第十三章 應用程式介面設計

我們開始介紹另一個圖形介面設計，Python另一個從C++語言延伸出的wxPython模組庫，專門用來開發應用程式。應用程式也就是我們平常所說的軟體，文書處理軟體如Word、Writer或Pages，網頁瀏覽器如I.E.、FireFox或Safari，還有其他各式各樣的軟體，可以進行不同的應用。

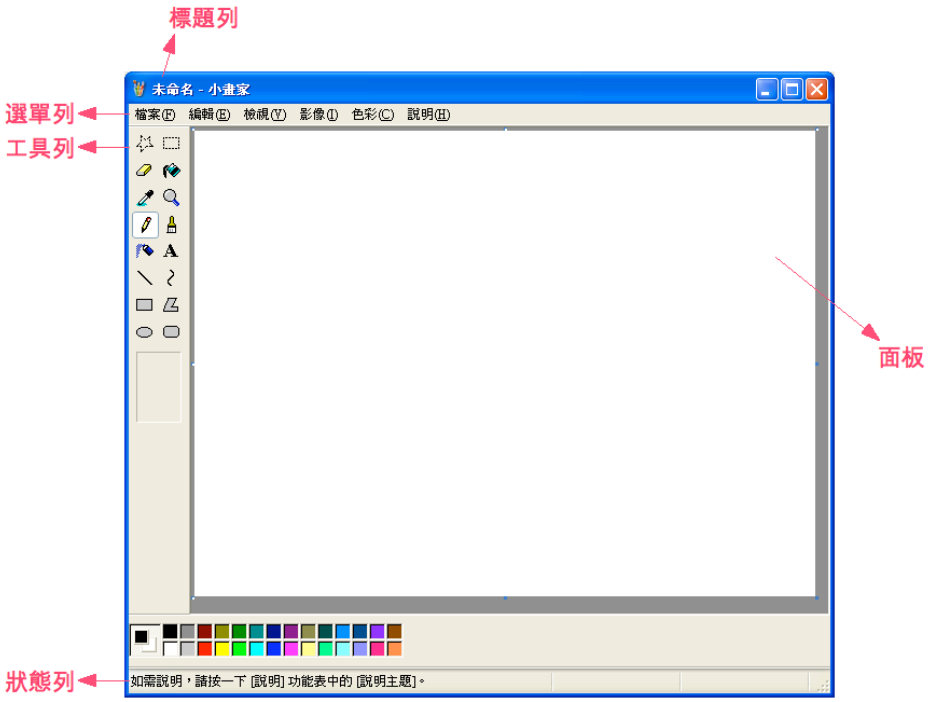
在利用Pygame寫遊戲的時候，由於遊戲中的物體如乒乓球中的球、球拍，或是象、虎、貓、鼠生存遊戲中四隻動物的移動，實際在螢幕上移動的是Pygame繪出的圖形或是我們提供的圖檔，他們在視窗中的顯示位置並不固定，所有的座標都是由計算得來的。

然而當我們使用軟體時，絕大多數應用軟體的情況很不一樣，大部分視窗所顯示的圖形都已經安排好位置，因而要進行一項操作時，常常是用將滑鼠點擊固定位置的按鈕，或是放置在選單的選項，而資料的處理也通常在某一特定的區域。

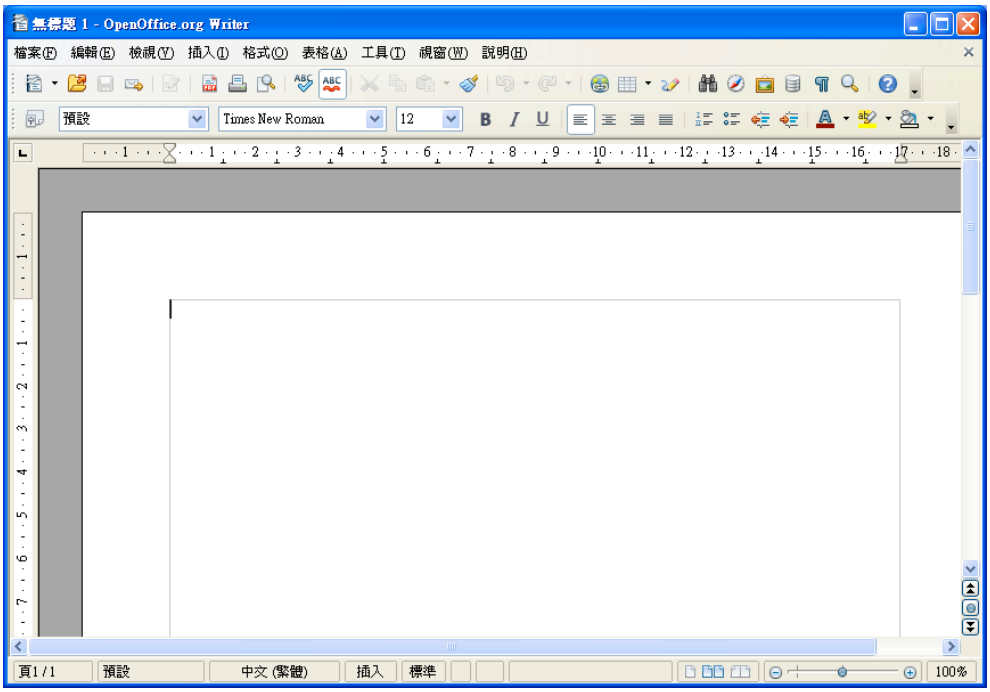
應用程式介面設計就像是放置選單、按鈕、捲軸、控制桿等等的位置，然後替每個控制項撰寫相對應的程式碼，使設計出的應用程式能夠發揮某種功能。圖形介面發展已久，所以對於視窗的各個元件都有習慣的稱法，我們先來看看實際的例子。

MS-Windows的小算盤是個典型的應用，我們可以看到從上依次為標題列、選單列，然後是文字方塊與按鈕合稱的「面板」。所謂的「面板」也就是我們處理資料實際的操作區域，文字方塊可分成顯示或是輸入兩種，小算盤的例子直接顯示使用者的輸入與計算結果。

我們再來看看另一個例子。



小畫家多了工具列及狀態列，工具列是常用功能的按鈕集合，狀態列則以文字方塊的模式顯示提示資訊，面板的地方也就是繪圖的畫布。小畫家的工具列放在視窗左邊，下方也有選取顏色的工具列，小算盤則是仿照我們平常所用的計算機排列控制項，這是說要設計應用程式介面時，我們可以依需求或使用者方便編排控制項。另一個典型的例子如Writer。



從下而下依序為標題列、選單列及兩行工具列，然後是面板區，視窗下方則是狀態列，附加一些控制項。接下來，我們先以一個簡單的「Say Hello!」視窗應用程式介紹如何運用Python語言及wxPython模組庫，進行應用程式介面設計。

Say Hello!

程式剛開始執行如下圖。

輸入名字後，按下〔Say Hello....〕的按鈕結果如下圖。

所有的程式碼如下。

```
#-*- coding: UTF-8 -*-

import wx
from time import asctime

weeks = {"Mon":u"星期一", "Tue":u"星期二", "Wed":u"星期三", "Thu":u"星期四",\
         "Fri":u"星期五", "Sat":u"星期六", "Sun":u"星期日"}
months = {"Jan":u"一月", "Feb":u"二月", "Mar":u"三月", "Apr":u"四月", \
          "May":u"五月", "Jun":u"六月", "Jul":u"七月", "Aug":u"八月", \
          "Sep":u"九月", "Oct":u"十月", "Nov":u"十一月", "Dec":u"十二月"}

class MyFrame(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(400, 300))
        panel = wx.Panel(self, -1)

        font = wx.SystemSettings_GetFont(wx.SYS_SYSTEM_FONT)
        font.SetPointSize(12)

        prompt = u"請輸入你的名字"
        vbox = wx.BoxSizer(wx.VERTICAL)

        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        st1 = wx.StaticText(panel, -1, prompt)
        st1.SetFont(font)
        hbox1.Add(st1, 0, wx.RIGHT, 14)
        self.name = wx.TextCtrl(panel, -1)
```

```
        hbox1.Add(self.name, 1)

        vbox.Add(hbox1, 0, wx.EXPAND | wx.LEFT | wx.RIGHT | wx.TOP, 12)
        vbox.Add((-1, 10))

        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        self.button = wx.Button(panel, 1, "Say Hello....")
        self.Bind(wx.EVT_BUTTON, self.ButtonClick, id = 1)
        hbox2.Add(self.button, 0)

        vbox.Add(hbox2, 0, wx.EXPAND | wx.LEFT | wx.RIGHT, 12)
        vbox.Add((-1, 10))

        hbox3 = wx.BoxSizer(wx.HORIZONTAL)
        self.message = wx.StaticText(panel, -1)
        self.message.SetFont(font)
        hbox3.Add(self.message, 0, wx.Right, 14)

        vbox.Add(hbox3, 0, wx.EXPAND | wx.LEFT | wx.RIGHT, 12)
        vbox.Add((-1, 10))

        hbox4 = wx.BoxSizer(wx.HORIZONTAL)
        self.time1 = wx.StaticText(panel, -1)
        self.time1.SetFont(font)
        hbox4.Add(self.time1, 0, wx.Right, 16)

        vbox.Add(hbox4, 0, wx.EXPAND | wx.LEFT | wx.RIGHT, 12)
        vbox.Add((-1, 12))

        hbox5 = wx.BoxSizer(wx.HORIZONTAL)
        self.time2 = wx.StaticText(panel, -1)
        self.time2.SetFont(font)
        hbox5.Add(self.time2, 0, wx.Right, 16)

        vbox.Add(hbox5, 0, wx.EXPAND | wx.LEFT | wx.RIGHT, 12)
        vbox.Add((-1, 12))

        hbox6 = wx.BoxSizer(wx.HORIZONTAL)
        self.time3 = wx.StaticText(panel, -1)
        self.time3.SetFont(font)
        hbox6.Add(self.time3, 0, wx.Right, 16)

        vbox.Add(hbox6, 0, wx.EXPAND | wx.LEFT | wx.RIGHT, 12)
        vbox.Add((-1, 12))

        panel.SetSizer(vbox)
        self.Centre()
        self.Show(True)

    def ButtonClick(self, event):
        name = self.name.GetValue()
        message = u"哈囉， " + unicode(name) + u" ！"
        self.message.SetLabel(message)

        moment = asctime()
        s1 = u"今天是西元" + unicode(moment[20:])+ u"年"
        s2 = months[moment[4:7]] + unicode(moment[8:10]) +\
                                              u"日" + weeks[moment[:3]]
        s3 = u"現在時間是" + unicode(moment[11:19])
        self.time1.SetLabel(s1)
```

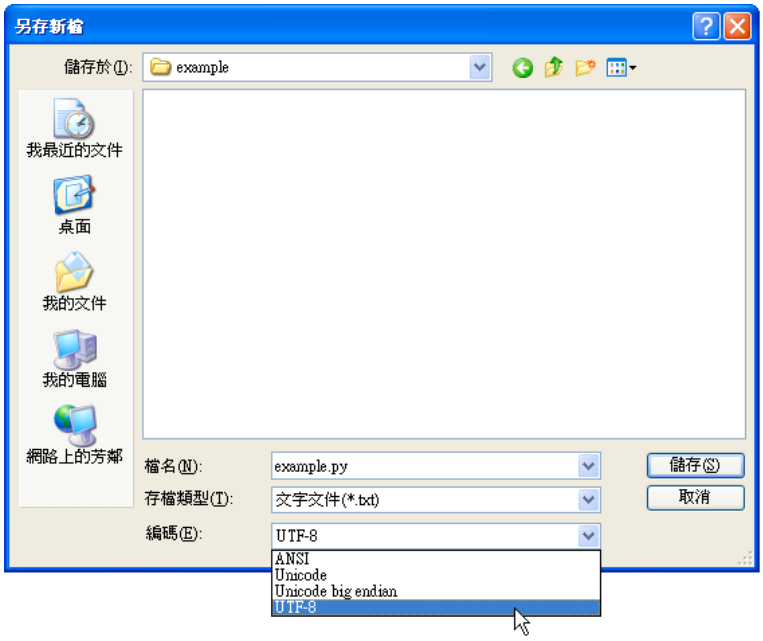
```
            self.time2.SetLabel(s2)
            self.time3.SetLabel(s3)

    if __name__ == "__main__":
        app = wx.App()
        frame = MyFrame(None, -1, title="Hello Test!!")
        app.MainLoop()
```

接下來我們逐步說明這個範例程式。

文件編碼及u前綴字串

為了更有效的處理中文字串，因此不論平台為MS-Windows、Linux或Mac，範例程式的純文字檔案我們都需要儲存為UTF-8編碼。在MS-Windows系統下，文件編碼要儲存為UTF-8格式，簡單的方式為利用記事本的另存新檔，如下圖。



在編碼下拉式選單選擇UTF-8即可。而在範例程式中，如

```
    weeks = {"Mon":u"星期一", "Tue":u"星期二", "Wed":u"星期三", "Thu":u"星期四",\
             "Fri":u"星期五", "Sat":u"星期六", "Sun":u"星期日"}
    months = {"Jan":u"一月", "Feb":u"二月", "Mar":u"三月", "Apr":u"四月", \
              "May":u"五月", "Jun":u"六月", "Jul":u"七月", "Aug":u"八月", \
              "Sep":u"九月", "Oct":u"十月", "Nov":u"十一月", "Dec":u"十二月"}
```

每個中文字串前都加上小寫的u，以及

```
    prompt = u"請輸入你的名字"
    message = u"哈囉， " + unicode(name) + u" ！"
    s1 = u"今天是西元" + unicode(moment[20:])+ u"年"
    s2 = months[moment[4:7]] + unicode(moment[8:10]) + u"日" + weeks[moment[:3]]
    s3 = u"現在時間是" + unicode(moment[11:19])
```

這些都是u前綴字串，字串的編碼格式為Unicode。但是我們需要注意一點，u前綴字串與利用內建函數unicode()轉換後的Unicode字串，這些全都屬於unicode型態，不同於一般字串的str型態。

此外，wxPython的下載安裝檔有ansi與unicode兩種編碼版本，顯示中文需要的是unicode的版本。

繼承結構

引入wxPython模組庫，只需要簡單的寫

```
import wx
```

這樣就引入wxPython模組庫了，我們先跳到程式實際執行的地方。

```
if __name__ == "__main__":
    app = wx.App()
    frame = MyFrame(None, -1, title="Hello Test!!")
    app.MainLoop()
```

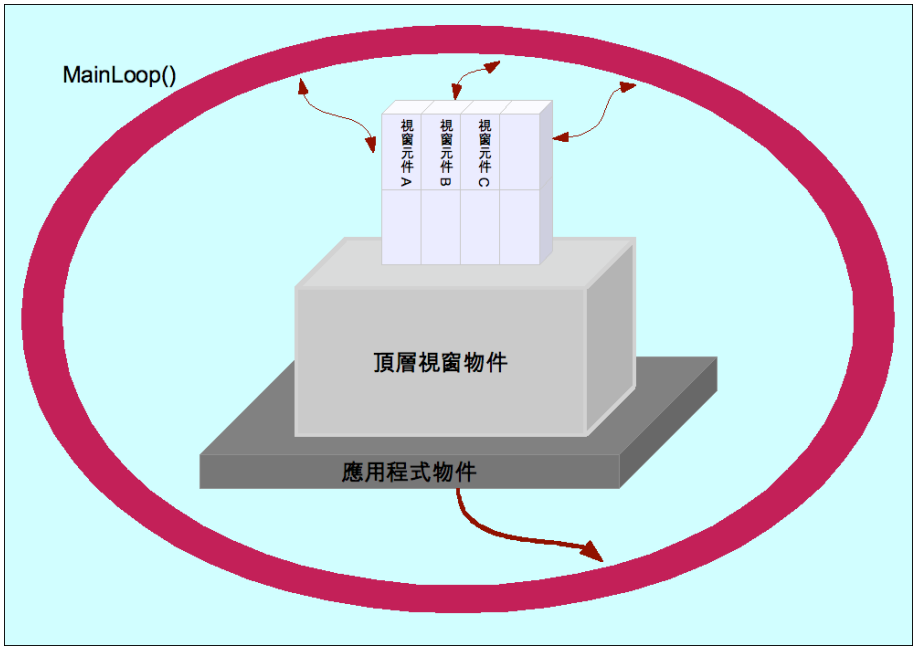
當wxPython的程式被執行時，我們需要建立一個應用程式的物件，這個例子中該物件被儲存在變數app之中，然後自訂一個型態囊括所有視窗元件，最後執行應用程式物件的MainLoop方法，藉此維持視窗圖形的顯示。

我們可以發現這跟上一章中，最後也是利用World型態的mainloop()方法一樣，把遊戲的主要迴圈放到該方法的定義裡面，遊戲的圖形顯示同樣以此維持，這些程式設計方法其實都是類似的（道理都是相通的）。

Pygame中的mainloop()方法是由我們自行定義，但是實際更新畫面是由display模組中的update()函數來控制，而wxPython維持圖形顯示是由MainLoop方法來進行，我們大可不必掌握這些程式介面詳實的情況，除了多半跟硬體有關，同時這些資訊已經隱藏到函數或方法中，我們需要信任發展維護這些模組庫的程式設計師，不僅僅是他們具有豐厚的經驗，事實上這些模組庫已經正確無誤的運作多年。

雖然我們說wxPython是模組庫，其內的確有些獨立的模組，然而我們實際運用卻會以繼承其型態為主。這是說，wxPython提供的是一套完整層級性的繼承結構，我們設計應用程式介面時，直接套用wxPython本身提供的結構，不僅開發更為快速，維護與繼續擴展也相形方便許多。

這樣的結構如下圖。



應用程式物件就是我們建立的app變數，這是程式建構的基礎，其上開始構築頂層視窗物件，我們以frame變數來建立，MyFrame型態繼承自wxFrame，接著繼續往上搭建所有的視窗元件。程式執行由應用程式物件依照MainLoop()方法，其中視窗元件可與MainLoop()迴圈產生連結，互相作用。

MyFrame

MyFrame型態直接繼承自wx.Frame。

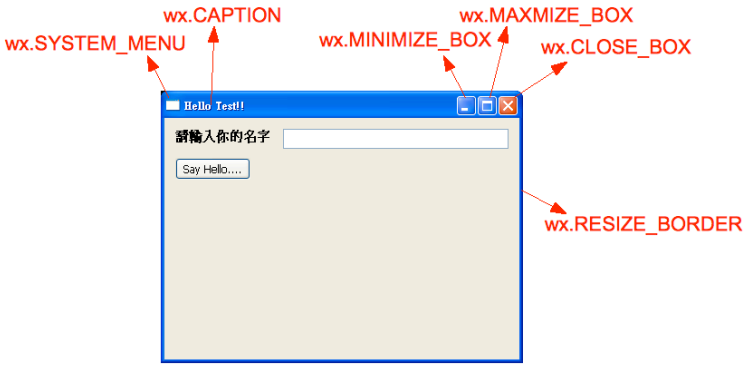
```
class MyFrame(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(400, 300))
```

wxPython習慣稱整個視窗為frame，其中文為框架或畫面的意思，在不影響理解的情況下，我們仍稱視窗為視窗，但是型態名稱依需求或習慣，仍以MyFrame或其他名稱定義。

wx.Frame總共有七個參數，如下。

```
wx.Frame(parent, id = -1, title = "", pos = wx.DefaultPosition, size =
wx.DefaultSize, style = wx.DEFAULT_FRAME_STYLE, name = "frame")
```

參數parent儲存從哪一種型態繼承而來的資訊；id儲存編號，這只有在建立多個視窗時才需要用到，預設值為-1；title是標題列的字串，預設為空字串；pos為視窗在螢幕顯示的位置座標；style為視窗的樣式，預設值為wx.DEFAULT_FRAME_STYLE，這是一組預設旗幟的集合，共有六個旗幟，如下圖。



wx.CAPTION用作設定標題列，wx.SYSTEM_MENU則需wx.MINIMIZE_BOX 、 wx.MAXMIZE_BOX 、 wx.CLOSE_BOX三個標籤都有設定才有作用，wx.RESIZE_BORDER則是控制視窗大小能否調整，預設旗幟會呈現出視窗的基本樣貌，然而更改旗幟的設定，會依平台而有不同的顯示結果。

最後一個參數name指出這是“frame”。但是只有前三個是必須的，我們在這裡多指定了參數size的值，將視窗大小設定為400×300。當然除了預設的wx.DEFAULT_FRAME_STYLE之外，還有許多其他種類的旗幟可供設定，如wx.FRAME_SHAPED會去除長方形藍色邊緣，而以下的設定，會使視窗如同工具列的外觀。

```
style = wx.DEFAULT_FRAME_STYLE | wx.FRAME_TOOL_WINDOW
```

視窗看起來如下。

接下來建立變數panel，panel就是面板的意思，這是要作為後面建立視窗元件的參數parent之用。

```
panel = wx.Panel(self, -1)
```

字型設定與顯示文字

建立調整字型的變數font，然後我們設定文字顯示的大小。

```
font = wx.SystemSettings_GetFont(wx.SYS_SYSTEM_FONT)
font.SetPointSize(12)
```

這裡我們將字型設定為系統預設字型，然後大小設為12。雖然說字型設定不是必要的，因為wxPython會自動載入系統預設的字型及大小，可能會有平台間的差異，在MS-Windows的預設字型就顯得有點小。

下面我們把所有顯示文字的視窗元件放在一起。

```
st1 = wx.StaticText(panel, -1, prompt)
st1.SetFont(font)

self.message = wx.StaticText(panel, -1)
self.message.SetFont(font)

self.time1 = wx.StaticText(panel, -1)
self.time1.SetFont(font)

self.time2 = wx.StaticText(panel, -1)
self.time2.SetFont(font)

self.time3 = wx.StaticText(panel, -1)
self.time3.SetFont(font)
```

用來顯示文字的視窗元件稱為StaticText，中文意思是靜態文字方塊。wx.StaticText同樣共有七個參數，如下。

```
wx.StaticText(parent, id, label, pos = wx.DefaultPosition, size =
wx.DefaultSize, style = 0, name = "staticText")
```

我們需要提供前三個參數，parent就是之前建立的變數panel，id都設為-1，因為目前還不需要id的控制，而label則為所要顯示的文字字串，prompt中儲存「請輸入你的名字」，這使程式執行後直接顯示出文字的提示訊息。

然而label不必建立時就設定，其中，self.message、self.time1、self.time2與self.time3因為沒有提供參數label，所以程式執行不會直接顯示。我們希望的是在按下按鈕才會顯示這些文字字串，不過我們仍須先建立這幾個變數，至於實際的顯示方法我們稍後再談。

另外還有一個視窗元件為TextCtrl，中文意思是文字控制方塊，這是用來輸入文字之用。

```
self.name = wx.TextCtrl(panel, -1)
```

使用BoxSizer來布局

Sizer並不是視窗元件，而是wxPython提供給視窗元件在視窗中編排的演算法，分為兩大類，格子狀或是盒狀，我們在範例中用到的BoxSizer就是盒狀的一種。以下的這一行，就建立了一個BoxSizer，儲存在變數vbox之中，我們給予的參數wx.VERTICAL使這個BoxSizer為縱向的。

```
vbox = wx.BoxSizer(wx.VERTICAL)
```

vbox如下圖的綠色框框。

這就好比我們做了一個具有深度的盒子，然後再做一些橫擺的盒子，可以放進具有深度的盒子中，同樣的，橫擺的盒子也能夠依水平方向可以放入新的東西。

```
hbox1 = wx.BoxSizer(wx.HORIZONTAL)
st1 = wx.StaticText(panel, -1, prompt)
st1.SetFont(font)
hbox1.Add(st1, 0, wx.RIGHT)
self.name = wx.TextCtrl(panel, -1)
hbox1.Add(self.name, 1)

vbox.Add(hbox1, 0, wx.EXPAND | wx.LEFT | wx.RIGHT | wx.TOP, 12)
vbox.Add((-1, 10))
```

hbox為我們第一個建立橫擺的盒子，這個盒子裡利用Add()的方法，先放進一個存放在變數st1中的StaticTex視窗元件，然後再一次的利用Add()方法，放進另一個存放在self.name的TextCtrl視窗元件中。

接下來，具有深度的盒子vbox也要利用Add()方法，把hbox這個橫擺的盒子放進去。Add()具有三種可以增加的東西，如下為所需的參數。

```
Add(window, proportion=0, flag=0, border=0, userData=None)
Add(sizer, proportion=0, flag=0, border=0, userData=None)
Add(size, proportion=0, flag=0, border=0, userData=None)
```

window就是視窗元件，如st1與self.name分別代表的StaticText與TextCtrl，sizer是其他的盒子，在我們的例子中就是其他的BoxSizer，size則是寬與高的空白範圍，我們第一個參數都給-1，因為作為間隔不需要寬度。proportion預設為0、userData預設為None，因為不會影響範例程式的執行效果，所以我們暫時不理會這兩個參數。border在第一個參數為sizer會發生作用，這會指定該水平BoxSizer盒子的高。

flag的作用類似wx.Frame的style參數，flag的中文正是旗幟的意思，這些旗幟是用來在盒子內，也就是在方框範圍對齊之用。從英文字面大約就可以了解這些旗幟的用途，wx.EXPAND的目的是讓視窗元件盡可能充滿盒子，這個效果在TextCtrl可以看到，無論我們如何調整視窗大小，TextCtrl都會隨著改變長度。

「|」運算子與or運算子相同，兩者都是「或」的意思。wx.LEFT向左對齊，wx.RIGHT向右對齊，wx.TOP則是向上對齊，由於都是用「|」運算子連接，所以只要最左方的旗幟能夠套用，就會直接採取最左方的旗幟，如果最左方的旗幟無法套用，就會依序往右找旗幟。

加入每個橫擺的盒子後，我們都利用Add()方法另外做出10畫素的間隔。接下來的盒子，從hbox2到hbox6，我們都以類似的方式加入到vbox之中。

最後，我們要把vbox放在面板中作設定，Centre()把視窗放在螢幕中央，Show()方法則是讓視窗顯示出來。

```
panel.SetSizer(vbox)
self.Centre()
self.Show(True)
```

按鈕事件

在hbox2之中放的是按鈕的視窗元件，建立過程很簡單。

```
self.button = wx.Button(panel, 1, "Say Hello....")
```

我們將所建立的按鈕儲存在self.button的變數之中，wx.Button所需的參數如下。

```
wx.Button(parent, id, label, size = wxDefaultSize, style = 0, validator, name
= “button”)
```

同樣需要七個參數，但是只有前三個是必須的，parent為所要繼承的型態，我們在這裡將id設為1，這是等一下用在按鈕事件中的序號，label則是按鈕上顯示的名稱。

要讓點擊按鈕產生反應，我們利用Frame中的Bind()方法，把點擊按鈕形成按鈕事件。

```
self.Bind(wx.EVT_BUTTON, self.ButtonClick, id = 1)
```

Bind()方法中三個參數，第一個為事件名稱，這個例子是按鈕事件，wxPython稱之為wx.EVT_BUTTON，第二個是我們要求按下按鈕後發生的事情，這裡我們另外定義ButtonClick()方法來操作，第三個則是id，這裡要跟self.button的id相同。

接下來我們看到ButtonClick()的定義。

```
def ButtonClick(self, event):
    name = self.name.GetValue()
    message = u"哈囉， " + unicode(name) + u" ！"
    self.message.SetLabel(message)

    moment = asctime()
    s1 = u"今天是西元" + unicode(moment[20:])+ u"年"
    s2 = months[moment[4:7]] + unicode(moment[8:10]) + u"日" \
                                                    + weeks[moment[:3]]
    s3 = u"現在時間是" + unicode(moment[11:19])
    self.time1.SetLabel(s1)
    self.time2.SetLabel(s2)
    self.time3.SetLabel(s3)
```

除了self，參數列要增加event。因為ButtonClick()的目的在於顯示打招呼以及現在時間的字詞，所以一開始便是從self.name得到使用者輸入的文字，這裡是用GetValue()方法，儲存到變數name之中。

然後利用字串連接建立我們所要顯示的訊息，儲存到變數message，再以self.message的SetLabel方法將訊息傳給StaticText作為label參數，使文字得以顯示。

顯示現在時間的作法也類似，moment利用asctime()函數抓取電腦的時鐘，然後s1、s2及s3分別是作為三個盒子的字串，在self.time1、self.time2與self.time3同樣利用SetLabel方法使文字顯示出來。