methods.

Now, time to **add them to the layout manager**.

In **Tkinter**, we call the **.grid()** method on the widget. We indicate where to put it in the grid
(**column=0, row=0**).

When a cell grows larger than the widget is contains, you can ask the widget to stick to some edges of
the cell. That's the **sticky='EW'**.

(E=east (left), W=West (right), N=North (top), S=South (bottom))

We specified 'EW', which means the widget will try to stick to both left and right edges of its cell.

(That's one of our goals: Have the text entry expand when the window is resized.)

In **wxPython**, we call the **.Add()** method of the layout manager.

We pass the widget we just created (**self.entry**) and its coordinates in grid (**0,0**).

(1,1) is the span: In our case, the widget does not span over several cells.

wx.EXPAND tells the layout manager to expand the text entry if its cell is resized.

✱ *At this point, you can run both programs: They work !*

We have a window with a single text field. You can even enter some text.

Hey ! The text field does not resize when I resize the window ! You lied !

Keep cool, there is a good reason for this: We told the text fields to automatically expand if their **column or cell** expands, but we did not yet tell the **layout manager** to expand the columns if the window is resized. We'll see that later.

STEP 10 : Adding the button

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

        self.entry = Tkinter.Entry(self)
        self.entry.grid(column=0,row=0,sticky='EW')

        button = Tkinter.Button(self,text=u"Click me !")
        button.grid(column=1,row=0)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"
```

```

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))

        self.SetSizerAndFit(sizer)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Its quite easy in both cases: Create the button, and add it.

Note that in this case, we do not keep a referece to the button (because we will not read or alter its value later).

✳ *At this point, you can run both programs: They work !*

We have a window with a text field and a button.

STEP 11 : Adding the label

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

    def initialize(self):
        self.grid()

```

```

self.entry = Tkinter.Entry(self)
self.entry.grid(column=0,row=0,sticky='EW')

button = Tkinter.Button(self,text=u"Click me !")
button.grid(column=1,row=0)

label = Tkinter.Label(self,
                        anchor="w",fg="white",bg="blue")
label.grid(column=0,row=1,columnspan=2,sticky='EW')

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        self.SetSizerAndFit(sizer)
        self.Show(True)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')

```

app.MainLoop()

We also create and add the label.

- This is a **Label** object in **Tkinter**.
- This is a **StaticText** in **wxPython**.

For the **color**: We use a white text on a blue background.

- In **Tkinter**, it's **fg="white",bg="blue"**.
- In **wxPython**, we have to call two methods (**SetForegroundColour** and **SetBackgroundColour**)

For the **text alignment**:

- In **Tkinter**, **anchor="w"** means that the text should be left aligned in the **label**.
- In **wxPython**, the text is aligned to the left by default.

For the label **position** in grid:

- In **Tkinter**, this is again the **.grid()** method, but this time we also span it across two cells (so that it appears below the text field **and** the button.): This is the **columnspan=2** parameter.
- We do the same in **wxPython** by specifying a span of (1,2) (which mean: *span 1 cell vertically, and 2 cells horizontally*).

For the label **expansion**:

- Again, we use **sticky="EW"** for **Tkinter**
- We use **wx.EXPAND** for **wxPython**.

✱ *At this point, you can run both programs: They work !*

Ok. 3 widgets. So what's next ?

STEP 12 : Enable resizing

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```
    def __init__(self,parent):
```

```
        Tkinter.Tk.__init__(self,parent)
```

```

    self.parent = parent
    self.initialize()

def initialize(self):
    self.grid()

    self.entry = Tkinter.Entry(self)
    self.entry.grid(column=0,row=0,sticky='EW')

    button = Tkinter.Button(self,text=u"Click me !")
    button.grid(column=1,row=0)

    label = Tkinter.Label(self,
                           anchor="w",fg="white",bg="blue")
    label.grid(column=0,row=1,columnspan=2,sticky='EW')

    self.grid_columnconfigure(0,weight=1)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

```



```
sizer.AddGrowableCol(0)
self.SetSizerAndFit(sizer)
self.Show(True)
```

```
if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()
```

Now we tell the layout manager to resize its columns and rows when the window is resized.
Well, not the rows. Only the first column (**0**).

Thats AddGrowableCol() for wxPython.
Thats grid_columnconfigure() for Tkinter.

Note that there are additional parameters. For example, some column can grow more than other when resized. That's what the **weight** parameter is for. (In our case, we simply set it to 1).

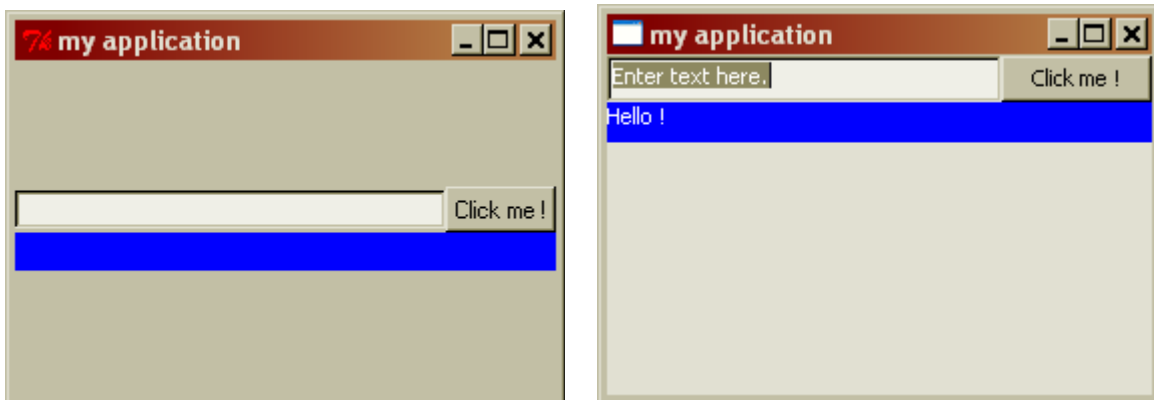
☼ *At this point, you can run both programs: They work !*

Now try resizing the window.

See ?

Text fields and blue label now properly resize to adapt to window size.

But that's not pretty when resizing the window **vertically**:



So let's add a constraint so that the user can only resize the window horizontally.

STEP 13 : Adding constraint

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```
    def __init__(self,parent):
```

```
        Tkinter.Tk.__init__(self,parent)
```

```
        self.parent = parent
```

```
        self.initialize()
```

```
    def initialize(self):
```

```
        self.grid()
```

```
        self.entry = Tkinter.Entry(self)
```

```
        self.entry.grid(column=0,row=0,sticky='EW')
```

```
        button = Tkinter.Button(self,text=u"Click me !")
```

```
        button.grid(column=1,row=0)
```

```
        label = Tkinter.Label(self,
```

```
                                anchor="w",fg="white",bg="blue")
```

```
        label.grid(column=0,row=1,columnspan=2,sticky='EW')
```

```
        self.grid_columnconfigure(0,weight=1)
```

```
        self.resizable(True,False)
```

```
if __name__ == "__main__":
```

```
    app = simpleapp_tk(None)
```

```
    app.title('my application')
```

```
    app.mainloop()
```

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
try:
```

```
    import wx
```

```
except ImportError:
```

```
    raise ImportError,"The wxPython module is required to run this program"
```

```
class simpleapp_wx(wx.Frame):
```

```
    def __init__(self,parent,id,title):
```

```
        wx.Frame.__init__(self,parent,id,title)
```

```
        self.parent = parent
```

```
        self.initialize()
```

```
    def initialize(self):
```

```
        sizer = wx.GridBagSizer()
```

```
        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
```

```
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
```

```

button = wx.Button(self,-1,label="Click me !")
sizer.Add(button, (0,1))

self.label = wx.StaticText(self,-1,label=u'Hello !')
self.label.SetBackgroundColour(wx.BLUE)
self.label.SetForegroundColour(wx.WHITE)
sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

sizer.AddGrowableCol(0)
self.SetSizerAndFit(sizer)
self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
self.Show(True)

```

```

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Now we prevent the vertical resizing of the window:

- In **Tkinter**, we use **.resizable(True,False)**.
- In **wxPython**, you can specify the maximum and minimum widths and heights of your window. We set minimum and maximum **height** to our current window height (**self.GetSize().y**) so that the window cannot be resized vertically. We leave **-1, -1** for the **width** so that the window can be freely resized horizontally (**-1** means "No limit").

☼ *At this point, you can run both programs: They work !*

Now try resizing the window.

STEP 14 : Adding event handlers

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```
    def __init__(self,parent):
```

```
        Tkinter.Tk.__init__(self,parent)
```

```
        self.parent = parent
```

```
        self.initialize()
```

```
    def initialize(self):
```

```
        self.grid()
```

```

self.entry = Tkinter.Entry(self)
self.entry.grid(column=0,row=0,sticky='EW')
self.entry.bind("<Return>", self.OnPressEnter)

button = Tkinter.Button(self,text=u"Click me !",
                        command=self.OnButtonClick)
button.grid(column=1,row=0)

label = Tkinter.Label(self,
                      anchor="w",fg="white",bg="blue")
label.grid(column=0,row=1,columnspan=2,sticky='EW')

self.grid_columnconfigure(0,weight=1)
self.resizable(True,False)

def OnButtonClick(self):
    print "You clicked the button !"

def OnPressEnter(self,event):
    print "You pressed enter !"

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

```

```

self.label = wx.StaticText(self,-1,label=u'Hello !')
self.label.SetBackgroundColour(wx.BLUE)
self.label.SetForegroundColour(wx.WHITE)
sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

sizer.AddGrowbleCol(0)
self.SetSizerAndFit(sizer)
self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
self.Show(True)

```

```

def OnButtonClick(self,event):
    print "You clicked the button !"

```

```

def OnPressEnter(self,event):
    print "You pressed enter !"

```

```

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Event handlers are methods which will be called when something happens in the GUI. We bind the event handlers to specific **widgets** on specific **events** only. So let's do something when the **button** is **clicked** or **ENTER** pressed in the **text field**.

- We create a **OnButtonClick()** method which will be called then the button is clicked.
- We create a **OnPressEnter()** method which will be called then ENTER is pressed in the text field.

Then we bind these methods to the widgets.

For the **button**:

- In **Tkinter**, simply add **command=self.OnButtonClick**
- In **wxPython**, we use the **.Bind()** method:
button is the widget on which we want to catch an event.
wx.EVT_BUTTON is the kind of event we want to catch (a click on a button)
self.OnButtonClick is the method we want to be fired when this event is caught.

For the **text field**:

- In **Tkinter**, we use a **.bind()** method.
"<Return>" is the key we want to catch.
self.OnPressEnter is the method we want to be fired when this event is caught.
- In **wxPython**, this is again the **.Bind()** method but with the **wx.EVT_TEXT_ENTER** event.

Therefore:

- Clicking on the button will trigger the **OnButtonClick()** method.
- Pressing ENTER in the text field will trigger the **OnPressEnter()** method.

✿ *At this point, you can run both programs: They work !*

Enter some text, then press ENTER or click the "Click me !" button: Some text will appear.
(Tkinter will display the text in the console ; wxPython will popup the text.)

STEP 15 : Changing the label

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```
    def __init__(self,parent):
```

```
        Tkinter.Tk.__init__(self,parent)
```

```
        self.parent = parent
```

```
        self.initialize()
```

```
    def initialize(self):
```

```
        self.grid()
```

```
        self.entry = Tkinter.Entry(self)
```

```
        self.entry.grid(column=0,row=0,sticky='EW')
```

```
        self.entry.bind("<Return>", self.OnPressEnter)
```

```
        button = Tkinter.Button(self,text=u"Click me !",
```

```
                                command=self.OnButtonClick)
```

```
        button.grid(column=1,row=0)
```

```
        self.labelVariable = Tkinter.StringVar()
```

```
        label = Tkinter.Label(self,textvariable=self.labelVariable,  
                              anchor="w",fg="white",bg="blue")
```

```
        label.grid(column=0,row=1,columnspan=2,sticky='EW')
```

```
        self.grid_columnconfigure(0,weight=1)
```

```
        self.resizable(True,False)
```

```
    def OnButtonClick(self):
```

```
        self.labelVariable.set("You clicked the button !")
```

```

def OnPressEnter(self,event):
    self.labelVariable.set("You pressed enter !")

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        sizer.AddGrowbleCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
        self.Show(True)

    def OnButtonClick(self,event):
        self.label.SetLabel("You clicked the button !")

    def OnPressEnter(self,event):
        self.label.SetLabel("You pressed enter !")

```

```

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Now let's change the label.

- Under **wxPython**, it's easy: simply call **.SetLabel()** on our label (**self.label**)
- Under **Tkinter**, this is a bit more tricky. You have to:
 - create a special Tkinter variable (**self.labelVariable = Tkinter.StringVar()**)
 - then bind it to the widget (**textvariable=self.labelVariable**)
 - then use **set()** or **get()** to set or read its value (**self.labelVariable.set("You clicked the button !")**)

In Tkinter, each time you want to read or set values in a widgets (text field, label, checkbox, radio button, etc.), you have to create a Tkinter variable and bind it to the widget. There are several Tkinter variable types (StringVar, IntVar, DoubleVar, BooleanVar).

With wxPython, you directly call a method on a widget to read/set its value.

✱ *At this point, you can run both programs: They work !*

Press ENTER or click the button: The label will change.

STEP 16 : Display the value

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```

    def __init__(self,parent):
        Tkinter.Tk.__init__(self,parent)
        self.parent = parent
        self.initialize()

```

```

    def initialize(self):
        self.grid()

```

```

        self.entryVariable = Tkinter.StringVar()
        self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
        self.entry.grid(column=0,row=0,sticky='EW')
        self.entry.bind("<Return>", self.OnPressEnter)
        self.entryVariable.set(u"Enter text here.")

```



```

        button = Tkinter.Button(self,text=u"Click me !",
                                command=self.OnButtonClick)
        button.grid(column=1,row=0)

        self.labelVariable = Tkinter.StringVar()
        label = Tkinter.Label(self,textvariable=self.labelVariable,
                                anchor="w",fg="white",bg="blue")
        label.grid(column=0,row=1,columnspan=2,sticky='EW')
        self.labelVariable.set(u"Hello !")

        self.grid_columnconfigure(0,weight=1)
        self.resizable(True,False)

    def OnButtonClick(self):
        self.labelVariable.set( self.entryVariable.get()+" (You clicked the button)" )

    def OnPressEnter(self,event):
        self.labelVariable.set( self.entryVariable.get()+" (You pressed ENTER)" )

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

```

```

self.label = wx.StaticText(self,-1,label=u'Hello !')
self.label.SetBackgroundColour(wx.BLUE)
self.label.SetForegroundColour(wx.WHITE)
sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

sizer.AddGrowbleCol(0)
self.SetSizerAndFit(sizer)
self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
self.Show(True)

def OnButtonClick(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You clicked the button)" )

def OnPressEnter(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You pressed ENTER)" )

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Now let's read the value of the text field and display it in the blue label.

- In **wxPython**, this is a matter of **self.entry.GetValue()**
- In **Tkinter**, we have to create a Tkinter variable again, so that we can do **self.entryVariable.get()**.

As we have a text variable to access the Tkinter text field content, we can also set the default value ("Enter text here." and "Hello !")

☼ ***At this point, you can run both programs: They work !***

Enter some text in the text field, then press ENTER or click the button: The label will display the text you typed in.

STEP 17 : Small refinement: auto-select the text field

```

#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

import Tkinter

class simpleapp_tk(Tkinter.Tk):
    def __init__(self,parent):

```

```

Tkinter.Tk.__init__(self,parent)
self.parent = parent
self.initialize()

def initialize(self):
    self.grid()

    self.entryVariable = Tkinter.StringVar()
    self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
    self.entry.grid(column=0,row=0,sticky='EW')
    self.entry.bind("<Return>", self.OnPressEnter)
    self.entryVariable.set(u"Enter text here.")

    button = Tkinter.Button(self,text=u"Click me !",
                           command=self.OnButtonClick)
    button.grid(column=1,row=0)

    self.labelVariable = Tkinter.StringVar()
    label = Tkinter.Label(self,textvariable=self.labelVariable,
                          anchor="w",fg="white",bg="blue")
    label.grid(column=0,row=1,columnspan=2,sticky='EW')
    self.labelVariable.set(u"Hello !")

    self.grid_columnconfigure(0,weight=1)
    self.resizable(True,False)
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

def OnButtonClick(self):
    self.labelVariable.set( self.entryVariable.get()+" (You clicked the button)" )
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

def OnPressEnter(self,event):
    self.labelVariable.set( self.entryVariable.get()+" (You pressed ENTER)" )
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

```

```

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        sizer.AddGrowableCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
        self.entry.SetFocus()
        self.entry.SetSelection(-1,-1)
        self.Show(True)

    def OnButtonClick(self,event):
        self.label.SetLabel( self.entry.GetValue() + " (You clicked the button)" )
        self.entry.SetFocus()
        self.entry.SetSelection(-1,-1)

    def OnPressEnter(self,event):
        self.label.SetLabel( self.entry.GetValue() + " (You pressed ENTER)" )
        self.entry.SetFocus()
        self.entry.SetSelection(-1,-1)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Here is a small refinement: The text field will be automatically selected is the user presses ENTER or clicks the button, so that he/she can immediately type a new text again (replacing the existing one.).

We first set the focus in this element (**focus_set()** or **SetFocus()**), then select all the text (**selection_range()** or **SetSelection()**).

STEP 18 : Tkinter resize hiccup

```
#!/usr/bin/python
```

```
# -*- coding: iso-8859-1 -*-
```

```
import Tkinter
```

```
class simpleapp_tk(Tkinter.Tk):
```

```
    def __init__(self,parent):
```

```
        Tkinter.Tk.__init__(self,parent)
```

```
        self.parent = parent
```

```
        self.initialize()
```

```
    def initialize(self):
```

```
        self.grid()
```

```
        self.entryVariable = Tkinter.StringVar()
```

```
        self.entry = Tkinter.Entry(self,textvariable=self.entryVariable)
```

```
        self.entry.grid(column=0,row=0,sticky='EW')
```

```
        self.entry.bind("<Return>", self.OnPressEnter)
```

```
        self.entryVariable.set(u"Enter text here.")
```

```
        button = Tkinter.Button(self,text=u"Click me !",
```

```
                                command=self.OnButtonClick)
```

```
        button.grid(column=1,row=0)
```

```
        self.labelVariable = Tkinter.StringVar()
```

```
        label = Tkinter.Label(self,textvariable=self.labelVariable,
```

```
                              anchor="w",fg="white",bg="blue")
```

```
        label.grid(column=0,row=1,columnspan=2,sticky='EW')
```

```
        self.labelVariable.set(u"Hello !")
```

```
        self.grid_columnconfigure(0,weight=1)
```

```
        self.resizable(True,False)
```

```
        self.update()
```

```
        self.geometry(self.geometry())
```

```
        self.entry.focus_set()
```

```
        self.entry.selection_range(0, Tkinter.END)
```

```
    def OnButtonClick(self):
```

```
        self.labelVariable.set( self.entryVariable.get()+" (You clicked the button)" )
```

```
        self.entry.focus_set()
```

```
        self.entry.selection_range(0, Tkinter.END)
```

```

def OnPressEnter(self,event):
    self.labelVariable.set( self.entryVariable.get()+" (You pressed ENTER)" )
    self.entry.focus_set()
    self.entry.selection_range(0, Tkinter.END)

if __name__ == "__main__":
    app = simpleapp_tk(None)
    app.title('my application')
    app.mainloop()
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-

try:
    import wx
except ImportError:
    raise ImportError,"The wxPython module is required to run this program"

class simpleapp_wx(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id,title)
        self.parent = parent
        self.initialize()

    def initialize(self):
        sizer = wx.GridBagSizer()

        self.entry = wx.TextCtrl(self,-1,value=u"Enter text here.")
        sizer.Add(self.entry,(0,0),(1,1),wx.EXPAND)
        self.Bind(wx.EVT_TEXT_ENTER, self.OnPressEnter, self.entry)

        button = wx.Button(self,-1,label="Click me !")
        sizer.Add(button, (0,1))
        self.Bind(wx.EVT_BUTTON, self.OnButtonClick, button)

        self.label = wx.StaticText(self,-1,label=u'Hello !')
        self.label.SetBackgroundColour(wx.BLUE)
        self.label.SetForegroundColour(wx.WHITE)
        sizer.Add( self.label, (1,0),(1,2), wx.EXPAND )

        sizer.AddGrowbleCol(0)
        self.SetSizerAndFit(sizer)
        self.SetSizeHints(-1,self.GetSize().y,-1,self.GetSize().y );
        self.entry.SetFocus()
        self.entry.SetSelection(-1,-1)
        self.Show(True)

    def OnButtonClick(self,event):
        self.label.SetLabel( self.entry.GetValue() + " (You clicked the button)" )

```

```

self.entry.SetFocus()
self.entry.SetSelection(-1,-1)

def OnPressEnter(self,event):
    self.label.SetLabel( self.entry.GetValue() + " (You pressed ENTER)" )
    self.entry.SetFocus()
    self.entry.SetSelection(-1,-1)

if __name__ == "__main__":
    app = wx.App()
    frame = simpleapp_wx(None,-1,'my application')
    app.MainLoop()

```

Tkinter has a small resize hiccup: It will try to accomodate window size *all the time* to widgets. This is nice, but not always a desired behaviour.

For example, type a loooooooooong text in the text field, press ENTER: The window grows.

Type a short text again, press ENTER: the window shrinks.

You usually do not want your application window grow and shrink automatically all the time. The users will not like that.

That's why we fix the size of the window by setting the window size to its own size

(**self.geometry(self.geometry())**).

This way, Tkinter will stop trying to accomodate window size all the time.

(And we perform an **update()** to make sure Tkinter has finished rendering all widgets and evaluating their size.)

wxPython/wxWidgets does not have this behaviour, so we have nothing special to do.

We're done !

We now have our application. :-)

You can download the sources of both programs: [simpleapp_tk.py](#) [simpleapp_wx.py](#)

Now some remarks.

RAD tools and pixel coordinates

It may look a bit cumbersome to build a GUI this way, but once you get the picture, it's pretty straightforward to add elements in a GUI and compose them in nested containers (tabbed panes, resizable panes, scrollable panes, etc.). In then end, that's pretty flexible.

If you prefer VB-like RAD environments, you can use things like *Boa Constructor*. But I'm less and less fond of these tools.

They tend to generate overkill code, and sometimes even fail to read them back correctly (Any VisualStudio.Net user in the audience ? I don't like my GUI beeing completely fucked up by the "designer", event bindings suddenly disappearing from the code for no reason, the designer not understanding basic HTML and CSS, or the designer generating a general protection fault when re-opening a project. :-@).

It may be a bit longer to program a GUI without a RAD, but you have greater control over what is done, and - last but not least - you can use grid layout managers (Most RAD environments only work in fixed pixels coordinates.)

Why not work in pixels coordinates ?

- Ever encountered dialogs where you could not read the text because it overflowed out of a widget ? That's pixel coordinates in action.
If you use pixel coordinates, your GUI may be unusable on systems with different font sizes. With a grid sizer, the widgets can adapt to their content.
- Ever beeing bothered with those ridiculously small "open file" dialogs when you have a 1600x1200 screen ? That's pixel coordinates in action too.
With a grid sizer, the user can resize the window to enjoy it full screen.

Non blocking GUI ?

You must keep in mind that while an event handler is performing, the GUI toolkit loop cannot receive new event and process them. (New event will be put in a waiting queue.)

Therefore the GUI will be completely blocked until the event handler method has finished.

(The buttons and menu will be unresponsive, and the window cannot even be moved or resized.

I'm sure you already have encountered such applications.)

You have two ways around this:

- Perform only *short* actions in event handlers.
- Use threads.

Threads programming is not trivial. It may even become a nightmare when debugging. But it's worth the price if you want to create an application which looks responsive (For example, my program webGobbler has a thread exclusively dedicated to the GUI, so that it's responsive even when crunching big images or waiting for web servers to respond to network requests.)

Most users expect a modern GUI to be non-blocking.

Beware that most GUI toolkit are not thread-safe. It means that if two threads try to change the value of a widget, this may hang your entire application (or even generate a general protection fault.). You have to use critical sections or other means (message queues, etc.)

Python supports threads and has several object to deal with this (Thread-safe queue object, semaphores, critical sections...)


Clearly separating the logic of your program from the GUI will greatly ease the transition to multi-threaded programming.


That's why I recommended separating GUI and program logic in this document.

Besides, you will be able to create several different GUIs for your program if you want !


Tkinter or wxPython ?


Tkinter / Tcl/tk :


 Tkinter is provided in the standard Python distribution. Most Python users will be able to use your program straight out of the box.

 No advanced widgets, although you can use [Pmw](#) (Python Megawidgets).

wxPython / wxWidgets:

 Uses native operating system widgets when possible (look & feel closer to the operating system's).

 More advanced widgets (date selector, floating toolbars, shaped windows, treeviews, tooltips, etc.)

 Not part of the standard Python distribution. Must be downloaded and installed separately.

Other GUI toolkit

There are a lot more available GUI toolkits, like GTK (through pyGTK), Qt (through PyQt), MFC, SWING, Fltk and even wrappers around Tkinter/wxPython themselves (PythonCard, etc.)

For more GUI toolkits, see

- <http://wiki.python.org/moin/GuiProgramming>
- http://home.pacbell.net/atai/guitool/#free_python
- <http://awaretek.com/toolkits.html>

Packaging to an EXE

If your application uses Tkinter or wxPython, is it possible to package it in the form of an

executable (binary) with programs like py2exe or cx_Freeze.
(Although it sometimes requires some tricks like this
one: http://sebsauvage.net/python/snypets/index.html#tkinter_cxfreeze)

See: <http://sebsauvage.net/python/snypets/index.html#py2exe>

For example, [*webGobbler*](#) uses Tkinter and was packaged into a Windows installer.

About this document

This document was written by Sébastien SAUVAGE, webmaster of <http://sebsauvage.net>
The address of this document is <http://sebsauvage.net/python/gui>
Last update: 2008-06-25.

If you find errors in this document, please report them.