

Using PAGE 8.x

By Greg Walters



Table of Contents

Copyright.....	4
Introduction.....	5
Theme Support.....	6
What is a theme?.....	7
Using the page-legacy theme.....	8
Updating a legacy project.....	10
Theme Chooser.....	11
The Norwegian Lift Feature.....	13
TNotebook Tab Positions.....	16
Copy and Paste of menubars and popup menus.....	19
Changing the themes in your application.....	20
Imports.....	21
Program level global variables.....	21
The startup function.....	22
The load_tcl_themes function.....	23
The Callback for the Combobox.....	26
The fix_all_widgets function.....	26
The get_Toplevel_Children function.....	29
Hacking Themes.....	30
PAGE 8.x Theme Gallery.....	33
Adding third-party themes to PAGE 8.x.....	45
Getting Help for PAGE 8.x.....	47

Copyright

This document is copyright © 2023, 2024 by Gregory Walters.

All rights reserved worldwide 2023, 2024 by Gregory Walters.

This document is licensed under the CC BY-SA 4.0 license

(<https://creativecommons.org/licenses/by-sa/4.0/>)



Introduction

Welcome to the new and improved PAGE. There are many changes in PAGE 8.x, and I will try to discuss all of them. Since this new release of PAGE is the “.0” version, I will try to update this document whenever new features are added in the PAGE 8.x family.

The changes for PAGE 8.0 are as follows

- Support for ttk Themes
- Stacked Frame support (aka The Norwegian Lift)
- Ability to change the TNotebook Tab positions
- Support for copy and paste of Menubars and popup menus in borrow
- Many “behind the scenes” fixes and enhancements

Each of the top four changes will be discussed in sections that follow.

•Note:

This document is designed to be a companion document to the Learning PAGE tutorial, not a replacement.

I have made the assumption that you already have used PAGE somewhere in the past and already have installed PAGE 8x. I won’t go through how to install PAGE, how to get started with PAGE in general as a first time user.

Theme Support

Unlike earlier versions of PAGE, PAGE 8.x allows a programmer who wants to design a GUI that uses ttk widgets, to be able to see how the widgets will look using various themes in real time. In earlier versions of PAGE, all widgets were rendered on the screen using the “OS default” theme, not the theme named “default”. Under Linux, the OS default theme is named “default”. Under Windows the OS default is “winnative” and under Mac OS X the OS default is “aqua”, but both Windows and Mac OS X machines have access to the “default” theme.

Operating system themes can become confusing, since Linux has 4 system themes, Mac OS X has 5 and Microsoft Windows has 7 system themes. I’ve created a table that might help explain what themes are available for which system.

Operating System	Themes							
	clam	alt	default	classic	aqua	winnative	vista	xpnative
Linux	X	X	*	X				
Mac OS X	X	X	X	X	*			
Windows	X	X	X	X		X	*	X

Table 1: Built-in Themes by Operating System

In Table 1, you will see every OS Theme. The ones with an “X” in the cell, means that theme is available. An “*” denotes that theme is the “default” theme (even if it isn’t named ‘default’).

In addition, PAGE comes with some other themes as well. At the time of this writing, the list of themes are:

- clearlooks
- notsodark
- page-dark
- page-legacy
- page-light
- page-notsodark
- page-wheat
- waldorf

The notsodark and all the themes that start with “page-” were written by me. One of the things that makes them special is that they have a special graphical support to modify the TCheckbutton and TRadiobutton to make them look nicer. You can see a screenshot of all the themes that PAGE supports in the section “PAGE 8.x Theme Gallery”.

What is a theme?

Simply put a theme is a group of styles that provide a way of changing the look and feel of the various ttk widgets. Each widget has a style that is applied to it when the theme is loaded. That's why the ttk widgets don't have as many attributes that you can change as the Tk widgets do.

When you start PAGE 8.x, you have immediate access to all the themes that Page supports as well as the OS Themes for your machine. However, unless you are using one of the system default themes, you will not be able to run your project under Python (even in PAGE) until you have a folder named "themes" in your project folder. The folder must contain the theme or themes you want to use for your project.

To change the Theme being used, access the dropdown on the PAGE Main window and select the theme you want to try and see how that theme will change the look of the widgets you want to use. It also allows you to properly size various widgets that might change their size under different themes, so your form is ready before you try to run your GUI outside of PAGE.

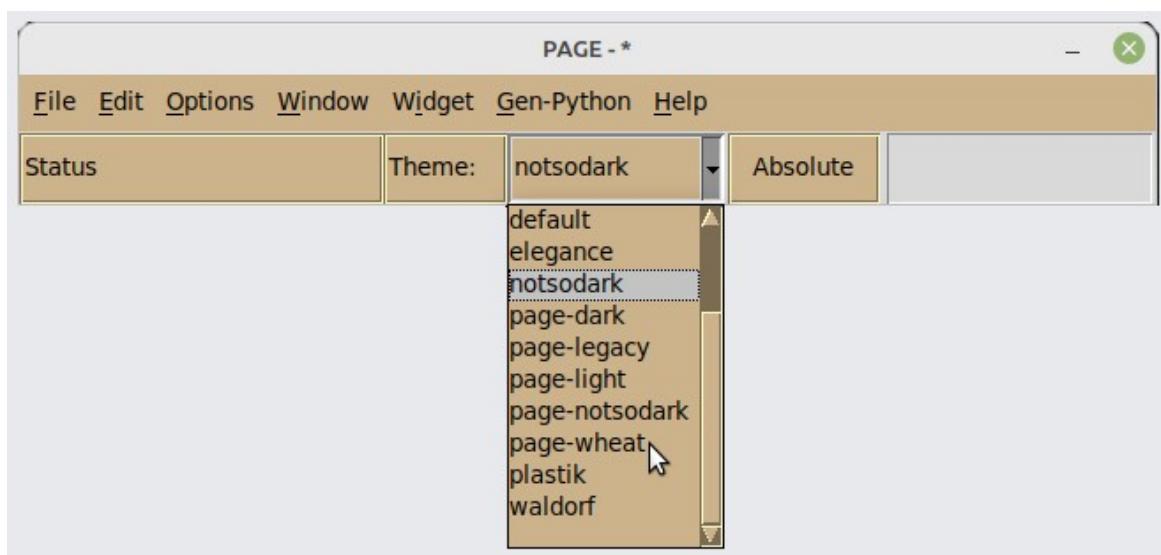


Figure 1: PAGE 8.x Theme Selector

If you want to (or need to) try your project directly under Python, you will need to consider the fact that the theme you like might not be available to Python yet. If you are using a "default" OS theme (like "clam" or "default"), you are all ready to go. On the other hand however, you are using one of the other themes, like "clearlooks" or "notsodark", you will need to copy the theme specific files into your local theme folder from your PAGE distribution folder. Some themes like "waldorf" are self contained so you only need to copy the file "waldorf.tcl" into your local theme folder. On the other hand, if you want to use a theme like "clearlooks", you will need to copy not only the "clearlooks.tcl" file, you will also need the graphics folder that goes along with it. In the case of "clearlooks.tcl", you will need the folder "clearlooks" as well.

If you **REALLY** want to, you can copy the entire theme folder from the PAGE distribution into your local project folder. The themes folder is around 1.8 MB in size and unless there is a real reason to, you should just simply copy the theme(s) you really need to your local themes folder.

You will find a number of themes that start with “page-”, like ‘page-wheat’, ‘page-dark’ and so on. These themes were created for use with PAGE, but you can use them in a standard Tcl project. The main thing that sets these themes apart from others is that they use a special graphic for the TCheckbutton and TRadiobutton. This is also why you need to be sure that you copy the theme folder as well as the theme.tcl file.

If you want to set a default theme so when PAGE starts up, your theme is already to go, you can change this in the Preferences menu.

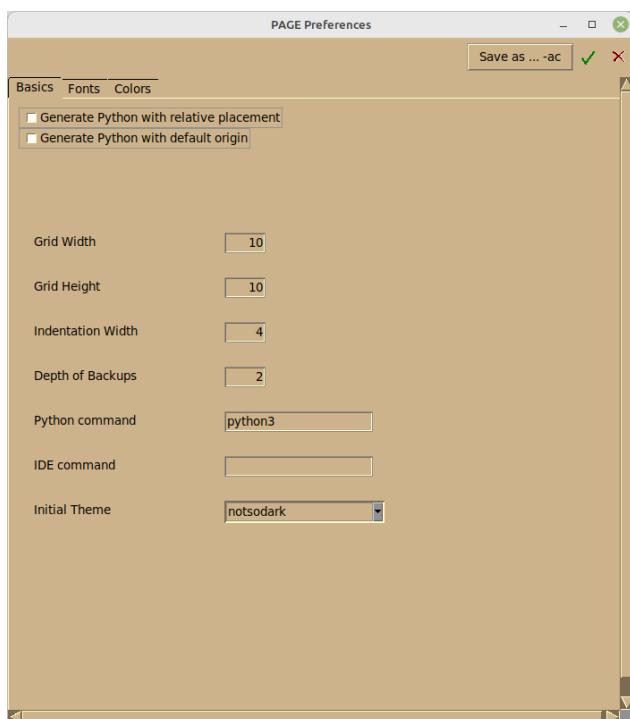


Figure 2: PAGE 8.x Preferences menu

All you have to do is select the theme that will become your default theme in PAGE, from the dropdown menu near the bottom of the window.

By the way, all the screenshots in this document were taken from my PAGE instance, so I have a brown background. Don uses a background of “wheat”. You can change the PAGE background on your machine by clicking on the ‘Colors’ tab of the Preferences window under ‘PAGE Colors’.

Using the page-legacy theme

•NOTE:

The page-legacy theme was designed to be very close to the “default” OS theme, but with some special code that allows users of earlier versions of PAGE to be able to support the special “GUI colour” settings in the preferences menu.

Unfortunately, this has made the page-legacy not quite compatible with “normal” tcl themes.

PAGE allows you to setup not only the colours that will be used by PAGE as the background of the PAGE windows, you can also set a default background and foreground colour to every widget both Tk AND ttk within the Preference menu under the Colors tab. For example, you might want to have a blue background colour and yellow foreground colour.

The page-legacy theme was designed to allow you to continue that in PAGE 8.x. The page-legacy theme is based on the theme named ‘default’, so no special styling has been done on any of the ttk widgets.

Be sure to set page-legacy as your default theme in the preferences menu.

Updating a legacy project

There are two paths that you can take to update one of your existing projects tp PAGE 8.x.

First is to open a copy of the project in PAGE 8.x, the immediately do a File|Save As and save the project under a new name. Then generate the GUI and SUPPORT modules, which will be created under the new filename.

The second would be to open PAGE 8.x with a new filename for your project then borrow the Toplevel(s) and paste each into your project.

The downside of either of these methods is that you will have a brand new SUPPORT module. However, you will find that you can pretty much copy most of your existing code into the new SUPPORT module.

Theme Chooser

Picking a theme for your GUI can become somewhat overwhelming with as many themes that come with PAGE “out of the box”. Don has included another feature to make this a bit easier for you.

You can access this from the PAGE Main Menu window, Window | Theme Chooser.

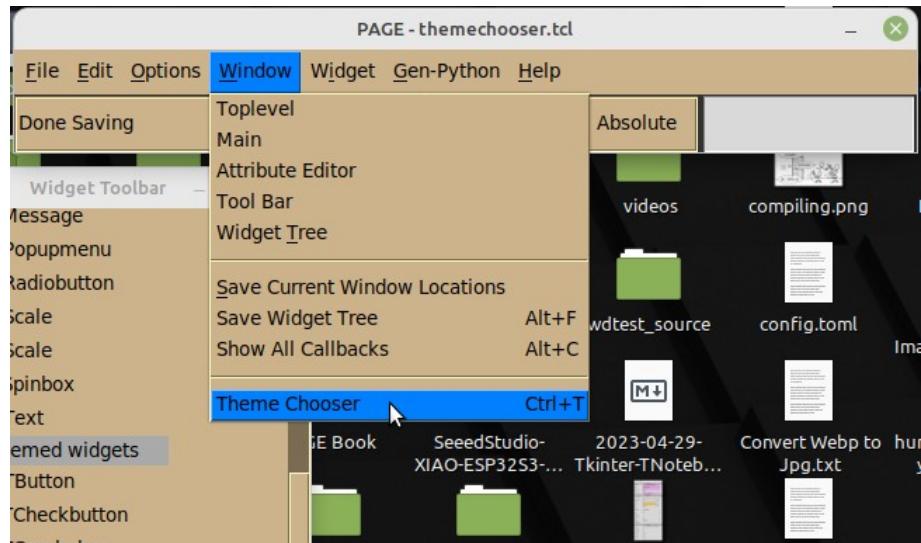


Figure 3: PAGE 8.x Theme Chooser Feature

When you select the Theme Chooser feature, a new window will popup over your project showing most of the ttk widgets that PAGE supports.

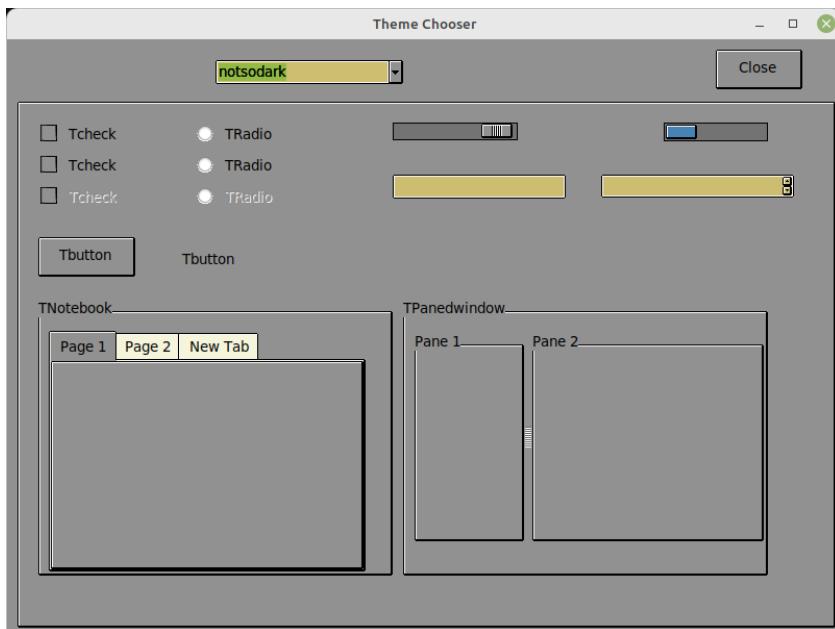


Figure 4: PAGE 8.x Theme Chooser Demo Form

To change the theme, simply click on the Combobox near the top center of the form. You can select any of the themes that PAGE currently has in the themes folder of your PAGE distribution. As you change the themes in the chooser, the theme for your project will change as well.

The Norwegian Lift Feature

Another new feature of PAGE 8.x, was code named “The Norwegian Lift”. This was named by Don after a feature request from my friend Halvard, who happens to live in Norway.

The request that Halvard sent me was based on one of his latest programs. He had created a GUI that had multiple screens in, for PAGE users, a rather unusual fashion.

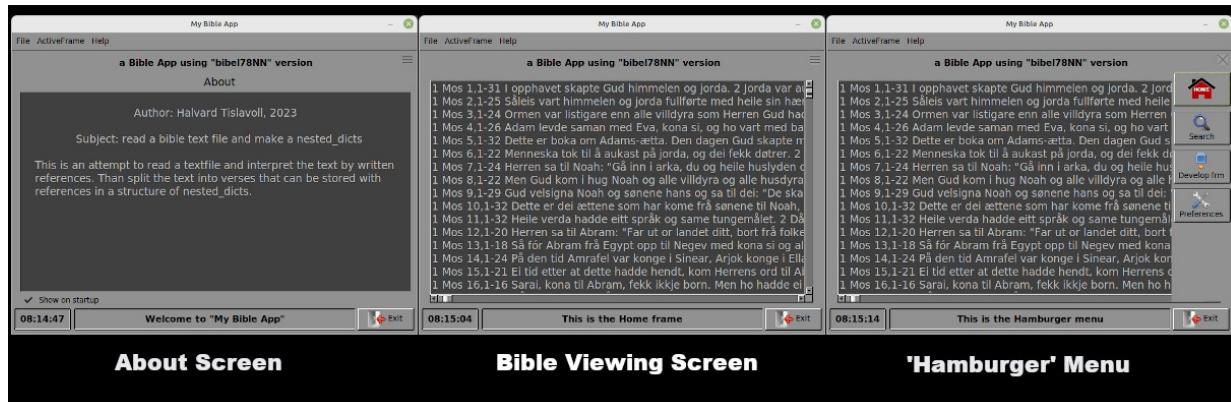


Figure 5: Halvard's Bible Viewer Program (in progress)

I know the image is fairly small, but the GUI has a “hamburger” menu (seen on the right) which is a frame with four buttons on it. Clicking on one of the buttons, will (as most menus do) cause another frame to appear, covering the previous frame. For example, the left view shows the About screen and the center view shows the actual Bible viewer portion.

What is unusual is that the main Toplevel form holds (in the centre section) four frames, each exactly the same size and are place at exactly the same X/Y location. To get the frames to show “on command”, he uses the lift method for each of the frames in code.

Now, the problem for Halvard was that he had to design each of the frames, one at a time in separate PAGE sessions and save each session as a separate project. Then when he was ready with the main Toplevel form that acts as the background for all the rest of the program, he needed to borrow each frame project, again one at a time, move that frame to the correct location then add the next frame project and so on. So his request was a way that PAGE could handle all of this in one session. So that when you clicked on a frame that held various child widgets, it would come to the top of any other widgets that might be on the Toplevel widget. When I presented this to Don, he sounded excited about the prospect and I left him to it, letting Halvard know that I had passed the request on to Don.

A couple of days later, Don sent me a new alpha cut of PAGE 8.x saying that this new cut included “the Norwegian Lift”. I have to say that I got quite a chuckle from this statement. The name really should be something like “Stacked Frames”, the name stuck. This, forevermore, will be known in my heart as “The Norwegian Lift”!

It's remarkably easy to use. Take this small demo I created as an example...

In PAGE, simply create a frame, add the widgets that you need. I aliased it as TFrame1 and added a TLabel. I added another TFrame slightly offset from the other (so PAGE doesn't think that you are trying to add the second frame as a child of the first frame), use the Attribute editor to set the x,y position and the width and height to be the same as the first frame. I aliased this as TFrame2 and added another TLabel. Finally, I added another TFrame, again, slightly offset from the first two. Using the Attribute Editor again, I aliased this one as TFrame3 and set all the x,y position and the size. I altered the relief, to make it a bit easier to notice that this is a totally different frame.

Now, take a look at the Widget Tree window.

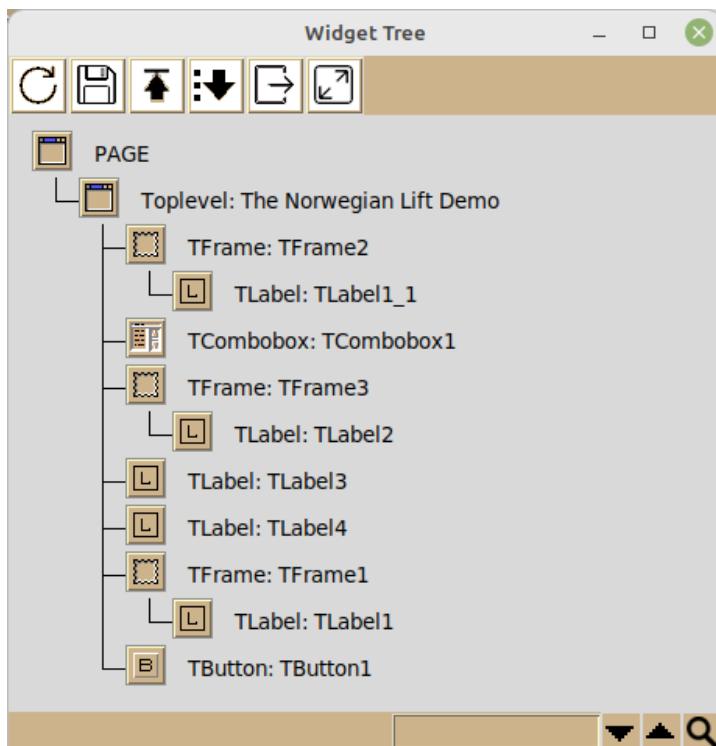


Figure 7: PAGE 8.x 'Norwegian Lift' Treeview view

If you want to work on, let's say TFrame1, simply click on the TFrame1 entry in the Widget Tree and it will pop to the top of the stack. Do whatever you need to (saving your design often) and click on the next TFrame entry you wish to modify.

Once you are done and you want to generate your Python code, try to remember which Frame you want the GUI to design first. Simply click that so it's on top of the stack, then generate your code.

Now for your code. In the support module, you will want to start with your startup function and make sure that your “start up” frame is on top. In order to switch frame visibility, I used a TCombobox.

```
def startup():
    global topframe
    _w1.TCombobox1.bind("<<ComboboxSelected>>", lambda e: on_Comboselect(e))
    _w1.TFrame1.lift(_w1.TFrame3)
    topframe = _w1.TFrame1
    _w1.TCombobox1.set("Frame 1")
```

Notice that I first set a global variable name “topframe” to hold what the current top frame is. We will reference this in the **ComboSelect** function. Then I set the binding for the ComboBox. Next I use the **frame.lift()** method to make sure that TFrame1 is visible. I set the global topframe variable to `_w1.TFrame1` and finally, force the TCombobox to display “Frame 1”.

Now for the `on_Comboselect()` function. This is where the magic happens.

```
def on_Comboselect(e):
    global topframe
    selected = _w1.combobox.get()
    print(f"Combobox Select: {selected}")
    if selected == "Frame 1":
        _w1.TFrame1.lift(topframe)
        topframe = _w1.TFrame1
    elif selected == "Frame 2":
        _w1.TFrame2.lift(topframe)
        topframe = _w1.TFrame2
    elif selected == "Frame 3":
        _w1.TFrame3.lift(topframe)
        topframe = _w1.TFrame3
```

- The actual syntax for the `widget.lift()` function is
`widget.lift(above_this=None)`

where `above_this` is the widget to be above on the stack list. Technically, you can just call this method without any parameters and it will work.

You can see that I simply use a simple **if** tree to decide what to do based on the value the user selected in the Combobox. Whichever selection the user makes, we then use the lift method of the desired frame and then set the global variable topframe to that frame alias.

That's it.

TNotebook Tab Positions

Assuming that you have used the TNotebook in your projects, you are probably familiar with the TNotebook Attribute Editor look in PAGE 7.6 and before.

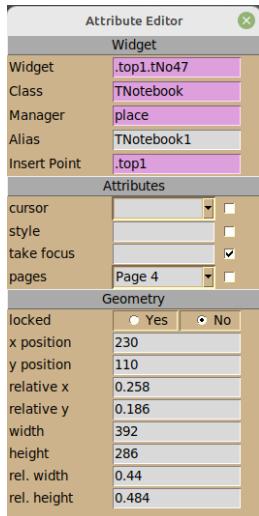


Figure 8: PAGE
7.6 TNotebook
Attribute Editor

In PAGE 8.x, that has changed, for the better.

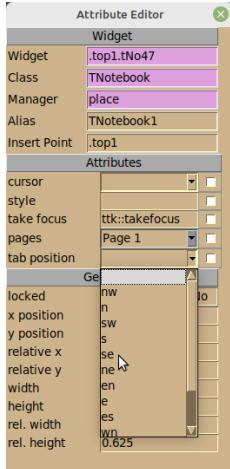


Figure 9: PAGE
8.x TNotebook
Attribute Editor

The new attribute entry is named “tab position” and is a dropdown selection. As you can tell from the image above, the dropdown has the following options nw, n, sw, s, se, ne, en, e, es, wn, w and ws, which are “compass” points corresponding to where the tabs start. This way, you don’t need to remember the actual style name.

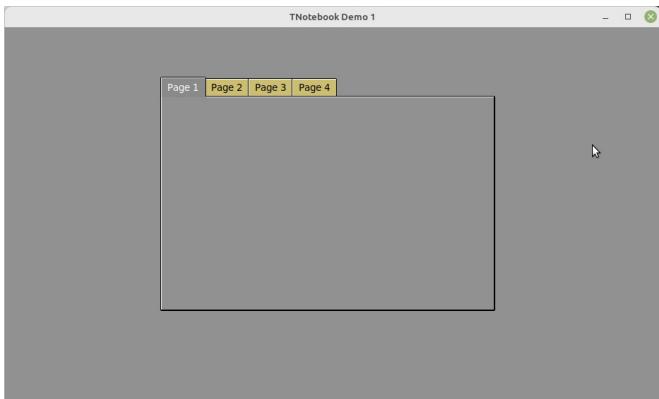


Figure 10: PAGE 8.x TNotebook Default Tab Position

The default is for the tabs to be set at the “nw” (north west) position. By selecting a different position from the dropdown, say “e” (or east), the tab position will change during your design session.

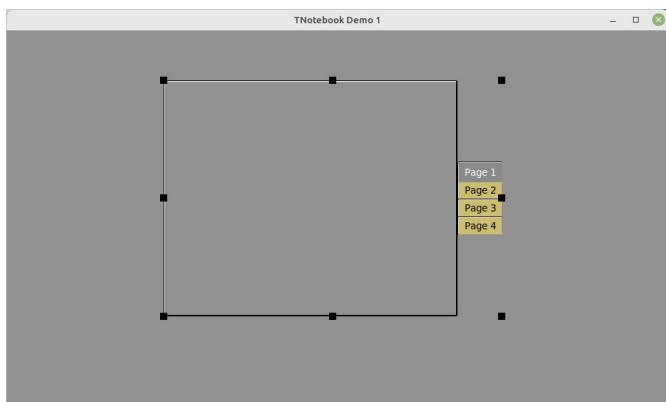


Figure 11: PAGE 8.x TNotebook Tab Position set to East

This allows you to take where the tabs are located when you are designing your application GUI.

This positioning is saved in your design GUI file for you by PAGE.

The one drawback to using the tab positioning in PAGE is that this only will work when you use one of the special PAGE themes. These are **notsodark**, **page-dark**, **page-light**, **page-notsodark** and **page-wheat**.

The reason for this, is when I created these themes, I included special code that takes advantage of the Tab Positioning. Themes like default, alt, classic and clam (and other OS dependent themes) don't include that code. Other third party themes, may or may not include this code, but probably won't.

Copy and Paste of menubars and popup menus

One of the things that many users wanted to see was the ability to “borrow” a menu bar or popup menu from an earlier version of their projects. Let’s face it. Menus take time to create properly and it’s a major pain recreating them every time you create a new project. Consistency in menus is as important to a good GUI as is as important as a good layout.

To copy a menubar or popup menu from an existing project, Select File | Borrow from the PAGE Main menu form. You will then be asked for the name of the project that has things that you want to add to your project. Once you have opened the project, the form(s) will be loaded and will have a purple colour. In addition, anything on that form will also appear in the Widget Tree with a purple background.

To copy a menubar from the “lender” project, click on the Menubar item in the Widget tree and use <Button-3> to bring up the Context menu. Select Copy. Then select the new Toplevel for your project, again use <Button-3> to paste the menu into the toplevel and press the <Enter> key on the keyboard.

For popup menus, use the same process.

•Note:

You can not copy both a menubar and a popup menu at the same time. You must use two separate processes.

Finally, when you have everything you wanted borrowed, you can click the second from the right button at the top of the Widget Tree to close the borrowed. Then save the changes to your project.

Changing the themes in your application

•NOTE

When I first created the function to allow you to change themes on the fly, I was only working on a Linux machine. I failed to test the code under Windows, so there were problems. The text here has been modified to provide a proper function that works under both Windows and Linux.

There might be times that you want to design your GUI with one theme, but offer another or others as options to your end user.

For example, let's say you want to offer a "default" theme which would be like something from PAGE 7.6, one that is "light" which would offer a white background (page-light theme) for everything and a "semi-dark" theme which would use the "notsodark" theme.

Thanks to PAGE 8.x, you can design your GUI using which ever theme you are comfortable with, then you can switch using the dropdown on the PAGE Main window to another theme to get an idea of what your GUI will look like. Once you have settled on the theme(s) you wish to offer, you will need to do a couple of things.

First, you will need to copy all the themes you want to use (even if it's just a single theme) into the theme folder in your project folder. Let's say you want to offer the three themes that I used in the example a few paragraphs ago. The "default" theme will be the OS theme "default". The light theme will be "page-light" and the semi-dark theme will be "notsodark". Your theme folder should look something like this.

Name	Size	Type	Date Modified
▶ notsodark	15 items	Folder	Thu 04 Jan 2024
▶ page-light	15 items	Folder	Thu 04 Jan 2024
notsodark.tcl	18.1 kB	Text	Tue 17 Oct 2023
page-light.tcl	18.0 kB	Text	Tue 17 Oct 2023

Figure 12: PAGE 8.xx Local Themes Folder

Notice that there are only two themes and two theme sub-folders for the three themes we need. Each of the two themes have a sub-folder that contains the graphics for the TRadiobuttons and the Tcheckbuttons and other widgets. The reason that we don't need to have a theme file for the "default" theme is that it is an operating system theme and is constant across Linux, Mac and Windows machines.

Imports

PAGE creates a basic import section for you, but for the themeswitch program, you need to add a few extra lines of code to support a few of the functions in the program. However, all the extra libraries are built into Python. I will highlight the lines that need to be change by using bold for the needed extra lines.

```
import sys
# =====
# Additional non-tkinter imports needed
# =====
import os.path
import glob

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

import themeswitch2
```

The last line of the import section imports the GUI.py file of your project (in this case themeswitch2.py) so the GUI definitions are all available. Don't change this line.

Program level global variables

These lines set some global variables (without the global keyword) to make them available to the entire program. PAGE creates the first line (`_debug=True`) for you. We need to add three more lines. Again, the lines you need to add, are in bold.

```
_debug = True # False to eliminate debug printing from callback functions.
location = themeswitch._location
programName = "Theme Switch Demo"
version = "0.1.0"
```

Having a variable for the location of the Python script is very important for applications that use images, graphics, databases and now themes. By setting this up here, you are creating a global variable without having to use the global keyword. The most important line in the above code is the one that sets the variable 'location'. We use this to be able to get the proper path of our project within the operating system.

```

# =====
# This is the main function that PAGE creates.
# =====
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 = themeswitch.Toplevel1(_top1)
    startup()
    root.mainloop()

```

The startup function

This is the first function that PAGE doesn't create a skeleton for us, so everything in the code presented here will be in bold.

The startup function for this program, is pretty short, only 8 lines including the function definition (but not the comments).

```

def startup():
    global currentTheme
    sty = ttk.Style()
    currentTheme = sty.theme_use()
    # Load the themes (OS level and theme folder themes)
    themelist = load_tcl_themes(location, True)
    # Bind the virtual <<ComboboxSelected>> event to the callback
    _w1.TCombobox1["values"] = themelist
    _w1.TCombobox1.bind("<<ComboboxSelected>>", Lambda e: on_ComboThemeSelect(e))
    # call the fix_all_widgets function
    fix_all_widgets()
    # Set the titlebar
    _top1.title(f"{programName} version {version}")

```

The first thing we do is define a global variable named `currentTheme`. Then we call the `load_tcl_themes` function which loads all of our themes into memory and returns a list of all the theme names in the variable '`themelist`'. Next we load the contents of the `themelist` variable into the Combobox then bind the virtual event `<<ComboboxSelected>>` to the callback function `on_ComboThemeSelect` so that when the user makes a selection in the Combobox, that selection will activate the proper theme. Then, we need to do some housekeeping and update some of the widget backgrounds to match the theme in the `fix_all_widgets` function. This is especially important since we are using some Tk widgets as well as ttk widgets and to make sure the Toplevel background gets the proper colour. Finally, we set the title bar text to our program name and the version of the program.

The `load_tcl_themes` function

I've created a function that will load all the themes that your system supports and any themes in your project theme folder. Remember, you must add the theme AND the theme sub folder (where needed) before this will be able to recognize the themes.

I'll comment on various parts of this (and other functions later on) to help you understand some of my cryptic coding style.

```
def load_tcl_themes(folder, silent):
    sty = ttk.Style()
    localThemes = sty.theme_names()
    currentTheme = sty.theme_use()
    if sys.platform == "win32":
        cthemename = f"\\"{currentTheme}.tcl"
    else:
        cthemename = f"/{currentTheme}.tcl"
    filedef = os.path.join(folder, "themes", "*.tcl")

    themelist1 = glob.glob(filedef)
    themelist = []
```

The first thing that happens in this function is to create an instance of the `ttk.Style` object that I call "sty". I then get the a list of the themes that are loaded by your operating system and the current theme in use (which PAGE loads based on the theme select at generate time) and save that theme name to the variable `currentTheme`. Since the theme name doesn't have an extension at this point, I need to add that as well as the leading slash so it can be compared later on and save that as `cThemeName`. (The variable `cThemeName` is setup dynamically, so this function can be used no matter which theme you design your program with.) Then we get a list of all the theme names that `ttk` knows about. At this point, it will only be the OS level themes. At this point the list `localThemes` contains (on my system) `['notsodark', 'clam', 'alt', 'default', 'classic']`. The theme name '`notsodark`' is included since that is the theme that I designed the program under, so PAGE starts up with that (from the GUI file).

The next things that get done is we create a string with the `os.path.join` method that includes the location of the program, the name of the themes folder (`themes`) and the wildcard `*.tcl`. By using the `os.path.join` method, we can ensure the slashes are correct for the Operating System. Then we use the `glob` python library to create a list of files matching the `filedef` string. Next, we create an empty list to hold all the theme names and finally add the items in `localThemes` into the newly created `themelist`.

```

if silent:
    print(f"{localThemes}")
    print(f"{themelist1}")
for theme in localThemes:
    themelist.append(theme)

```

Now themelist includes the names for both the OS themes and any themes in my theme folder ('notsodark', 'clam', 'alt', 'default', 'classic', 'page-light') and themelist1 contains the theme name with the fully qualified path..

At this point, we walk through themelist1 and compare the current item to the cThemeName string we set up before. If it matches, we step over it and don't do anything. This keeps the theme from being listed twice. If it doesn't match, then we call tcl to load the theme into memory, get just the name of the theme (without a path or extension) and add that to the themelist list. The partial code listing below, only deals with the Linux OS. I basically do the same thing for the Windows OS later on in the function.

```

for theme in themelist1:
    _top1.option_clear()
    if silent:
        print(f"{theme}")
    if theme.endswith(cthemename):
        pass
    elif theme.endswith("themes_list.tcl"):
        pass
    elif theme.endswith("page-legacy.tcl"):
        if silent:
            print(f"{theme}")
            root.tk.eval("set ::xframe #d9d9d9")
            root.tk.eval("set ::xfore #000000")
            root.tk.eval("set ::ana2color beige")
            root.tk.eval("set ::_tabfg1 black")
            root.tk.eval("set ::_tabfg2 black")
            root.tk.eval("set ::_tabbg1 #d9d9d9")
            root.tk.eval("set ::_tabbg2 #afb5cc")
            root.tk.eval("set ::_bgmode light")
        try:
            if sys.platform == "win32":
                rs = theme.rsplit("\\", 1)[1]
                nameend = rs.rfind(".tcl")
                themename = rs[:nameend]
            else:
                rs = theme.rsplit("/", 1)[1]
                nameend = rs.rfind(".tcl")
                themename = rs[:nameend]
            _top1.tk.call("source", theme)
            themelist.append(themename)
        except:
            pass

```

Now themelist includes the names for both the OS themes and any themes in my theme folder (['notsodark', 'clam', 'alt', 'default', 'classic', 'page-light']) and themelist1 contains the theme name with the fully qualified path..

At this point, we walk through themelist1 and compare the current item to the cThemeName string we set up before. If it matches, we step over it and don't do anything. This keeps the theme from being listed twice. If it doesn't match, then we call tcl to load the theme into memory, get just the name of the theme (without a path or extension) and add that to the themelist list. The partial code listing below, only deals with the Linux OS. I basically do the same thing for the Windows OS later on in the function.

```
else:  
    if sys.platform == "win32":  
        if silent:  
            print(f"{theme}")  
            rs = theme.rsplit("\\", 1)[1]  
            nameend = rs.rfind(".tcl")  
            themename = rs[:nameend]  
    else:  
        rs = theme.rsplit("/", 1)[1]  
        nameend = rs.rfind(".tcl")  
        themename = rs[:nameend]  
    if silent:  
        print(f"{themename}")  
    themelist.append(themename)  
    _top1.tk.call("source", theme)
```

Finally, the variable themelist, which is returned from the function, now (on my system) contains the following data...

```
themelist=['notsodark', 'clam', 'alt', 'default', 'classic', 'page-light']
```

```
themelist.sort()  
return themelist
```

The themelist variable is returned to the line that called our function.

Now all of our “local” themes are loaded into memory and can be applied to our program GUI at any time by using the `ttk style.theme_use({themename})` call.

The Callback for the Combobox

This is another function that we need to create from scratch. This is the callback that gets called whenever the user selects something from the Combobox.

It's fairly short.

The first part is the “normal” code that I use every time I use a Combobox. It basically gets the selection text from the Combobox and assign it to a variable named “selected”. If the `_debug` variable is set to True, I print the selected text to the terminal. I also set a special variable `toUse` to an empty string, preparing it for later use.

```
def on_CombоТhemeSelect(e):
    global currentTheme
    # toUse = ""
    selected = _w1.combobox.get()
    if _debug:
        print(f"Combobox Select: {selected}")
```

Finally, we normally would want to apply the style through the `ttk.theme_use()` method. We need to create an instance of the `ttk.Style` object first, so I use the variable `sty` to hold this. However, we have some Tk widgets mixed in to the GUI and I didn't put a `ttk.Frame` on the `Toplevel` to hold all of the widgets, so the `Toplevel` background won't get the theme background. So we call the `fix_all_widgets` function with the proper theme name as a parameter.

```
sty = ttk.Style()
# =====
# If you only use ttk widgets, you can uncomment the
# next line and skip the fix_all_widgets function.
# =====
sty.theme_use(selected)
fix_all_widgets()
```

The `fix_all_widgets` function

This function depends on another function named `all_children()` which we will look at in a moment, but we have to do some work first.

Since I didn't make the sty variable global, we need to redefine it first. Then we set the theme (which is passed in as a parameter) and use the style.lookup() method to get the background and foreground of the current theme. The first parameter of style.lookup is a “.” which means the theme global value for all widgets. If we needed to know a specific widget.class(), like TlabelFrame, we could use that instead. The second parameter is the attribute to get. In the two cases, it's background and foreground.

```
def fix_all_widgets():
    sty = ttk.Style()
    # =====
    # Apply the theme and get the background and foreground
    #   colours.
    # =====

    bg = sty.lookup(".", "background")
    fg = sty.lookup(".", "foreground")
```

Next, we need to create an empty list to pass to the all_children function. This is a recursive function, that gets all the child widgets for any given container widget. For example, a Frame is usually used as a container for other widgets, so usually will have many children widgets. In this case, since we want to get ALL children of the Toplevel, and all sub-children of any container widgets, we just need to pass the alias of our Toplevel form. Then the recursive function takes care of everything from there.

```
kids = get_Toplevel_Children()
```

As the recursive function all_children runs, it builds the list that we provide as kids. Once it's done, the list kids then contains all of the tkinter names of every widget and any children widgets it might have.

Notice that I specified that the list contains the tkinter names. It doesn't return your alias, but a name that Tkinter uses to directly reference the widget. For example, a Tk.Button returns <tkinter.Button object .!button> . So we can't really work with that, but we can reference the class of that name, in the case of the Tk.Button, the class would be 'Button' which we can use. If the widget were to be a ttk::Button, the class would be 'TButton' .

So we walk through the widget list kids and obtain the class of the widget by using the winfo_class method. Then we check to see if the widget is a ttk widget. Since these are usually taken care of by the theme, we can easily ignore them, with the exception of the TLabel widget. SOMETIMES, it doesn't get the theme properly, so we want to make sure it uses the proper background/foreground colours based on the theme.

```
for k in kids:
    print(f"{k} - {k.winfo_class()}")
    # print(f"{k.winfo_class()}")
    clas = k.winfo_class()
    if str(clas).startswith("T"):
        if str(clas) == "TLabel":
            k.configure(background=bg)
            k.configure(foreground=fg)
    else:
        pass
```

If the class is NOT a ttk class, then we need to apply the background and attempt to apply the foreground colours. I say attempt, because some widgets, like the Tk::Frame, has no text, therefore no foreground colour to set. If we try, we'll get an error.

```
else:
    k.configure(background=bg)

    try:
        k.configure(foreground=fg)
    except:
        pass
```

Finally we apply the background colour to the Toplevel widget. IF you use a TFrame as the main container for your project then you don't need to do this. Again, I didn't do that in this project, so we need to make sure the background colour is properly set. (Toplevel doesn't have a foreground attribute, so don't try to set it.)

```
_w1.TNotebook1_t1.configure(background=bg)
_w1.TNotebook1_t2.configure(background=bg)
_w1.TNotebook1_t3.configure(background=bg)
_w1.TNotebook1_t4.configure(background=bg)
```

The last thing we do is call update for the Toplevel, just to make sure that everything gets updated.

```
_top1.configure(background=bg)
_top1.update()
```

The get_Toplevel_Children function

The last thing we need to cover for this project is the recursive all_children function. I found the code for this at <https://stackoverflow.com/questions/49026353/get-all-children-of-tkinter-python-app>. The function works very well.

Remember to pass in the Toplevel alias (in this case _top1) and the list variable to use.

```
def get_TopLevel_Children():
    kids = _top1.winfo_children()
    # print(kids)
    return kids
```

That's the code. I've included the project as part of the documentation in the code folder as themeswitch2.

Hacking Themes

I bounced the title of this section off of one of my friends on the Discord PAGE forum who goes by the alias of Marvin. He suggested either “Theme Hacking” or “Hacking Themes”. I responded with “*‘Theme Hacks’ sounds somewhat passive, while ‘Hacking Themes’ is aggressive. When I think of ‘Hacking Themes’ I think of Indiana Jones hacking and slashing through the jungle wearing a white cowboy hat to get to the treasure at the end of the path.*”

But the whole thing came about by him asking about ttk::Frames and relief settings under different Themes. Unlike the Tk widgets, ttk widgets are mainly controlled by the theme that is being used (which is usually set at startup). So what can you do if you like the colour set of a theme but don’t like the relief or borderwidth of a particular ttk widget, say a ttk::LabelFrame.

Generally speaking, there are three ways to change a theme to suit your particular needs. First, is to learn Tcl well enough to modify (read **hack**) the existing theme. Second is to contact the author (if you can) and ask for a change the theme or to create a special version of the theme for you. Both of these options, though, can be problematical, since the OS themes are hard to find and the theme authors really aren’t available much any more. In addition, if you want to hack a system theme, you also will need to hack the theme of your users. That would probably not happen. Also remember, programs (which include Themes) are like a programmer’s child. Would you like it if someone said that your child looks nice, but would be better with green hair and skin? Maybe, but for most of us, our children are fine (for the most part) the way they are, without modification.

The third way is to learn to configure the widgets ourselves in our code. That way, it doesn’t matter who is running the program and on what OS Platform.

I’m going to give you a relatively quick way to change most any attribute of most any ttk widget under most any theme. I say relatively, because it will take a little bit of research in order to do this.

Let’s take for example a ttk Labelframe. I picked this, since many of us use them often and we are mostly familiar with them. Modifying a **Tk Labelframe** is easy, you just use the **widget.configure()** method, setting the attributes like **relief** or **bordercolor** to whatever you want. While there are some attributes that you can change on a **ttk Labelframe** through the **configure** method, that won’t work for most attributes that you would want to change like **background** colour or **bordercolor** or **relief**.

A website that will (mostly) help you is <https://www.tcl-lang.org/man/tcl8.6/TkCmd/contents.htm>. Under that is a link for the **ttk::labelframe**. From that page, I borrowed the information on Styling Options. (You **MUST** use the styling options for **most** changes to any **ttk** widget.)

TLabelframe styling options configurable with ttk::style are:

- background color
- bordercolor color
- borderwidth amount
- darkcolor color
- labelmargins amount
- labeloutside boolean

```
-lightcolor color  
-relief relief
```

In order to change any of these options, you need to use the configure option a little bit differently. Say that you want to set the background of the **ALL** of the TLabelframe widgets to “pink2”, the bordercolor to “blue” and the borderwidth to 4. Here’s how you would do it.

```
sty.configure(  
    "TLabelframe", background="pink2", bordercolor="blue", borderwidth=4  
)
```

Please notice that I said **ALL** of the TLabelframe widgets. That’s because when you configure things like this, we are using the “generic” TLabelframe widget. In order to do just one TLabelframe widget, not all of them, you must define a style, then apply that style to the TLabelframe widget in question.

```
sty.configure(  
    "My.TLabelframe", background="pink2", bordercolor="blue", borderwidth=4  
)  
_w1.TLabelframe1.configure(style="My.TLabelframe")
```

Here’s what the recipe looks like on the TLabelframe.

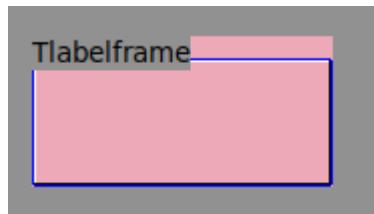


Figure 13: TLabelframe
'hacked'

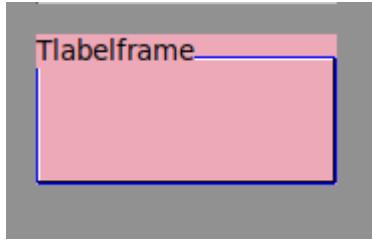
Well, it worked somewhat, but why is the label portion not changed? That’s because there is one more part to the style, I didn’t tell you about. It is on the web though.

TLabelframe.Label styling options configurable with ttk::style are:
-background color
-font font
-foreground color

So we have to add another bit of code.

```
sty.configure("My.TLabelframe.Label", background="pink2")
```

This needs to go after the first `sty.configure` line but before the `My.TLabelframe` gets the style.



*Figure 14: TLabelframe
fully 'hacked'*

By using this “recipe”, you can change one, many or all widgets of pretty much any type of widget.

For the most part, you can find the attributes for just about any ttk widget from the web page.

PAGE 8.x Theme Gallery

Here you will find a screenshot of each of the “out of the box” themes that PAGE 8.x supports, using most of the ttk widgets that PAGE supports. I chose to use the Theme Chooser feature to display each of the different themes for the screenshots.

Many of the differences in the themes might be hard to notice at first glance, but they are there.

Since I’m running Linux, I only have access to the four System themes that Linux supports. The extra themes that Mac OS X supports aren’t shown at this time.

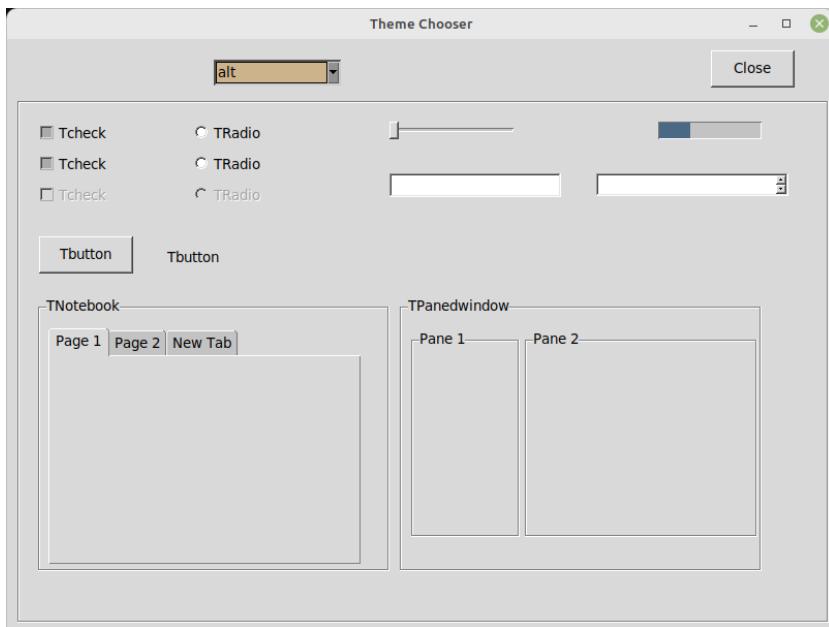


Figure 15: Alt theme

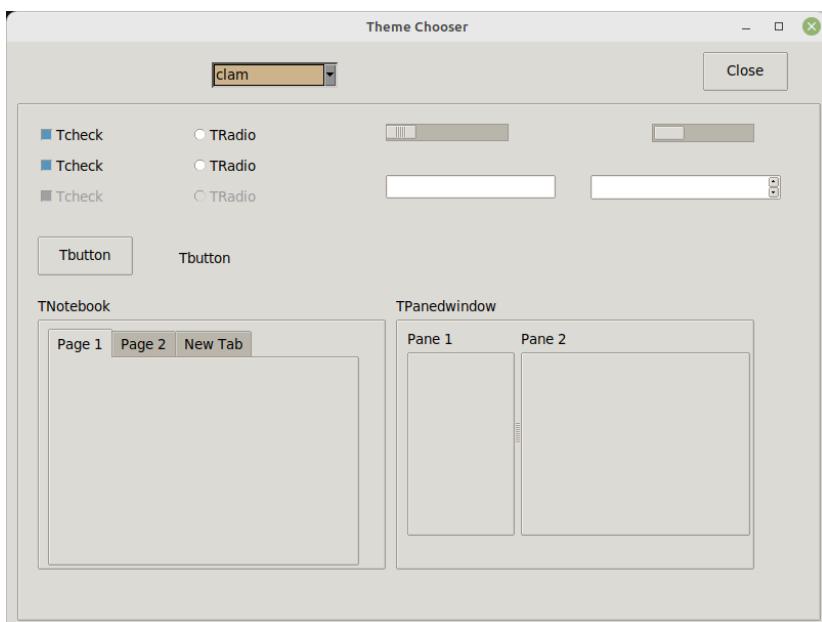


Figure 16: Clam theme

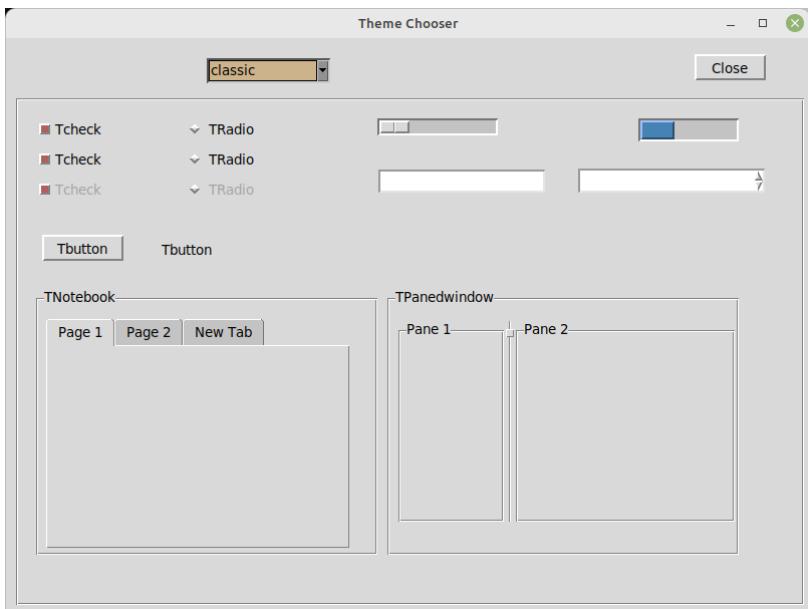


Figure 17: Classic theme

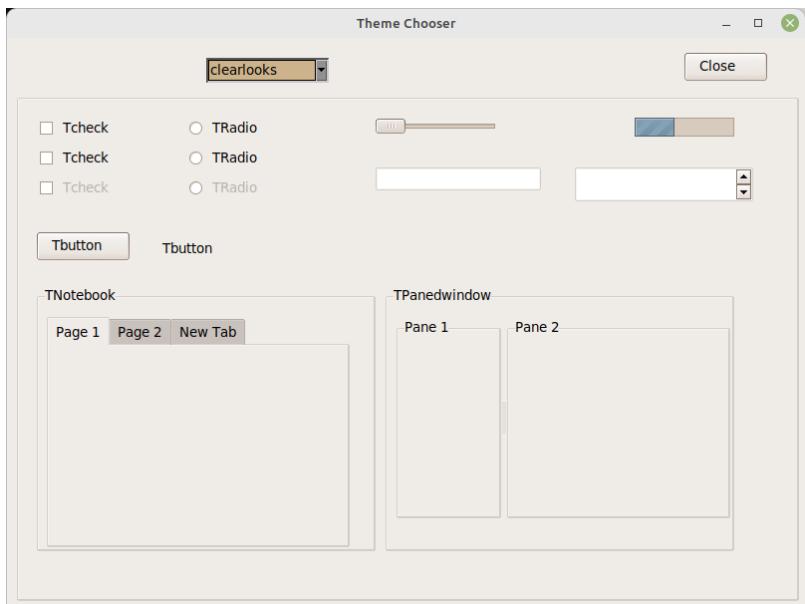
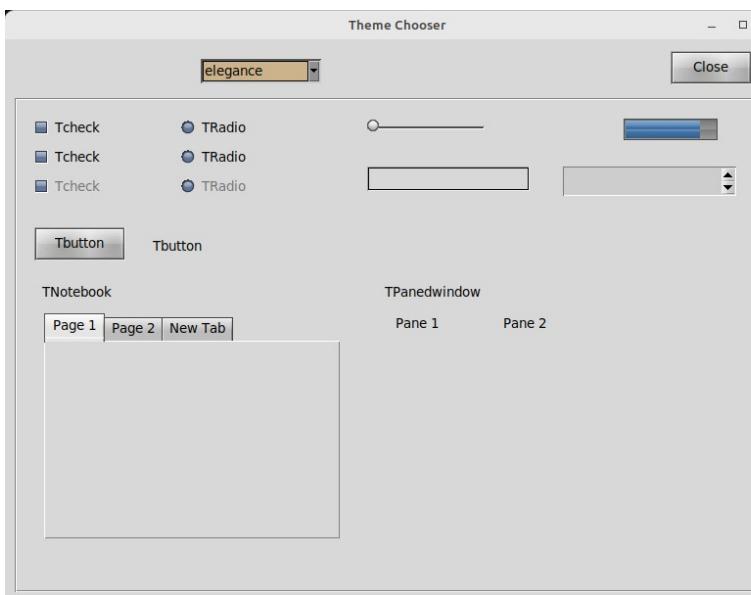
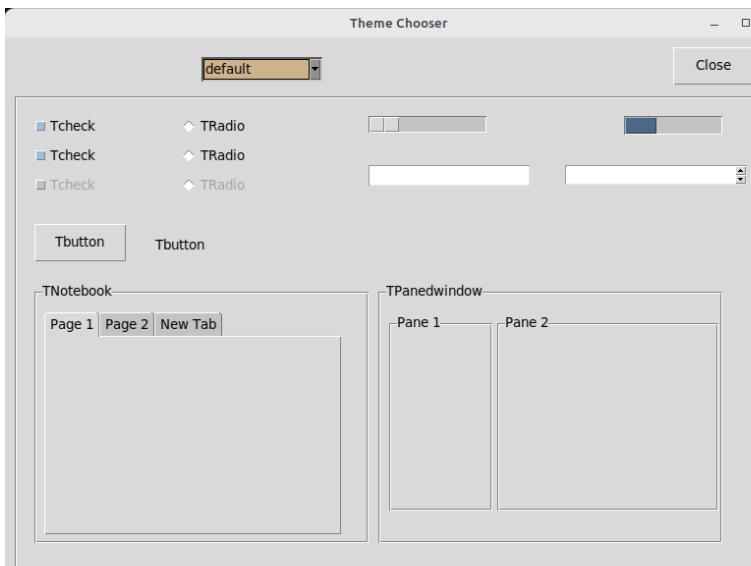


Figure 18: Clearlooks theme



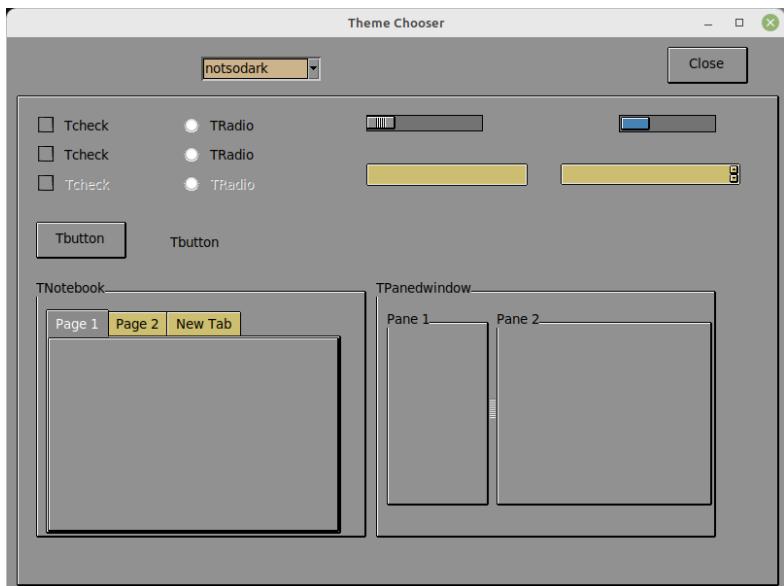


Figure 21: notsodark theme

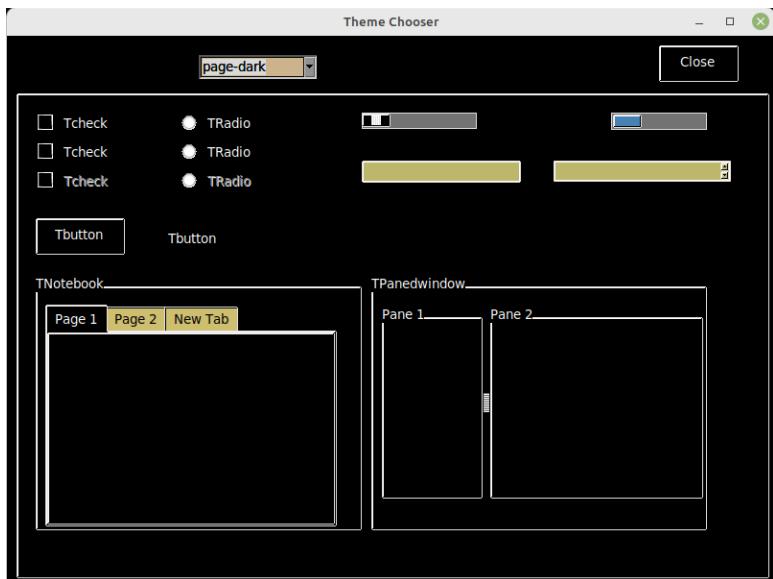


Figure 22: page-dark theme

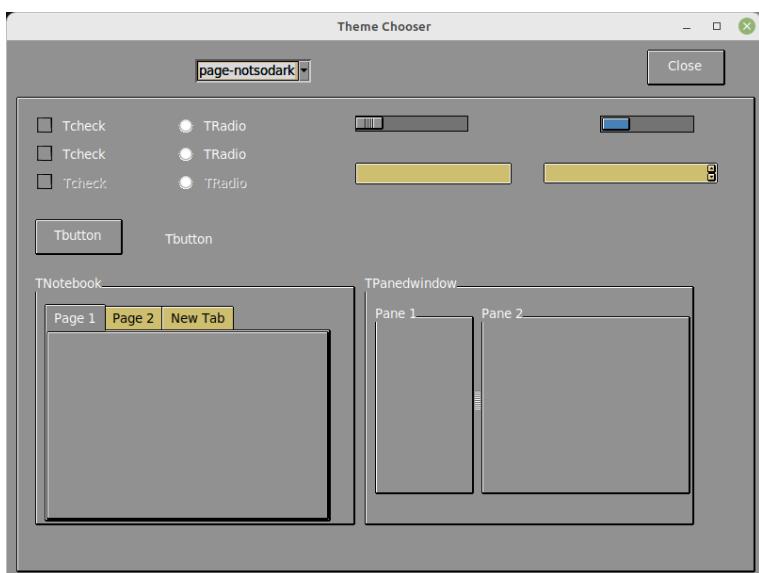
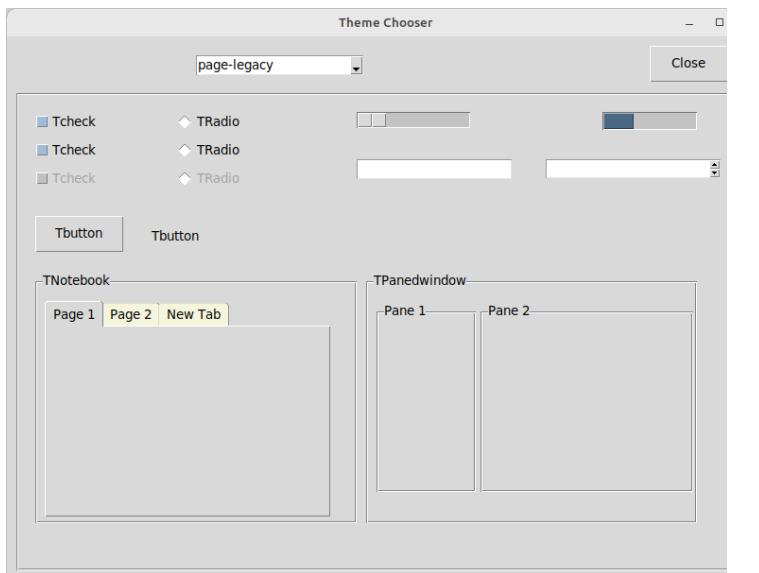


Figure 25: page-notsodark theme

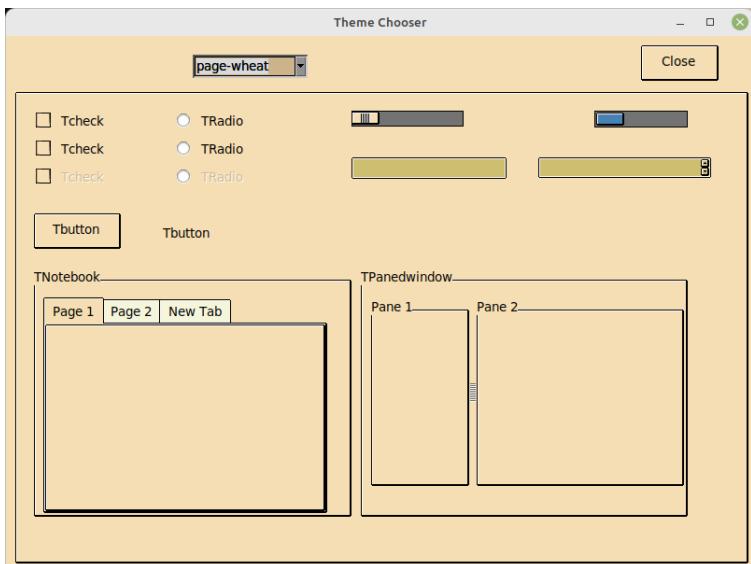


Figure 26: page-wheat theme

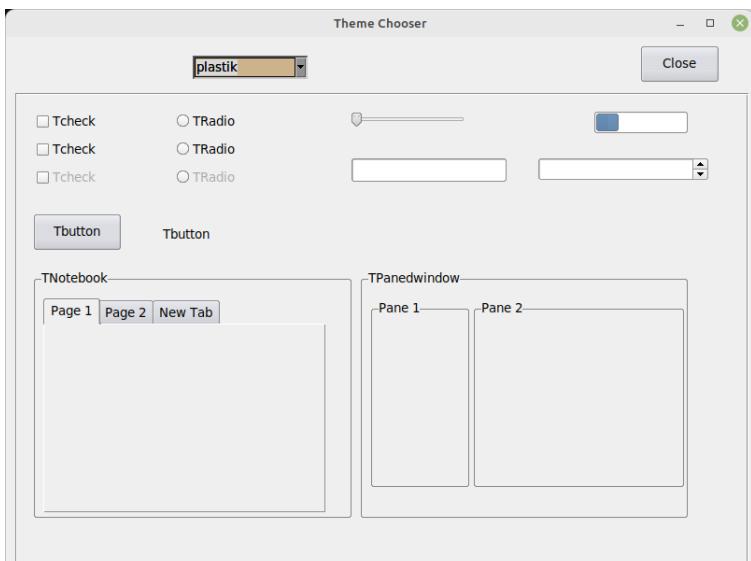


Figure 27: Plastik theme

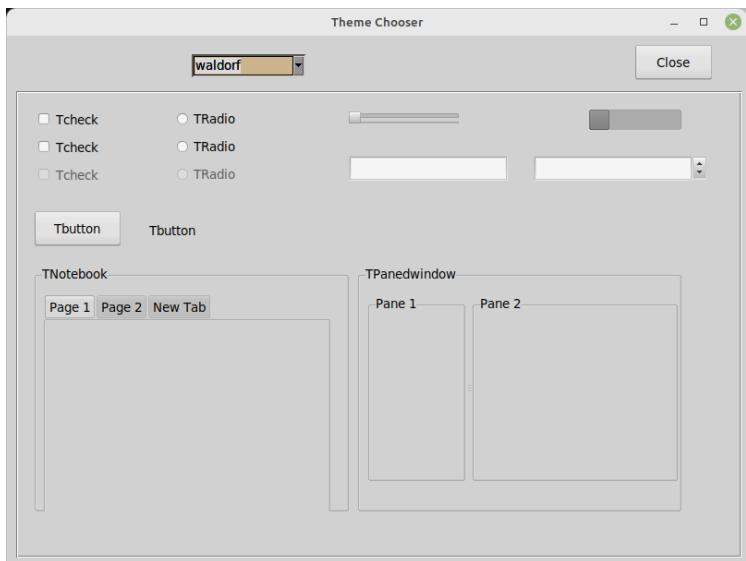


Figure 28: *Waldorf theme*

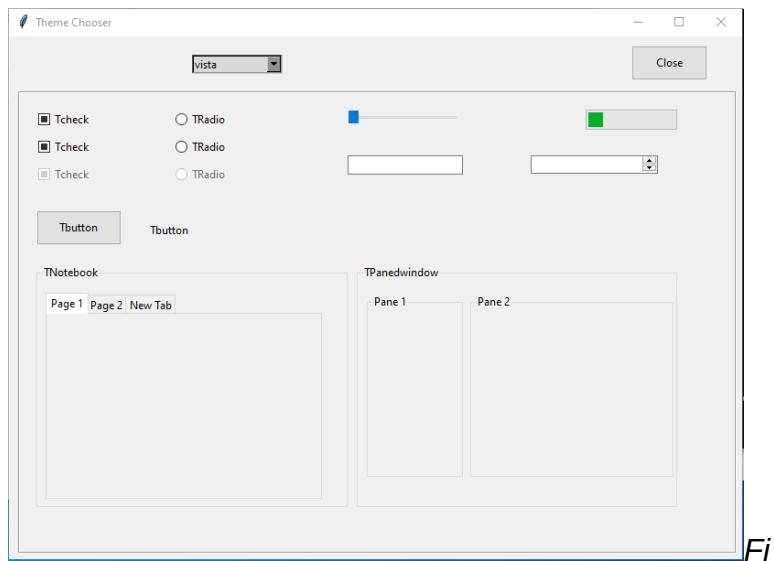


Figure 29: Vista theme

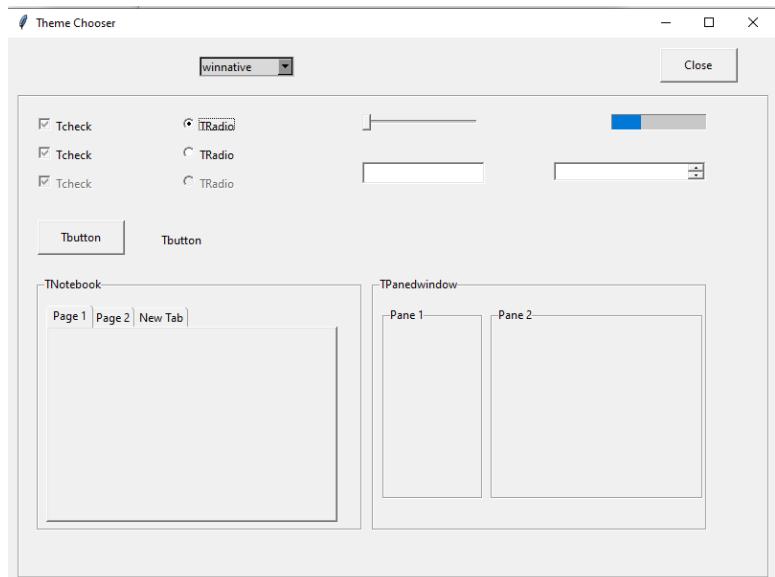


Figure 30: Winnative theme

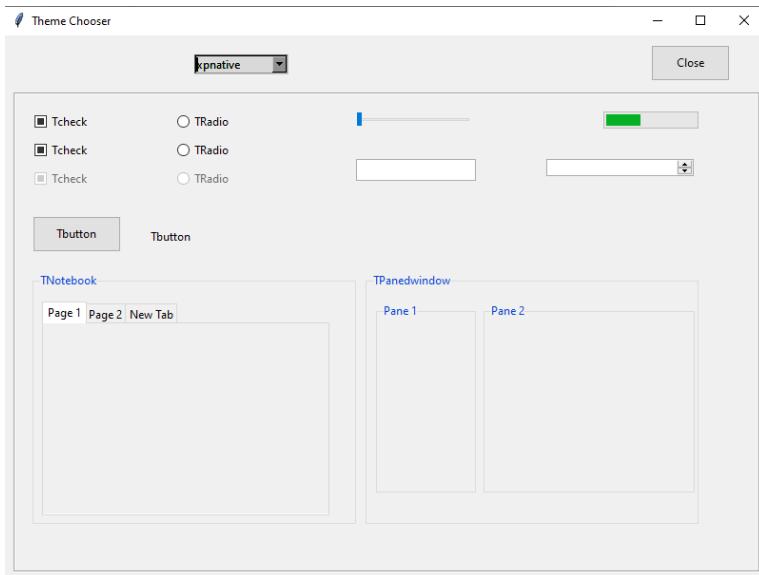


Figure 31: Xpnative theme

Adding third-party themes to PAGE 8.x

While the theme set that comes with PAGE 8.x will probably be enough for most users of PAGE, there are some of us that will continue to strive to be ‘cutting edge’ and these themes might not fit these users.

All of the themes that PAGE 8.x supports in the designer are located in the distribution folder under the “themes” folder.

If you want to push the envelope and try some other third-party themes, <https://wiki.tcl-lang.org/page/List+of+ttk+Themes> is a very good starting point. While it might be a little bit out of date when you get to it, many of the themes can be used, with caution.

⚠ PLEASE be aware, neither Don nor I are responsible for any support for any additional themes you add to PAGE or your programs. There is no way that we can provide user support for any themes that were not included in the distribution of PAGE.

You will need to edit the themes_list.tcl file, but don’t worry, you don’t have to understand Tcl to do this. You can use just about any text editor like notepad, gedit or whatever is on your system.

First, here is what the themes_list.tcl file looks like...

```
set theme_dir $vTcl(VTCL_HOME)
set themes {
    clearlooks
```

```
notsodark
page-dark
page-legacy
page-light
page-notsodark
page-wheat
waldorf
}
foreach theme $themes {
    set filename [file join $theme_dir themes $theme.tcl]
    source $filename
}
```

The part that you will need to edit is the part that says set themes. In that there is a single theme on each line. This list of themes will be the ones that will show up in the dropdown in PAGE 8.x. Just add the theme name that you want to add, on a line by itself between the { }. Try to make sure the indentation is the same as the other lines.

Getting Help for PAGE 8.x

We still have the Discussion forum at <https://.net/p/page/discussion/> . You can always post your questions, comments or feature requests there. Please create an account there and login, so we can see your username rather than the Anonymous default username.

You can also use our Discord PAGE users forum. If you don't have a Discord account, it's easy to create one. There is a Discord native app for most every operating system, tablet and phone. Here is a link that will NEVER expire.

<https://discord.gg/F7MQ3nQM9M>

My homepage is also available at <https://thedesignedgeek.xyz/> . You can leave an email at the bottom of the page under Contact.

Finally, you can always email Don directly. His email can be found in the documentation for PAGE.

Table of Figures

Figure 1: PAGE 8.x Theme Selector.....	7
Figure 2: PAGE 8.x Preferences menu.....	8
Figure 3: PAGE 8.x Theme Chooser Feature.....	11
Figure 4: PAGE 8.x Theme Chooser Demo Form.....	12
Figure 5: Halvard's Bible Viewer Program (in progress).....	13
Figure 6: PAGE 8.x 'Norwegian Lift' in Action.....	14
Figure 7: PAGE 8.x 'Norwegian Lift' Treeview view.....	14
Figure 8: PAGE 7.6 TNotebook Attribute Editor.....	16
Figure 9: PAGE 8.x TNotebook Attribute Editor.....	16
Figure 10: PAGE 8.x TNotebook Default Tab Position.....	17
Figure 11: PAGE 8.x TNotebook Tab Position set to East.....	17
Figure 12: PAGE 8.xx Local Themes Folder.....	20
Figure 13: TLabelframe 'hacked'.....	31
Figure 14: TLabelframe fully 'hacked'.....	32
Figure 15: Alt theme.....	34
Figure 16: Clam theme.....	34
Figure 17: Classic theme.....	35
Figure 18: Clearlooks theme.....	36
Figure 19: Default theme.....	37
Figure 20: Elegance theme.....	37
Figure 21: Notsodark theme.....	38
Figure 22: page-dark theme.....	38
Figure 23: page-light theme.....	39
Figure 24: page-legacy theme.....	40
Figure 25: page-notsodark theme.....	40
Figure 26: page-wheat theme.....	41
Figure 27: Plastik theme.....	41
Figure 28: Waldorf theme.....	42
Figure 29: Vista theme.....	43
Figure 30: Winnative theme.....	44
Figure 31: Xpnative theme.....	45

