

# Migrating to PAGE 7

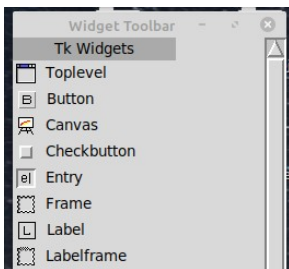
Written by G.D. Walters

## Introduction

The purpose of this document is to help users of PAGE 6.2 and below to quickly be able to find their way through the changes that make up PAGE 7.

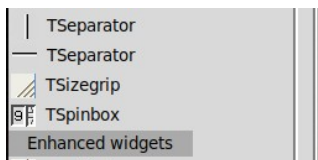
## Changes to the user interface

At first glance, you might not notice any difference in the way that PAGE looks. However, there are two big changes to the Widget Toolbar or Toolbox.



The first change that you will see is that the Toplevel widget is back in the toolbox. Somewhere around PAGE version 4.10 the Toplevel icon was removed from the toolbar. This is a big change for PAGE. In previous versions you could create projects that supported multiple forms, but each form needed to be set up as a separate project. So if you created a project with five different screens or forms, you needed to create five different projects, each having three files (minimum) for each project for a total of 15 project files. This could be very hard to manage especially when you start to include image files and other support files. This also created other issues for users wanting to create installers. Now with PAGE version 7, there are only three files created, no matter how many forms your project might contain. This can be both a blessing and a curse. We'll discuss this in a few minutes.

The other obvious change is that there is now support for the ttk Spinbox.

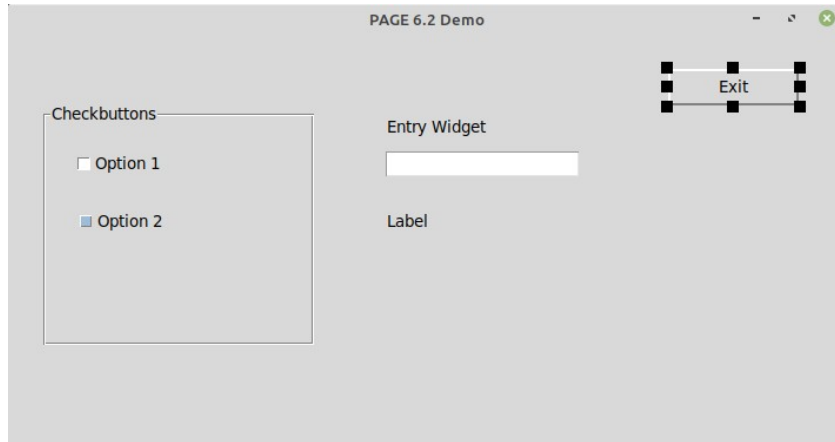


## Changes to the Code Generator

The real changes (and benefits) are in the code generation processes, where PAGE converts the Tcl instructions into Python code.

Let's compare a simple project created in PAGE 6.2 to the same project created in PAGE 7. In this project, there is

only a single form which contains an Exit button, a couple of Checkbuttons inside a Labelframe, a couple of Labels and an Entry widget. Many of these widgets have support text variables that allow for interactive control of the project. For example, the Entry widget has a text variable that holds whatever has been typed into the widget. The Label widget below it, has a different text variable that will change the text of the label.



When the Python code is generated, these variables are stored in the **support.py** file in a function called **set\_Tk\_var()**. You can see the generated function here...

```
def set_Tk_var():
    global che48
    che48 = tk.IntVar()
    global tch49
    tch49 = tk.IntVar()
    global EntryData
    EntryData = tk.StringVar()
    global EntryDisplay
    EntryDisplay = tk.StringVar()
    EntryDisplay.set('Label')
```

These variables were always globals and available anywhere within the **support.py** file. Another function that PAGE created was the **init** function.

```
def init(top, gui, *args, **kwargs):
    global w, top_level, root
    w = gui
    top_level = top
    root = top
```

This was the last bit of code to be run before the form was shown to the user. So if you needed any start up code to be run, you would put it at the end of the **init** function. Notice the line **w = gui** in the function. The variable **w** is used whenever you want to refer to a widget directly that is on our form. For example, if you wanted to change the colour of the Exit button you could reference it like this:

```
w.btnExit.configure(background='cadetblue3')
```

You could also create an easy alias to an existing widget by doing something like this:

```
global treeview
treeview = w.Scrolledtreeview1
```

After that, you could refer to the **Scrolledtreeview** widget simply as **treeview**.

Now that we've taken a quick look at how things are in PAGE code before version 7, let's take a look at what you can expect from the new version. I opened the .tcl using version 7, saved it and generated the two Python modules. Now let's take a look at the new support module.

The first thing you should notice is that there is no support for Python 2.x. PAGE 7 will only run under Python 3.x. Another thing that should be obvious is that the constants library from tkinter is imported by default. This allows you to reference the tk constants much easier.

```
import sys
import tkinter as tk
import tkinter.ttk as ttk
from tkinter.constants import *

import p7demo

def main(*args):
    '''Main entry point for the application.'''
    global root
    root = tk.Tk()
    root.protocol( 'WM_DELETE_WINDOW' , root.destroy)
    # Creates a toplevel widget.
    global _top1, _w1
    _top1 = root
    _w1 = p7demo.Toplevel1(_top1)
    # Your start up code goes here
    root.mainloop()
```

Notice that there is no longer a function called **init**. It is now called **main**. Just like **init** did, it sets up everything that needs to be done to create a workable Tkinter program. Instead of putting your start up code at the very bottom of the function (if needed), you need to put it right before the last line.

Something else you need to be aware of is there is no alias **w** to the **gui** python module. It's now **\_wX** and **\_topX**. In the example above it is **\_top1** and **\_w1**, but these numbers can change depending on a number of things.

If you need to migrate an older program to PAGE 7, it's usually just a matter of doing a global find and replace for **w**. to **\_w1**. or whatever the alias is.

One other thing that PAGE 6.2 and below offered is a function to end the PAGE created application code called **destroy\_window()**.

```
def destroy_window():
    # Function which closes the window.
    global top_level
    top_level.destroy()
    top_level = None
```

Now all you need to do is call **root.destroy()** to properly close your PAGE application.

The final thing you should have noticed is there is no **set\_Tk\_var** function. In fact, none of the global textvariables

are listed. I have to admit, when this change was made, Don forgot to tell me and I sent in a number of bug reports complaining about the missing function. Not to worry, it's been moved to the **gui** file...sort of.

Here's what the function in PAGE 6 looked like...

```
def set_Tk_var():
    global che48
    che48 = tk.IntVar()
    global tch49
    tch49 = tk.IntVar()
    global EntryData
    EntryData = tk.StringVar()
    global EntryDisplay
    EntryDisplay = tk.StringVar()
    EntryDisplay.set('Label')
```

Just in case you are wondering, yes, there are two Checkbuttons, one Tk Checkbutton and one ttk::Tcheckbutton. That's why one has the variable che48 and the other tch49. Now in PAGE 7, you will find it inside the gui class...

```
class Toplevel1:
    def __init__(self, top=None):
        '''This class configures and populates the toplevel window.
           top is the toplevel containing window.'''
        ....
        top.geometry("713x349+423+276")
        top.minsize(1, 1)
        top.maxsize(4225, 1410)
        top.resizable(0, 0)
        top.title("PAGE 6.2 Demo")
        top.configure(highlightcolor="black")
        self.top = top

        # HERE IT IS !!!
        self.EntryData = tk.StringVar()
        self.EntryDisplay = tk.StringVar()
        self.che48 = tk.IntVar()
        self.tch49 = tk.IntVar()
```

All you have to do to reference these variables from your support file is something like this...

```
_w1.EntryDisplay.set('Here is what you typed')
_w1.che48.set(1)
```

And so on. The reason for this is pretty straight forward. PAGE 7 now supports multiple forms in the same code set. This means that if you have a project that has 5 forms, instead of having 15 code files, you now only have three. Since all the code for all of your forms are all in the same file, there is a very big chance that you could have the same widget alias (like Label1) in more than one form. By adding the \_w? to all the alias and Tk vars, there is no risk of this happening. We'll discuss the multiform projects shortly.

One other small "gotcha" that you should be aware of. When you use a ttk widget like the ttk::TEntry widget and assign a text variable you will see something like this...

text var	::vTcl::EntryData	<input checked="" type="checkbox"/>
Validate	none	<input type="checkbox"/>

That seems to be an artifact in Visual Tcl that Don uses to create PAGE. It only shows that way in the Attribute Editor and when you see the widget on the designer form. In code, it is saved and generated as simply “EntryData” and as a tk.StringVar. Don is still trying to find a way around this and hopefully will have a fix by the time PAGE 7.1 comes out.

Now you might wonder “WHY the big changes? These things worked in PAGE for well over 10 years! WHY NOW?!?!?”

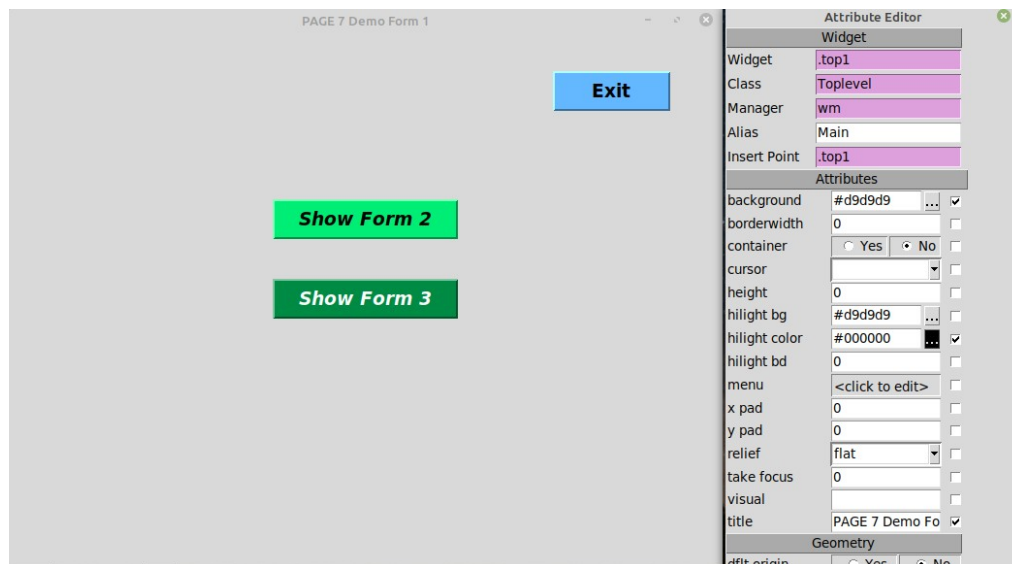
We’ll discover that next.

## PAGE Projects With Multiple Forms

Many projects created with PAGE work fine with just a single form. However, there are many times that you need two, three or more forms to get the job done. In PAGE 6.2 and before to do this, you had to create each form separately, which created three file for each form, then tie them all together with code to call each form when needed. Many times, you needed to minimize or hide one form to show the next form and then hide the new form and restore the old form to continue working. While this worked well, the number of files grew quickly and became somewhat difficult to maintain the code.

With PAGE 7, you create the entire project in one set of files. The GUI.py file holds all the forms for the project and the SUPPORT.py file holds all the code to make the project work.

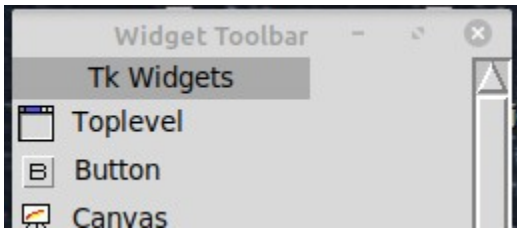
Let’s see how this works. First, create a simple form that will act as our main form.



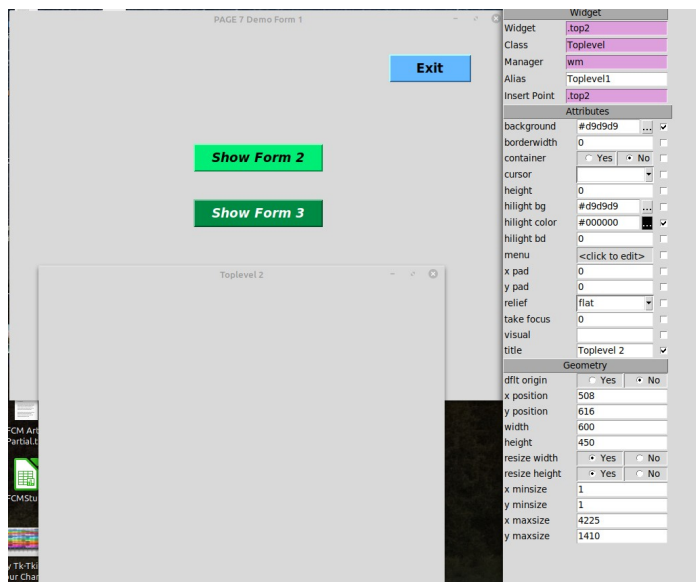
We’ll just put three buttons on our form. Notice that I have created an alias for this form as “Main”. Creating an alias for all your Toplevel forms is something you should do in all of your projects when using PAGE 7. This will help you determine which form is which in the GUI.py file module. Create an Exit button, one with the text of

“Show Form 2” and the other to “Show Form 3”. Set your commands to “on\_btnExit”, “on\_btnShowForm2” and “on\_btnShowForm3”.

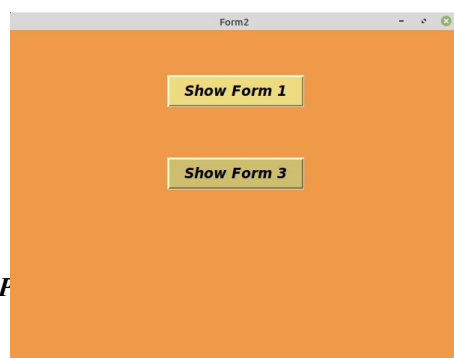
Next, click on the Toplevel button in the toolbox to add another form.



This creates a new form along with your Main form.



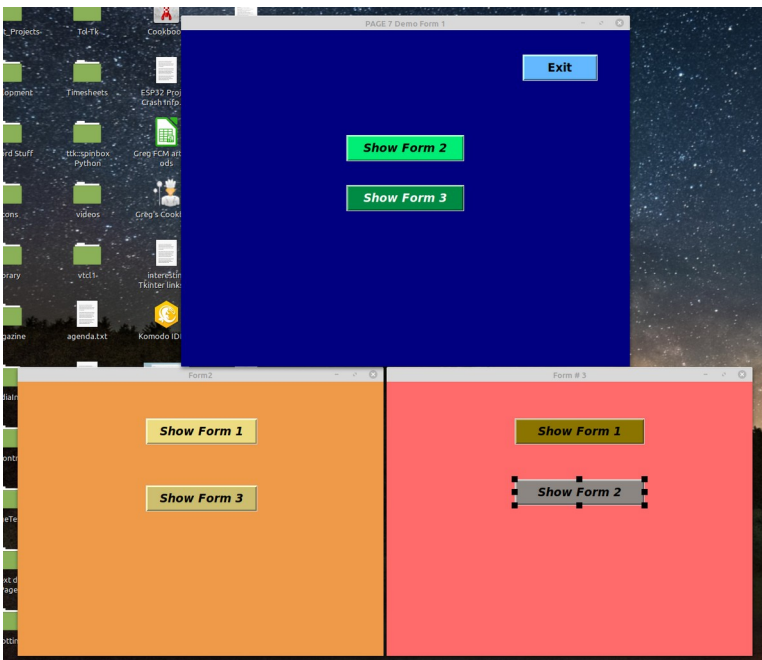
Alias this new form as Form2. Now add two buttons and set the text to “Show Form 1” and “Show Form 3” and set the commands for each button as “on\_btnShowForm1From2” and “on\_btnShowForm3From2”. You might want to also change the background colours for the form and the two buttons.



Again, click the Toplevel button in the toolbox which will add another form. Alias this form as “Form3” and add two buttons to this form. Set their text attributes to “Show Form 1” and “Show Form 2”. Set the commands to “on\_btnShowForm1From3” and “on\_btnShowForm2From3”. Set the colours as you wish.



The last thing you should do is move your forms to where you want them to show when the program runs. For this part of the project, I simple moved the forms around so that all three are visible at the same time.



Now save your project as something like “Multiform1.tcl”. When you run your project under Python, all three

forms show up in the same place you left them when you saved your project.

## Controlling The Forms In Code

There are many times that you want your project to be able to show a single form at a time, which can be controlled by the code as needed. We'll look at doing that next. This also gives us a good introduction into the new code that PAGE 7 generates. Let's look at the support file to start.

```
def main(*args):
    '''Main entry point for the application.'''
    global root
    root = tk.Tk()
    root.protocol( 'WM_DELETE_WINDOW' , root.destroy)
    # Creates a toplevel widget.
    global _top1, _w1
    _top1 = root
    _w1 = Multiform1.Main(_top1)
    # Creates a toplevel widget.
    global _top2, _w2
    _top2 = tk.Toplevel(root)
    _w2 = Multiform1.Form2(_top2)
    # Creates a toplevel widget.
    global _top3, _w3
    _top3 = tk.Toplevel(root)
    _w3 = Multiform1.Form3(_top3)
    root.mainloop()
```

You can quickly see that all three forms are created before the form is shown. Each form has a **\_w?** variable associated with it and a **\_top?** variable as well. You should use the **\_top?** variable to replace any calls you would normally make to **root**, since root is the very first form (Main in this case).

When I designed this demo, I intended on only showing one form at a time. To do this, we'll utilize a set of functions I wrote a number of years ago (slightly modified) called **show\_me** and **hide\_me**. Let's look at the original functions.

```
def show_me():
    global root
    root.deiconify()

def hide_me():
    global root
    root.withdraw()
```

You can see, the functions are very simple. The **hide\_me** function simply causes the form to become minimized or hidden and the **show\_me** function restores the form to it's last position. We will have to change the names of the functions a little bit and change the root definitions to whatever form we are working with.

To show and hide form #1, we'll use the following code...

```
def show_me1():
    global _top1
    _top1.deiconify()
```



```
def hide_me1():
    global _top1
    _top1.withdraw()
```

In the original code, we referenced root. In this case, we need to reference **\_top1**. The rest of the functions are below and they each reference the **\_top?** that the routine is associated with.

```
def show_me2():
    global _top2
    _top2.deiconify()
```

```
def hide_me2():
    global _top2
    _top2.withdraw()
```

```
def show_me3():
    global _top3
    _top3.deiconify()
```

```
def hide_me3():
    global _top3
    _top3.withdraw()
```

Simple enough. Now in the callback command functions for each button, we need to make some form specific additions. We'll look at the two callbacks for form 1 (Main).

```
def on_btnShow2(*args):
    print('Multiform1_support.on_btnShow2')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    hide_me1()
    show_me2()
```

```
def on_btnShow3(*args):
    print('Multiform1_support.on_btnShow3')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    hide_me1()
    show_me3()
```

In each callback, the first 4 lines are provided by PAGE. We simply call the **hide\_me1** function to hide form 1 and then call the proper **show\_me** function for the form we want to show.

Let's press pause for a moment and look at the actual skeletons that PAGE 7 creates for callbacks.

```
def on_btnShow2(*args):
    print('Multiform1_support.on_btnShow2')
    for arg in args:
```

```
    print('another arg:', arg)
    sys.stdout.flush()
```

In earlier versions of PAGE, it was simply a function definition, a line that printed the function name and the `sys.stdout.flush()` call to make sure the information was actually printed to the terminal window. So what is the reason behind the `(*args)` and the `for` statement printing out the arguments? This is now standard for callbacks and allows you the ability to pass information back to the callback easily to the callback function. If you don't pass anything into the callback, that's fine. It will simply print the function name.

```
test1_support.on_btnExit
```

However, if you have, let's say two buttons that you want to share the same callback function but pass the number of the button in to the callback, you can set the `command` attribute to

```
lambda : on_btn1(1)
```

for the first and

```
lambda : on_btn1(2)
```

for the second. When you run the program and click the first button, you will see that the print routine in the callback will print to the terminal

```
test1_support.on_btn1
another arg: 1
```

and when you click the second button, you will see

```
test1_support.on_btn1
another arg: 2
```

When you are through with your coding and debugging, you can simply comment the five lines out or just delete them.

Now back to the program. You might have wondered why we named the callbacks `on_btnShowForm1From2` and `on_btnShowForm1From3` so strangely. Since all the callbacks are in the same support module, we need to be sure that we don't accidentally repeat a function name or make a callback function name that is confusing to us. It's always a good idea to keep your function names fairly self documenting.

```
def on_btnShowForm1From2(*args):
    print('Multiform1_support.on_btnShowForm1From2')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    hide_me2()
    show_me1()
```

```
def on_btnShowForm1From3(*args):
    print('Multiform1_support.on_btnShowForm1From3')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    hide_me3()
    show_me1()
```

```
def on_btnShowForm2From3(*args):
    print('Multiform1_support.on_btnShowForm2From3')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    hide_me3()
    show_me2()

def on_btnShowForm3From2(*args):
    print('Multiform1_support.on_btnShowForm3From2')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    hide_me2()
    show_me3()
```

Now we need to modify the Main function to hide forms 2 and 3 and just show the main form. This time, the lines that need to be added are shown in bold face with the code that doesn't get modified is shown in regular face.

```
def main(*args):
    '''Main entry point for the application.'''
    global root
    root = tk.Tk()
    root.protocol('WM_DELETE_WINDOW', root.destroy)
    # Creates a toplevel widget.
    global _top1, _w1
    _top1 = root
    _w1 = Multiform1.Main(_top1)
    # Creates a toplevel widget.
    global _top2, _w2
    _top2 = tk.Toplevel(root)
    _w2 = Multiform1.Form2(_top2)
    # Creates a toplevel widget.
    global _top3, _w3
    _top3 = tk.Toplevel(root)
    _w3 = Multiform1.Form3(_top3)
    hide_me2()
    hide_me3()
    root.mainloop()
```

It's just the two lines before **root.mainloop()**. They hide forms 2 and 3.

Last but not least, we need to modify the callback for the Exit button. This way we close the application properly. All we have to do is add the line **root.destroy()**.

```
def on_btnExit(*args):
    print('Multiform1_support.on_btnExit')
    for arg in args:
        print('another arg:', arg)
    sys.stdout.flush()
    root.destroy()
```

When you run the project, you will only see Form #1 until you click one of the buttons. Then you can switch between forms at will.

Since PAGE saves the position of each form when you save the designer session, that might not be the way you want it to be displayed to your user. Writing a function to centre each form once it's displayed is simple. Just remember not to use the root variable, but the `_top?` variable.

There are many other smaller changes in PAGE 7, but those can be found through the documentation or your testing.

I sincerely hope this document will help you in your journey from PAGE 6.x to PAGE 7.

*Greg Walters*