

tkinter_utils

**MANUAL
VERSIÓN 1.0**

Indice

Presentación breve del sistema basado en herencias de clases	páginas 3 – 5
CLASE MADRE <code>gui_trkinter_widgets</code>	páginas 6 – 12
CLASE HIJA <code>frame_con_scrollbar</code>	páginas 13 – 14
CLASE HIJA <code>entry_propio</code>	páginas 15 – 17
CLASE HIJA <code>treeview_propio</code>	páginas 18 – 20
CLASE HIJA <code>scrolledtext_propio</code>	páginas 21 – 23
ANEXOS EJEMPLOS DE USO	páginas 24 – 85

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

Este sistema (en adelante **mi_sistema_tkinter**) es un sistema de herencias de clases de la librería **tkinter** de Python que permite facilitar la configuración de GUI's (interfaces de usuario) mediante el uso de kwargs.

No pretende ser una refactorización completa de la librería tkinter. Simplemente, **un sistema híbrido entre objetos y métodos nativos de tkinter y otros personalizados.**

El manual que se presenta continuación corresponde a la **versión 1.0** en el cual se explica brevemente la arquitectura del sistema y posteriormente en los anexos se muestra como configurar widgets en GUI's en base a ejemplos documentados.

Se irán **incorporando en el futuro mas widgets y funcionalidades** siempre con el fin de simplificar en pocas líneas de código la configuración de GUI's de forma limpia, dinámica y escalable.

Una mejora prevista **de cara el futuro es generar un log de errores de configuración de los kwargs.** En la versión actual todos los posibles errores de configuración se omiten. Esto implica que el usuario tenga que ser muy meticuloso a la hora de configurar sus GUI's.

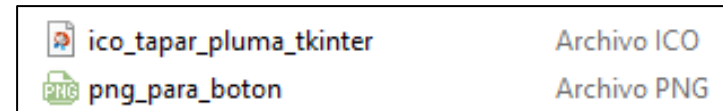
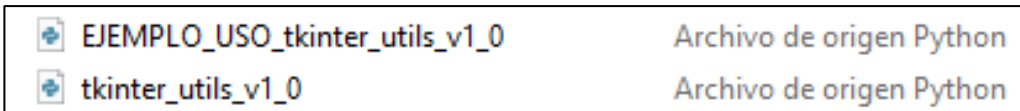
No hay fechas previstas para futuras versiones, será cuando necesite incorporar otros widgets en otros proyectos personales que también iré publicando a lo largo del tiempo en Github.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

mi_sistema_tkinter se compone de un archivo .py (llamado **tkinter_utils_v1_0** en el repositorio GitHub). Se puede renombrar como se quiera.

Para poder usar **mi_sistema_tkinter** en otro proyecto (en adelante **mi_proyecto**) hay que ponerlo en la ubicación de **mi_proyecto**.

En este manual **mi_proyecto** es el archivo .py llamado **EJEMPLOS_USO _tkinter_utils_v1_0**, también en el repositorio GitHub, y se puede renombrar como se quiera. Se adjuntan también un fichero .ico y otro .png para usar en los ejemplos detallados en los anexos.



mi_proyecto además de las librerías Python necesarias para su uso requiere para el uso de **mi_sistema_tkinter** de tener las librerías Python siguientes instaladas:

Librerías	Nativas Python	Versión instalación
diffliib	Si	
inspect	Si	
itertools	Si	
os	Si	
pkgutil	Si	
re	Si	
sys	Si	
tkinter	Si	
typing	Si	
Pillow	No	11.0.0
pandas (*)	No	1.5.0
tkcalendar	No	1.6.1

(*) la versión de pandas aquí es antigua pero se puede usar una versión posterior

Comandos ms-dos para instalar sea en el pc o dentro de un entorno virtual (venv) en una ubicación dentro del pc:

Si eres admin de tu pc ➡ **pip install Pillow==11.0.0 tkcalendar==1.6.1 pandas==1.5.0**

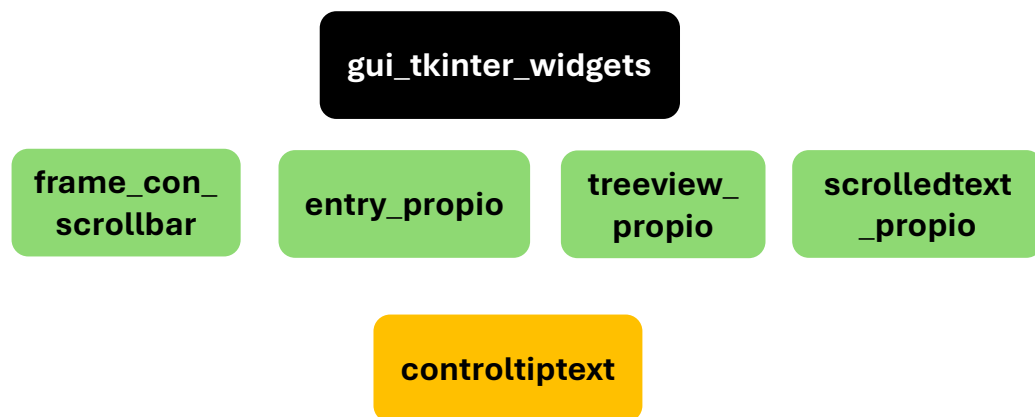
Si NO eres admin de tu pc ➡ **pip install Pillow==11.0.0 tkcalendar==1.6.1 pandas==1.5.0 --user**

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

Hay que importar en **mi_proyecto** el archivo Python de **mi_sistema_tkinter** (el alias que se la da es de libre elección por el usuario).

```
import UTILS as mod_utils
```

El módulo python **mi_sistema_tkinter** se compone de las clases siguientes:



HERENCIA	CLASE	DESCRIPCIÓN
madre	gui_tkinter_widgets	Permite crear widgets nativos tkinter aplicandole atributos / metodos nativos pero también propios
hijas	frame_con_scrollbar	Permite crear frames con scrolling del ratón y con scrollbars vertical y/o horizontal. Usa atributos / metodos nativos del objeto "Frame" de tkinter y también con atributos / metodos propios a esta clase e Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .
	entry_propio	Permite crear entries (textbox) con atributos / metodos nativos del objeto "Treeview" de tkinter y también con atributos / metodos propios a esta clase. Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .
	treeview_propio	Permite crear treeview s (subformularios) con atributos / metodos nativos del objeto "Treeview" de tkinter y también con atributos / metodos propios a esta clase. Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .
	scrolledtext_propio	Permite crear scrolledtext (textbox con scroll vertical) con atributos / metodos nativos del objeto "Treeview" de tkinter y también con atributos / metodos propios a esta clase. Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .
independientes	controlliptext	Clase independiente que permite generar un mensaje cuando el usuario coloca el cursor del ratón sobre un botón.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1 gui tkinter widgets

La clase **gui_tkinter_widgets** toma en su constructor hasta 3 parámetros (además del master):

PARAMETROS	OBLIGARIEDAD	TIPO DATO
tipo_widget_param	Todos los widgets	String
self_clase_gui_donde_call_rutina	Según el tipo de widget	Ver más adelante
kwargs_config_parametros	Todos los widgets	Diccionario

```
class gui_tkinter_widgets():  
    def __init__(self, master, tipo_widget_param = None, self_clase_gui_donde_call_rutina = None, **kwargs_config_parametros):
```

Para el parámetro **tipo_widget_param** hay que informar el nombre del widget nativo en tkinter. **mi_sistema_tkinter** tiene un sistema interno en el que se eliminan todos los espacios en blanco y se prueban todas las combinaciones posibles de letras (minúsculas y mayúsculas) hasta toparse con el string esperado para poder crear el widget.

Ejemplo para un **Label** donde el usuario ha escrito “**L AB eL**”. **sistema_tkinter** elimina todos los espacios en blanco y prueba cada combinación de caracteres minúsculas / mayúsculas hasta toparse con **Label** que es un atributo de tkinter.

label	incorrecto	Label	correcto
label	incorrecto	Label	incorrecto
labEl	incorrecto	LabEl	incorrecto
labEL	incorrecto	LabEL	incorrecto
laBel	incorrecto	LaBel	incorrecto
laBeL	incorrecto	LaBeL	incorrecto
laBEI	incorrecto	LaBEI	incorrecto
laBEL	incorrecto	LaBEL	incorrecto
lAbel	incorrecto	LAbel	incorrecto
lAbEl	incorrecto	LAbEl	incorrecto
lAbEL	incorrecto	LAbEL	incorrecto
lABel	incorrecto	LABel	incorrecto
lABEl	incorrecto	LABEl	incorrecto
lABEL	incorrecto	LABEL	incorrecto
LAbel	incorrecto	LABel	incorrecto
LABEl	incorrecto	LABEL	incorrecto
LABEL	incorrecto	LABEL	incorrecto

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1 gui tkinter widgets

Para el parámetro **kwargs_config_parametros**, a continuación, se listan los atributos propios y se especifica como combinarlos con atributos nativos tkinter (en el **Anexo** del presente documento se explican uno por uno como se han de configurar con ejemplos).

1 / 2

CLASE	HERENCIA	DESCRIPCIÓN	ATRIBUTOS PROPIOS - NIVEL 1			ATRIBUTOS PROPIOS - NIVEL 2	
			Nombre	Tipo dato	Aplica para que widgets	Nombre	Tipo dato
gui_tkinter_widgets	madre	Permite crear widgets nativos tkinter aplicandole atributos / metodos nativos pero también propios	dicc_config_root	diccionario	root	tupla_geometry	tupla de 2 elementos numéricos positivos (width, height)
			dicc_colocacion	diccionario	Todos los widgets nativos tkinter	metodo	string (solo acepta el valor "place"). La versión presentada aquí de sistema_tkinter es muy reducida con respecto a la que estubo en desarrollo hace unos meses.
						coord_x	Coordenada horizontal (número positivo o nulo)
						coord_y	Coordenada vertical (número positivo o nulo)
			combobox_lista_opciones	lista, tupla o set (para lista o tupla sus items deben ser string o números)	combobox	na	na
			alineacion	string	Todos los widgets nativos tkinter que acepten los atributos nativos anchor y/o justify (valores posibles: center, left, right, top_center, top_left, top_right, bottom_center, bottom_left y bottom_right)	na	na
			dicc_imagen	diccionario	Todos los widgets nativos tkinter que acepten el metodo nativo image	png_imagen	ruta de acceso a un fichero .png
			controltiptext	String con el mensaje que el usuario desee	Todos los widgets nativos tkinter que acepten los metodos nativos; bind , after , wininfo_rootx , wininfo_rooty , wininfo_height y after_cancel (botones por ejemplo)	tupla_imagen_resize	tupla de 2 elementos numéricos positivos
						na	na
			lista_dicc_rutina_trace_variable_enlace	Lista de diccionarios donde cada diccionario contiene las keys: rutina (obligatorio), tipo_trace (obligatorio), parametros_args (opcional) y parametros_kwargs (opcional)	Todos los widgets nativos que acepten stringvar (variables de enlace)	rutina	String con el nombre de la rutina o función
						tipo_trace	Cualquier valor de trace que se aplica en el metodo nativo trace_add (write, read etc)
						parametros_args	tupla
						parametros_kwargs	diccionario

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1 gui tkinter widgets

2 / 2

CLASE	HERENCIA	DESCRIPCIÓN	ATRIBUTOS PROPIOS - NIVEL 1			ATRIBUTOS PROPIOS - NIVEL 2	
			Nombre	Tipo dato	Aplica para que widgets	Nombre	Tipo dato
gui_tkinter_widgets	madre	Permite crear widgets nativos tkinter aplicandole atributos / metodos nativos pero también propios	listbox_lista_items	lista o tupla (donde cada item es string o número)	listbox	na	na
			listbox_lista_items_seleccionados	lista de valores seleccionados por el usuario (no es configurable, es interno a la clase). Informar la key con el valor que se desee (lo importante es que la key este en los kwargs no su valor, recomendación: True)	listbox	na	na
			listbox_seleccionar_todo_o_nada	True o False	listbox	na	na
			bloquear_interaccion_nueva_ventana_con_otras	True o False	Todos los widgets nativos tkinter que acepten el metodo nativo grab_set	na	na
			mantener_nueva_ventana_encima_otras	True o False	Todos los widgets nativos tkinter que acepten el metodo nativo transient	na	na
			bloquear	True o False	Todos los widgets nativos tkinter que acepten el atributo nativo state	na	na
			destroy	True o False	Todos los widgets nativos tkinter que acepten el metodo nativo destroy	na	na
			dicc_rutina	diccionario	Botones o cualquier widget que acepte el metodo nativo tkinter bind (rutinas de evento)	rutina	String con el nombre de la rutina o función
						parametros_args	tupla
						parametros_kwargs	diccionario
			lista_dicc_aplicar_eventos_widget	Lista de diccionarios donde cada diccionario contiene las keys: tipo_bind (obligatorio), rutina (obligatorio), parametros_args (opcional) y parametros_kwargs (opcional)	Cualquier widget que acepte el metodo nativo tkinter bind (rutinas de evento)	tipo_bind	Acepta cualquier evento nativo: <<ComboboxSelected>>, <<TreeviewSelect>>, <ButtonRelease-1> etc
						rutina	String con el nombre de la rutina o función
						parametros_args	tupla
						parametros_kwargs	diccionario

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1

gui tkinter widgets

Los nombres de los atributos propios listados en los cuadros anteriores se definen una sola vez en el constructor de la clase **gui_tkinter_widgets** por lo que se pueden cambiar aquí y ponerles los nombres con los que los usuarios se sientan más cómodos.

```
103     #se asigna el nombre de los atributos propios
104     #(esto es por si se desea cambiar en el futuro el nombre el mismo sin tener que modificar el código de la clase)
105     self.nombre_kwarg_dicc_config_root = "dicc_config_root"
106     self.nombre_kwarg_dicc_config_root_tupla_geometry = "tupla_geometry"
107
108     self.nombre_kwarg_dicc_colocacion = "dicc_colocacion"
109     self.nombre_kwarg_dicc_colocacion_metodo = "metodo"
110     self.nombre_kwarg_dicc_colocacion_coord_x = "coord_x"
111     self.nombre_kwarg_dicc_colocacion_coord_y = "coord_y"
112
113     self.nombre_kwarg_combobox_lista_opciones = "combobox_lista_opciones"
114
115     self.nombre_kwarg_dicc_rutina = "dicc_rutina"
116     self.nombre_kwarg_dicc_rutina_nombre_rutina = "rutina"
117     self.nombre_kwarg_dicc_rutina_parametros_args = "parametros_args"
118     self.nombre_kwarg_dicc_rutina_parametros_kwarg = "parametros_kwarg"
119
120     self.nombre_kwarg_lista_dicc_rutina_aplicar_eventos_widget = "lista_dicc_rutina_aplicar_eventos_widget"
121     self.nombre_kwarg_lista_dicc_rutina_aplicar_eventos_widget_tipo_bind = "tipo_bind"
122     self.nombre_kwarg_lista_dicc_rutina_aplicar_eventos_widget_rutina = "rutina"
123     self.nombre_kwarg_lista_dicc_rutina_aplicar_eventos_widget_rutina_parametros_args = "parametros_args"
124     self.nombre_kwarg_lista_dicc_rutina_aplicar_eventos_widget_rutina_parametros_kwarg = "parametros_kwarg"
125
126
127     self.nombre_kwarg_dicc_imagen_boton = "dicc_imagen_boton"
128     self.nombre_kwarg_dicc_imagen_boton_png_imagen = "png_imagen"
129     self.nombre_kwarg_dicc_imagen_boton_tupla_imagen_resize = "tupla_imagen_resize"
130
131
132     self.nombre_kwarg_controltip = "controltip"
133
134     self.nombre_kwarg_lista_dicc_rutina_trace_variable_enlace = "lista_dicc_rutina_trace_variable_enlace"
135     self.nombre_kwarg_lista_dicc_rutina_trace_variable_enlace_tipo_trace = "tipo_trace"
136     self.nombre_kwarg_lista_dicc_rutina_trace_variable_enlace_nombre_rutina = "rutina"
137     self.nombre_kwarg_lista_dicc_rutina_trace_variable_enlace_parametros_args = "parametros_args"
138     self.nombre_kwarg_lista_dicc_rutina_trace_variable_enlace_parametros_kwarg = "parametros_kwarg"
139
140     self.nombre_kwarg_alineacion = "alineacion"
141     self.nombre_kwarg_bloquear = "bloquear"
142
143     self.nombre_kwarg_listbox_lista_items = "listbox_lista_items"
144     self.nombre_kwarg_listbox_lista_items_seleccionados = "listbox_lista_items_seleccionados"
145     self.nombre_kwarg_listbox_listbox_seleccionar_todo_o_nada = "listbox_seleccionar_todo_o_nada"
146
147     self.nombre_kwarg_destroy = "destroy"
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1

gui tkinter widgets

Los atributos propios (**nivel 1**) comentados se pueden combinar con atributos nativos.

```
{
  "text": "PROCESOS"
  , "font": ("Calibri", 13, "bold")
  , "width": 13
  , "bd": 1
  , "relief": "solid"
  , "bg": "#1F40AD"
  , "fg": "white"
  , "colocacion_dicc": {"metodo": "place", "coord_x": 0, "coord_y": 0}
}
```

*Kwargs para objeto tipo **label***

```
{
  "font": ("Calibri", 10)
  , "width": 25
  , "colocacion_dicc": {"metodo": "place", "coord_x": 100, "coord_y": 40}
  , "justify": tk.LEFT
  , "combobox_lista_opciones": [mod_gen.dicc_procesos[item] for item in mod_gen.dicc_procesos.keys()]
  , "dicc_rutina": {
    "rutina": "def_GUI_combobox_proceso" #la rutina tiene que estar definida en la clase gui_ventana
    , "tipo_rutina": "event"
    , "tipo_bind": "<<ComboboxSelected>>"
  }
}
```

*Kwargs para objeto tipo **combobox***

Los atributos propios (**nivel 2**) comentados se pueden combinar con atributos nativos tan solo en el caso de **dicc_config_root**.

```
"dicc_config_root":
{
  "title": mod_gen.nombre_app
  , "iconbitmap": mod_gen.ico_app
  , "tupla_geometry": (750, 570)
  , "resizable": (0, 0)
}
```

En ambos casos, si se configura un atributo nativo y uno propio que hace lo mismo (independientemente de que el propio salga en los kwargs antes que el nativo o no) **premia siempre el atributo propio**.

```
  , "alineacion": "right"
  , "anchor": "w"
```



premia el atributo propio **alineacion** a pesar de salir antes en los kwargs

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1

gui tkinter widgets

Todos los widgets que se crean con **gui_tkinter_widgets** tienen un stringvar (variable de enlace) asignado mediante el atributo **variable_enlace**.

```
kwargs_config_label = {"text": "prueba_label",
                        , "width": 15
                        , "bd": 2
                        , "relief": "solid"
                        , "colocacion_dicc": {"metodo": "place", "coord_x": 10, "coord_y": 0}
                        }

label = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "label", **kwargs_config_label)

label.variable_enlace.set("prueba")
stringvar_label = label.variable_enlace.get()

print(label.variable_enlace)
print(stringvar_label)
```

```
PS C:\Users\user> & C:/Users/user/AppData/Local/Programs/Python/Python39/python.exe
PY_VAR2
prueba
```

Los widgets nativos tkinter que no aceptan variables de enlace devuelven **None** al atributo **variable_enlace**.

Por defecto, la clase genera un **stringvar**, los casos donde se necesita un intvar no estan contemplados.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1 gui tkinter widgets

La clase **gui_tkinter_widgets** dispone de un método propio **config_atributos** que se usa internamente a la clase nada más crear el widget y que se puede reutilizar posteriormente en **mi_proyecto** una vez el widget creado.

Como único parámetro se le pasa los kwargs de configuración de atributos (nativos y/o propios) comentados (pueden ser otros a los que se usaron para crear el widget).

```
kwargs_config_root_1 = {"dicc_config_root":
                        {"title": "PRUEBA ROOT"
                        , "tupla_geometry": (100, 100)
                        , "resizable": (0, 0)
                        }
                        }

kwargs_config_root_2 = {"dicc_config_root":
                        {"title": "PRUEBA ROOT"
                        , "tupla_geometry": (300, 300)
                        , "resizable": (0, 0)
                        }
                        }

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root_1)
root.config_atributos(**kwargs_config_root_2)

root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

2 frame_con_scrollbar

Permite crear un frame con scrolling del ratón y con scrollbars vertical y/o horizontal..

La clase **frame_con_scrollbar** toma en su constructor el parámetro **kwargs_config_widget** (además del master):

```
class frame_con_scrollbar(gui_tkinter_widgets):
    #clase hija de la clase gui_tkinter_widgets para crear frames con scrolling
    def __init__(self, master, **kwargs_config_widget):
```

Se listan para **kwargs_config_widget** sus atributos propios (en el **Anexo** del presente documento se explican uno por uno como se han de configurar con ejemplos):

CLASE	HERENCIA	DESCRIPCIÓN	ATRIBUTOS PROPIOS - NIVEL 1		ATRIBUTOS PROPIOS - NIVEL 2		IMPLEMENTACIÓN SCROLLBAR
			Nombre	Tipo dato	Nombre	Tipo dato	
frame_con_scrollbar	hija	Permite crear frames con scrolling del ratón y con scrollbars vertical y/o horizontal. Usa atributos / metodos nativos del objeto "Frame" de tkinter y también con atributos / metodos propios a esta clase e Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets.	dicc_frame_scrollbar	diccionario	width_visible	Ancho del frame visible en pantalla (número positivo). Obligatorio	HORIZONTAL width_visible < width_total
					width_total	Ancho total del frame en pantalla, parte visible + parte invisible que se muestra con el scrolling (número positivo). Se puede omitir de configurarlo si no se desea scrollbar horizontal	
					height_visible	Altura del frame visible en pantalla (número positivo). Obligatorio	VERTICAL height_visible < height_total
					height_total	Altura total del frame en pantalla, parte visible + parte invisible que se muestra con el scrolling (número positivo). Se puede omitir de configurarlo si no se desea scrollbar vertical	
					tupla_coord_place	Tupla de 2 items numéricos con las coordenadas ("x" y "y") para colocar el frame. Obligatorio	
					velocidad_scrolling	Número positivo (mayor número mayor velocidad de scrolling). Si se omite su configuración se establece por defecto a 1 (muy lento)	

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

2

frame con scrollbar

Los nombres de los atributos propios listados en el cuadro anterior se definen una sola vez en el constructor de la clase **entry_propio** por lo que se pueden cambiar aquí y ponerles los nombres con los que los usuarios se sientan más cómodos.

```
#se asigna el nombre del diccionario kwargs donde recuperar los parametros de configuracion
#(esto es por si se desea cambiar en el futuro el nombre el mismo sin tener que modificar el codigo
self.nombre_kwargs_dicc_frame_scrollbar = "dicc_frame_scrollbar"
self.nombre_kwargs_dicc_frame_scrollbar_tipo_scrollbar = "tipo_scrollbar"
self.nombre_kwargs_dicc_frame_scrollbar_width_visible = "width_visible"
self.nombre_kwargs_dicc_frame_scrollbar_width_total = "width_total"
self.nombre_kwargs_dicc_frame_scrollbar_height_visible = "height_visible"
self.nombre_kwargs_dicc_frame_scrollbar_height_total = "height_total"
self.nombre_kwargs_dicc_frame_scrollbar_tupla_coord_place = "tupla_coord_place"
self.nombre_kwargs_dicc_frame_scrollbar_velocidad_scrolling = "velocidad_scrolling"

self.nombre_kwargs_dicc_frame_scrollbar_scrollbar_vertical_y_horizontal = "vertical_y_horizontal"
self.nombre_kwargs_dicc_frame_scrollbar_scrollbar_horizontal_y_vertical = "horizontal_y_vertical"
self.nombre_kwargs_dicc_frame_scrollbar_scrollbar_horizontal = "horizontal"
self.nombre_kwargs_dicc_frame_scrollbar_scrollbar_vertical = "vertical"
```

La clase dispone de un mecanismo interno que recalcula de forma automática el tipo de scrolling a implementar según como este configurado el kwargs incluido en el atributo propio **dicc_frame_scrollbar**.

La clase dispone de un método propio **modificaciones** que permite alterar el tipo de scrolling posteriormente a la creación del frame (matizar por motivos que se documentan dentro del código de la clase que, **si se localizan cambios en las configuraciones de los scrolling**, al aplicar este método propio el frame creado anteriormente se elimina y se vuelve a crear).

La clase **frame_con_scrollbar** al ser clase hija de **gui_tkinter_widgets** hereda el método propio **config_atributos** por lo que se puede configurar atributos nativos tkinter y propios definidos en la clase madre siempre y cuando sean compatibles con el objeto frame de tkinter (el de tk no el de ttk).

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3

entry_propio

Permite crear un entry (textbox) con reglas de validación e impedir que el usuario salga del mismo hasta que se cumplan los requisitos esperados.

La clase **entry_propio** toma en su constructor el parámetro **kwargs_config_widget** (además del master):

```
class entry_propio(gui_tkinter_widgets):  
    #clase hija de la clase gui_tkinter_widgets para crear  
  
    def __init__(self, master, **kwargs_config_widget):
```

Se listan para **kwargs_config_widget** sus atributos propios (en el **Anexo** del presente documento se explican uno por uno como se han de configurar con ejemplos):

CLASE	HERENCIA	DESCRIPCIÓN	ATRIBUTOS PROPIOS - NIVEL 1			ATRIBUTOS PROPIOS - NIVEL 2	
			Nombre	Tipo dato	Aplica para que widgets	Nombre	Tipo dato
entry_propio	hija	Permite crear entries (textbox) con atributos / metodos nativos del objeto "Treeview" de tkinter y también con atributos / metodos propios a esta clase. Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .	dicc_entry	diccionario	entry	formato_validacion	string y ha de estar definido en el atributo interno dicc_patrones_validacion de la clase (key 1): entero_positivo, entero_negativo, float_positivo, float_negativo, texto, fecha_ddmmaaaa, fecha_yyyymmdd o alfanumerico
						texto_longitud_maxima	número entero positivo o nulo
						titulo_messagebox_warning	String con el mensaje que el usuario desee
						calendario_tupla_coord_place_y_width	Tupla de 3 valores numéricos positivos (coordena x, coordenada y, width)
						calendario_iconbitmap	ruta fichero .ico para tapar la pluma tkinter en el calendario para fechas

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3

entry_propio

Los nombres de los atributos propios listados en el cuadro anterior se definen una sola vez en el constructor de la clase **entry_propio** por lo que se pueden cambiar aquí y ponerles los nombres con los que los usuarios se sientan más cómodos.

```
#se asigna el nombre del diccionario kwargs donde recuperar los parametros de configuracion
#(esto es por si se desea cambiar en el futuro el nombre el mismo sin tener que modificar el codigo de la clase)
self.nombre_kwargs_dicc_entry = "dicc_entry"
self.nombre_kwargs_dicc_entry_formato_validacion = "formato_validacion"
self.nombre_kwargs_dicc_entry_texto_longitud_maxima = "texto_longitud_maxima"
self.nombre_kwargs_dicc_entry_titulo_messagebox_warning = "titulo_messagebox_warning"
self.nombre_kwargs_dicc_entry_calendario_tupla_coord_place_y_width = "calendario_tupla_coord_place_y_width"
self.nombre_kwargs_dicc_entry_calendario_iconbitmap = "calendario_iconbitmap"
```

La clase **entry_propio** al ser clase hija de **gui_tkinter_widgets** hereda el método propio **config_atributos** por lo que se puede configurar atributos nativos tkinter y propios definidos en la clase madre siempre y cuando sean compatibles con el objeto entry de tkinter.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3 entry propio

La clase **entry_propio** dispone de un atributo propio **dicc_patrones_validacion** (diccionario) en el cual se definen las reglas de validación del entry creado y sirve de base para rutinas internas para fijar el comportamiento del widget cuando el usuario intenta salir del mismo.

```
self.dicc_patrones_validacion = {
    "entero_positivo": {
        "validacion_re": r"^\d+$"
        , "mensaje_warning": "Solo se admiten enteros positivos."
    }
    , "entero_negativo": {
        "validacion_re": r"^\d+$"
        , "mensaje_warning": "Solo se admiten enteros negativos."
    }
    , "float_positivo": {
        "validacion_re": r"^\d*\.\d+$"
        , "mensaje_warning": "Solo se admiten enteros o decimales positivos."
    }
    , "float_negativo": {
        "validacion_re": r"^\d*\.\d+$"
        , "mensaje_warning": "Solo se admiten enteros o decimales negativos."
    }
    , "texto": {
        "validacion_re": None
        , "mensaje_warning": "Solo se admite texto REPLACE_ME."
    }
    , "fecha_ddmmaaaa": {
        "validacion_re": r"^\d{2}[-./]\d{2}[-./]\d{4}$"
        , "mensaje_warning": "Solo se admiten fechas en formato EUR (dd/mm/aaaa, dd-mm-aaaa o dd.mm.aaaa)."
    }
    , "fecha_yyyymmdd": {
        "validacion_re": r"^\d{4}[-./]\d{2}[-./]\d{2}$"
        , "mensaje_warning": "Solo se admiten fechas en formato USA (aaaa/mm/dd, aaaa-mm-dd o aaaa.mm.dd)."
    }
    , "alfanumerico": {
        "validacion_re": r"^[w]+$"
        , "mensaje_warning": "Solo se admiten caracteres alfanumericos."
    }
}
```

Al configurar en los kwargs el atributo propio **texto_longitud_máxima**, se reemplaza el REPLACE_ME por "(longitud máxima: xx caracteres)" (donde xx es el valor del atributo propio) para que salga en el warning en caso de no cumplir el entry el requisito configurado

„

Las keys de este diccionario son los valores que se han de informar en los kwargs comentados en la página anterior (**formato_validacion**). Son configurables dentro de la clase siempre y cuando el diccionario conserve la misma estructura.

Las reglas de validación se definen con la librería Python **re**.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 treeview propio

La clase **treeview_propio** toma en su constructor hasta 2 parámetros (además del master):

PARAMETROS	OBLIGARIEDAD	TIPO DATO
<code>self_clase_gui_donde_call_rutina</code>	Según el tipo de widget	Ver más adelante
<code>kwargs_config_parametros</code>	Todos los widgets	Diccionario

```
class treeview_propio(gui_tkinter_widgets):
    #clase hija de la clase gui_tkinter_widgets para crear widget de tipo treeview con metodos propios
    def __init__(self, master, self_clase_gui_donde_call_rutina = None, **kwargs_config_widget):
```

Para el parámetro **kwargs_config_widget**, a continuación, se listan los atributos propios (en el **Anexo** del presente documento se explican uno por uno como se han de configurar con ejemplos).

CLASE	HERENCIA	DESCRIPCIÓN	ATRIBUTOS PROPIOS - NIVEL 1		ATRIBUTOS PROPIOS - NIVEL 2	
			Nombre	Tipo dato	Nombre	Tipo dato
treeview_propio	hija	Permite crear treeview s (subformularios) con atributos / metodos nativos del objeto "Treeview" de tkinter y también con atributos / metodos propios a esta clase. Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .	seleccion_item	string (ninguno, simple o multiple)	na	na
			dicc_treeview	diccionario	height	número positivo (aunque sea un atributo nativo aquí es obligatorio)
					columnas_df	lista donde cada uno de sus items son string o números (la longitud de la lista ha de ser la misma que las de columnas_treeview y width_columnas_treeview)
					columnas_treeview	lista donde cada uno de sus items son string o números (la longitud de la lista ha de ser la misma que las de columnas_df y width_columnas_treeview)
					width_columnas_treeview	lista donde cada uno de sus items son números positivos (la longitud de la lista ha de ser la misma que las de columnas_df y columnas_treeview). Permite calcular sumando todos los items el ancho del treeview
			dicc_rutina_click_item	diccionario	rutina	string
					parametros_args	tupla
					parametros_kwargs	diccionario

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4

treeview propio

Los nombres de los atributos propios listados en el cuadro anterior se definen una sola vez en el constructor de la clase **treeview_propio** por lo que se pueden cambiar aquí y ponerles los nombres con los que los usuarios se sientan más cómodos.

```
#se asigna el nombre del diccionario kwargs donde recuperar los parametros de configuracion
#(esto es por si se desea cambiar en el futuro el nombre el mismo sin tener que modificar el codigo de la clase)
self.nombre_kwargs_dicc_treeview = "dicc_treeview"
self.nombre_kwargs_dicc_treeview_seleccion_item = "seleccion_item"
self.nombre_kwargs_dicc_treeview_seleccion_item_ninguno = "ninguno"
self.nombre_kwargs_dicc_treeview_seleccion_item_simple = "simple"
self.nombre_kwargs_dicc_treeview_seleccion_item_multiple = "multiple"
self.nombre_kwargs_dicc_treeview_height = "height"
self.nombre_kwargs_dicc_treeview_columnas_df = "columnas_df"
self.nombre_kwargs_dicc_treeview_columnas_treeview = "columnas_treeview"
self.nombre_kwargs_dicc_treeview_width_columnas_treeview = "width_columnas_treeview"

self.nombre_kwargs_dicc_treeview_rutina_click_item = "dicc_rutina_click_item"
self.nombre_kwargs_dicc_treeview_rutina_click_item_nombre_rutina = "rutina"
self.nombre_kwargs_dicc_treeview_rutina_click_item_parametros_args = "parametros_args"
self.nombre_kwargs_dicc_treeview_rutina_click_item_parametros_kwargs = "parametros_kwargs"
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4

treeview propio

Los atributos propios (**nivel 1**) comentados **NO** se pueden combinar con atributos nativos (salvo el **height** que ha de estar incluido en **dicc_treeview**).

```
{
  "dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 10},
  "dicc_treeview": {
    "seleccion_item": "simple",
    "height": 4,
    "columnas_df": ["COLUMNA_1", "COLUMNA_2", "COLUMNA_3"],
    "columnas_treeview": ["col A", "col B", "col C"],
    "width_columnas_treeview": [50, 50, 50]
  },
  "dicc_rutina_click_item": {"rutina": "def_rutina_click_item"}
}
```

La clase **treeview_propio** al ser clase hija de **gui_tkinter_widgets** hereda el método propio **config_atributos** por lo que se puede configurar atributos propios definidos en la clase madre siempre y cuando sean compatibles con el objeto treeview de tkinter.

Al crearse el treeview se le asigna el atributo **datos_item_seleccionado** que permite recuperar varios datos al hacer click sobre cada item del treeview. Es un **diccionario**:

KEY	VALOR
lista_columnas_df	lista de las columnas del dataframe que ha servido de base para rellenar el treeview y configuradas en el atributo propio dicc_treeview (lista_columnas_df)
lista_columnas_treeview	lista de las columnas del dataframe que ha servido de base para rellenar el treeview y configuradas en el atributo propio dicc_treeview (columnas_treeview)
lista_width_columnas_treeview	lista de las columnas del dataframe que ha servido de base para rellenar el treeview y configuradas en el atributo propio dicc_treeview (width_columnas_treeview)
lista_tipo_dato_columna_df	lista con los tipos de dato (dtype) de las columnas del dataframe que ha servido de base para rellenar el treeview y configuradas en el atributo propio dicc_treeview (lista_columnas_df)
lista_datos_item_seleccionado	lista de los datos del item seleccionado (es lista de listas)

Para poder acceder a los datos del atributo **datos_item_seleccionado** hay que configurar en los kwargs el atributo propio **dicc_rutina_click_item** y declararla en **mi_proyecto**.

La clase **treeview_propio** trata internamente la rutina configurada como una rutina de evento por lo que cuando se declara en **mi_proyecto** no es necesario pasarle en sus parámetros el **event**.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

5

scrolledtext_propio

Permite crear un scrolledtext desde un string o un dataframe. En ambos casos, la clase tiene un mecanismo interno que permite agregar al scrolledtext un scrollbar vertical si el numero de líneas informadas en el mismo supera el atributo nativo **height** configurado en los kwargs (en caso de informarse desde un string la inserción depende también del atributo nativo **wrap** configurado, si este no se configura no se aplica ningún wrap).

En el caso de que el **scrolledtext creado se haga desde un dataframe** existe también la posibilidad de aplicar tags en cada línea:

- Marcar **todas las líneas de un color** según que se cumplan los criterios configurados en la misma (u otra) columna que se usa para informar el scrolledtext
- Marcar **fragmentos de texto dentro de una misma línea de un color comparando el texto con otra columna** del dataframe.

La clase **scrolledtext_propio** toma en su constructor el parámetro **kwargs_config_widget** (además del master):

```
class scrolledtext_propio(gui_tkinter_widgets):  
    #clase hija de la clase gui_tkinter_widgets para crear  
    # con metodos propios asociados  
  
    def __init__(self, master, **kwargs_config_widget):
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

5

scrolledtext propio

Se listan para **kwargs_config_widget** sus atributos propios (en el **Anexo** del presente documento se explican uno por uno como se han de configurar con ejemplos):

CLASE	HERENCIA	DESCRIPCIÓN	ATRIBUTOS PROPIOS - NIVEL 1		ATRIBUTOS PROPIOS - NIVEL 2	
			Nombre	Tipo dato	Nombre	Tipo dato
scrolledtext_propio	hija	Permite crear scrolledtext (textbox con scroll vertical) con atributos / metodos nativos del objeto "Treeview" de tkinter y también con atributos / metodos propios a esta clase. Hereda los atributos / metodos nativos y propios de la clase madre gui_tkinter_widgets .	colocacion_scrollbar_horizontal	diccionario	metodo	string (solo acepta el valor "place")
			df_datos	dataframe	coord_x	Coordenada horizontal (número positivo o nulo)
			columna_df_para_informar	string	coord_y	Coordenada vertical (número positivo o nulo)
			lista_dicc_tag_linea_completa	Lista de diccionarios donde cada diccionario contiene las keys de la columna siguiente	na	na
					na	na
					nombre_tag	string o número
			lista_dicc_tag_caracteres_cambiantes_comparativa	Lista de diccionarios donde cada diccionario contiene las keys de la columna siguiente	columna_df_tag_aplicar	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
					case_sensitive	True o False
					dicc_config	diccionario de atributos nativos
					nombre_tag	string o número
					columna_df_filtro_registros_aplicar_tag	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
					columna_df_filtro_registros_aplicar_tag_valor	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
					columna_df_comparar_1	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
					columna_df_comparar_2	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
					case_sensitive	True o False
					marcar_toda_linea_si_todo_varia	True o False
					dicc_config	diccionario de atributos nativos

Una vez creado el **scrolledtext**, la clase **scrolledtext_propio** dispone de un método propio **modificaciones** para poder configurar su contenido y/o tags (en el **Anexo** del presente documento se explican uno por uno como se han de configurar con ejemplos):

METODO PROPIO	OPCIONES
modificaciones	borrar_contenido_y_tags
	agregar_solo_contenido_desde_string
	agregar_solo_contenido_desde_dataframe
	agregar_contenido_y_tags_desde_dataframe

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

5

scrolledtext propio

Los nombres de los atributos propios listados en el cuadro anterior se definen una sola vez en el constructor de la clase **scrolledtext_propio** por lo que se pueden cambiar aquí y ponerles los nombres con los que los usuarios se sientan más cómodos.

```
#se asigna el nombre del diccionario kwargs donde recuperar los parametros de configuracion
#(esto es por si se desea cambiar en el futuro el nombre el mismo sin tener que modificar el codigo de la clase)
self.nombre_kwargs_scrolledtext_propio_colocacion_scrollbar_horizontal = "colocacion_scrollbar_horizontal"
self.nombre_kwargs_scrolledtext_propio_colocacion_scrollbar_horizontal_metodo = "metodo"
self.nombre_kwargs_scrolledtext_propio_colocacion_scrollbar_horizontal_coord_x = "coord_x"
self.nombre_kwargs_scrolledtext_propio_colocacion_scrollbar_horizontal_coord_y = "coord_y"

self.nombre_kwargs_scrolledtext_propio_df_datos = "df_datos"
self.nombre_kwargs_scrolledtext_propio_columna_df_para_informar = "columna_df_para_informar"

self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_linea_completa = "lista_dicc_tag_linea_completa"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_linea_completa_nombre_tag = "nombre_tag"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_linea_completa_columna_df_tag_aplicar = "columna_df_tag_aplicar"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_linea_completa_case_sensitive = "case_sensitive"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_linea_completa_dicc_config = "dicc_config"

self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa = "lista_dicc_tag_caracteres_cambiantes_comparativa"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_nombre_tag = "nombre_tag"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_columna_df_filtro_registros_aplicar_tag = "columna_df_filtro_registros_aplicar_tag"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_columna_df_filtro_registros_aplicar_tag_valor = "columna_df_filtro_registros_aplicar_tag_valor"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_columna_df_comparar_1 = "columna_df_comparar_1"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_columna_df_comparar_2 = "columna_df_comparar_2"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_case_sensitive = "case_sensitive"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_marcar_toda_linea_si_todo_varia = "marcar_toda_linea_si_todo_varia"
self.nombre_kwargs_scrolledtext_propio_lista_dicc_tag_caracteres_cambiantes_comparativa_dicc_config = "dicc_config"
```

La clase **scrolledtext_propio** al ser clase hija de **gui_tkinter_widgets** hereda el método propio **config_atributos** por lo que se puede configurar atributos nativos tkinter y propios definidos en la clase madre siempre y cuando sean compatibles con el objeto scrolledtext de tkinter.

ANEXOS

EJEMPLOS DE USO

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

Los ejemplos de uso que se detallan en este anexo son los siguientes:

INDICE	EJEMPLO	PÁGINAS
1	root	26 - 27
2	frames dentro del root	28 - 29
3	botón	30 - 35
4	frames scrollables dentro del root	36 - 45
5	label	46 - 47
6	entry	48 - 49
7	entry con reglas de validación	50 - 52
8	entry fecha con inclusión botón calendario calendario	53 - 56
9	combobox	57 - 58
10	listbox	59 - 61
11	treeview	62 - 65
12	nuevo root tras pulsar un botón (toplevel)	66 - 68
13	Aplicar a un widget varias rutinas de evento bind (ejemplo combobox)	69 - 71
14	Aplicar a una variable enlace (stringvar) varias rutinas de evento en modo trace (ejemplo lista opciones de combobox dependientes del valor escogido en otro combobox)	72 - 74
15	scrolledtext desde un string y desde un dataframe	75 - 77
16	scrolledtext desde un dataframe aplicando tags	78 - 84
17	Configuración de atributos nativos tkinter no integrados en mi_sistema_tkinter (messagebox, filedialog etc) dentro de clases propias en mi_proyecto	85

Los bloques de código de los pantallazos que salen a continuación para ilustrar cada ejemplo figuran en el fichero **EJEMPLO_USO_tkinter_utils_v1_0** disponible en el repositorio Github.

En **mi_proyecto** deben figurar estas importaciones (el alias que se le da a **mi_sistema_tkinter** es de libre elección por el usuario).

```
import pandas as pd
import tkinter as tk
from tkinter import messagebox, filedialog as fd
import pandas as pd
from tkcalendar import Calendar

#importar el modulo (OBLIGATORIO)
import UTILS as mod_utils
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1 EJEMPLO 1 – creación de un root

Se usa la clase `gui_tkinter_widgets`.

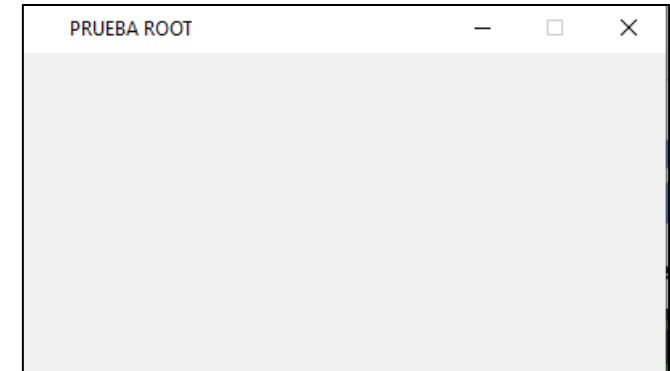
```
#####
# EJEMPLO 1 - root
#####

ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                             if getattr(sys, 'frozen', False)
                             else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

#atributos propios
# --> dicc_config_root
# --> tupla_geometry (width, height)
kwargs_config_root_1 = {"dicc_config_root":
                        {
                            "title": "PRUEBA ROOT",
                            "iconbitmap": ico_tapar_pluma_tkinter,
                            "tupla_geometry": (400, 200),
                            "resizable": (0, 0)
                        }
}

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root" **kwargs_config_root_1)

#el mainloop debe hacerse sobre objeto widget del root creado
#(usar el atributo widget objeto de la clase gui_tkinter_widgets)
root.widget_objeto.mainloop()
```



cuando se crea un root el **master** de la clase `gui_tkinter_widgets` se le pasa el valor **None**

informar **root**

el root creado al hacerse mediante la clase `gui_tkinter_widgets` y no directamente desde tkinter (`tk.Tk()`) requiere pasarle el objeto del widget (**widget_objeto**) definido en el constructor de la clase

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

1 EJEMPLO 1 – creación de un root

```
#####
# EJEMPLO 1 - root
#####

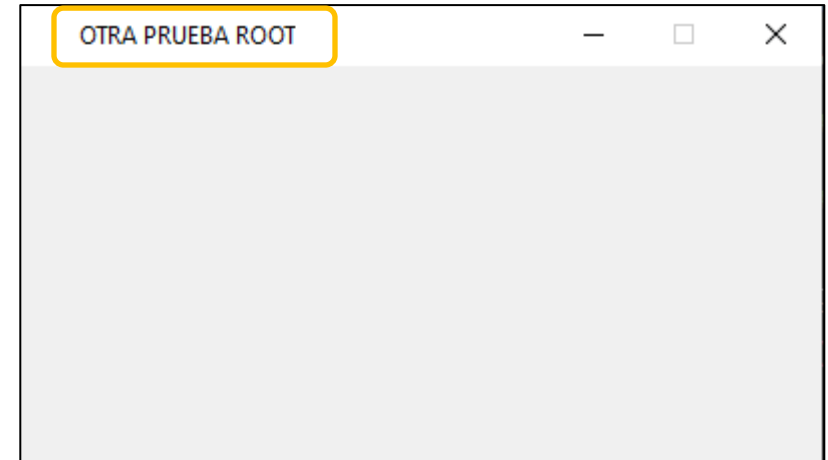
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                            if getattr(sys, 'frozen', False)
                            else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

#atributos propios
# --> dicc_config_root
# --> tupla_geometry (width, height)
kwargs_config_root_1 = {"dicc_config_root":
                        {"title": "PRUEBA ROOT"
                        , "iconbitmap": ico_tapar_pluma_tkinter
                        , "tupla_geometry": (400, 200)
                        , "resizable": (0, 0)
                        }
                        }

kwargs_config_root_2 = {"dicc_config_root":
                        {"title": "OTRA PRUEBA ROOT"
                        , "iconbitmap": ico_tapar_pluma_tkinter
                        , "tupla_geometry": (400, 200)
                        , "resizable": (0, 0)
                        }
                        }

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root_1)
#se puede reconfigurar el root una vez creado mediante el metodo config_atributos de la clase gui_tkinter_widgets
root.config_atributos(**kwargs_config_root_2)

#el mainloop debe hacerse sobre objeto widget del root creado
#(usar el atributo widget_objeto de la clase gui_tkinter_widgets)
root.widget_objeto.mainloop()
```



Se puede usar el metodo propio **config_atributos** de la clase **gui_tkinter_widgets** para re-configurar el root después de su creación

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

2 EJEMPLO 2 – creación frames dentro de un root

Se usa la clase `gui_tkinter_widgets`.

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

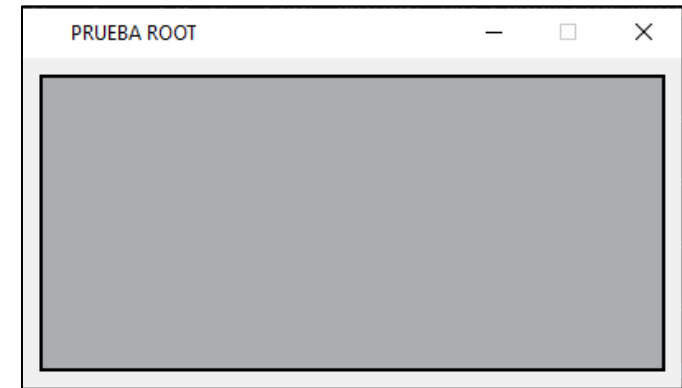
class ClaseVentanaInicio:
    def __init__(self, master):
        self.master = master

        self.kwargs_config_frame = {"width": 380
                                     , "height": 180
                                     , "bg": "#ACAD81"
                                     , "bd": 2
                                     , "relief": "solid"
                                     , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                                    }

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)

if __name__ == "__main__":
    kwargs_config_root = {"dicc_config_root":
                          {"title": "PRUEBA ROOT"
                           , "iconbitmap": ico_tapar_pluma_tkinter
                           , "tupla_geometry": (400, 400)
                           , "resizable": (0, 0)
                          }
                         }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    ClaseVentanaInicio(root)
    root.widget_objeto.mainloop()
```



informar frame

el **master** de la clase `gui_tkinter_widgets` tiene que ser el master de la clase propia creada de aquí que salga **`self.master.widget_objeto`**

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

2 EJEMPLO 2 – creación frames dentro de un root

CASO 2. Directamente en el módulo de *mi_proyecto*

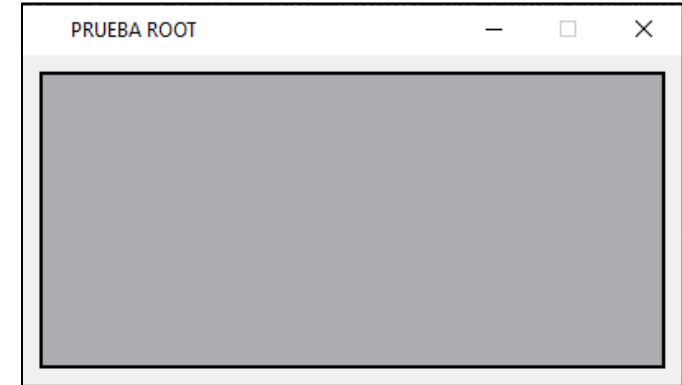
```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                             if getattr(sys, 'frozen', False)
                             else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
                      {"title": "PRUEBA ROOT",
                       "iconbitmap": ico_tapar_pluma_tkinter,
                       "tupla_geometry": (400, 200),
                       "resizable": (0, 0)}
                      }

kwargs_config_frame = {"width": 380,
                       "height": 180,
                       "bg": "#ACADB1",
                       "bd": 2,
                       "relief": "solid",
                       "dicc_colocacion": {"metodo": "place", "coord x": 10, "coord y": 10}}

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)

root.widget_objeto.mainloop()
```



informar frame

el master de la clase **gui_tkinter_widgets** tiene que ser el objeto del root creado de aquí que salga **root.widget_objeto**

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

- 3 EJEMPLO 3 – creación botón** *se presenta la configuración de los botones en este manual antes que el resto de widgets porque en los ejemplos del resto de widgets en algunos casos se integran botones.*

Se usa la clase `gui_tkinter_widgets`.

Las rutinas que se asocian a los botones se hacen en el **momento en el que el usuario interactúa con ellos** no en el momento de su creación (en la clase `gui_tkinter_widgets` se hace mediante asignaciones **lambda**).

mi_sistema_tkinter está diseñado para usarse en **mi_proyecto** mediante el uso de clases propias. No obstante, existe la posibilidad de configurar botones directamente en **mi_proyecto** sin usar clases propias, pero **no es bonito** porque requiere declarar las rutinas asociadas a cada botón antes de crear los mismos.

Cuando se crea un botón hay que pasarle en la llamada a la clase `gui_tkinter_widgets`, al parámetro **self_clase_gui_donde_call_rutina** el entorno desde el cual se crea el botón:

- **CASO 1:** el botón se crea en **mi_proyecto** en una clase propia (en adelante **clase_padre**) por lo que hay que pasarle el **self**.
- **CASO 2:** el botón se crea directamente en **mi_proyecto** sin pasar por una **clase_padre** por lo que hay que pasarle `sys.modules[__name__]` (requiere importar en **mi_proyecto** la librería **sys**).

Esto se debe a que **mi_sistema_tkinter** se encuentra en otro módulo Python al de **mi_proyecto** y al intentar recuperar el objeto asociado a la rutina declarada como **string** en los kwargs dentro de **mi_sistema_tkinter** (se hace mediante `getattr`) no reconoce el atributo sino.

Se puede re-configurar botones tras su creación usando el método propio **config_atributos** de la clase `gui_tkinter_widgets`.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3 EJEMPLO 3 – creación botón

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

png_para_boton = (os.path.join(sys._MEIPASS, "png_para_boton.png")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "png_para_boton.png"))

class clase_ventana_inicio:
    def __init__(self, master):
        self.master = master

        self.kwarg_config_frame = {"width": 380
        , "height": 180
        , "bg": "#ACAD81"
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.kwarg_config_boton_1 = {"text": "botón"
        , "width": 5
        , "bg": "black"
        , "fg": "white"
        , "controltip_text": "este boton solo imprime"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        , "dicc_rutina": {
            "rutina": "def_rutina_boton_1"
            , "parametros_args": ("args_1",)
            , "parametros_kwarg": {"kwarg_1": "prueba_1", "kwarg_2": "prueba_1"}
        }}

        self.kwarg_config_combobox = {"font": ("Calibri", 10)
        , "width": 15
        , "bd": 2
        , "relief": "solid"
        , "justify": tk.LEFT
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 50}
        , "combobox_lista_opciones": ["A", "B", "C"]}

        self.kwarg_config_boton_2 = {"width": 40
        , "dicc_imagen": {"png_imagen": png_para_boton , "tupla_imagen_resize": (23, 23)}
        , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 48}
        , "dicc_rutina": {
            "rutina": "def_rutina_boton_2"
            , "parametros_args": (lambda widget: self.combobox.widget.obieto.get(),)
        }}
```

pone "combobox" porque es el nombre que se le da al widget

1 El kwarg de configuración del **botón 1** incluye los atributos propios:

- **controltip_text**: para mostrar en la GUI un mensaje en pantalla cuando el usuario coloca el cursor del ratón sobre el botón
- **dicc_rutina**: se asigna una rutina al botón con parámetros:
 - **args estáticos**: la rutina se ejecuta solo para el valor que se define aquí para el parámetro posicional de la rutina asociada
 - **kwargs**: la rutina se ejecuta solo para los valores que se definen aquí para los kwargs de la rutina asociada

2 Se crea un combobox (ver en apartado combobox en el Anexo del presente documento) con la lista de opciones: A, B y C

3 El kwarg de configuración del **botón 2** incluye los atributos propios:

- **dicc_imagen_boton**: permite incrustar una imagen (desde una ruta a un archivo .png) en el botón y redimensionar la imagen incrustada mediante una tupla.
- **dicc_rutina**: se asigna una rutina al botón con parámetros:
 - **args dinámicos**: la rutina se ejecuta para el valor que toma su parámetro posicional cuando este valor corresponde a la opción seleccionada en el combobox

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3 EJEMPLO 3 – creación botón

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)

self.boton_1 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_boton_1)
self.boton_2 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_boton_2)

self.combobox = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "combobox", **self.kwargs_config_combobox)

def def_rutina_boton_1(self, opcion_boton_1, **kwargs):
    kwargs_1 = kwargs.get("kwargs_1", None)

    if opcion_boton_1 == "args_1":
        print("prueba aaa")


    elif opcion_boton_1 == "args_2":
        print("prueba bbb")

    if kwargs_1 == "prueba_1":
        print("prueba ccc")

def def_rutina_boton_2(self, opcion_boton_2):
    if opcion_boton_2 == "A":
        print("prueba ddd")

    elif opcion_boton_2 == "B":
        print("prueba eee")

    elif opcion_boton_2 == "C":
        print("prueba fff")
```



- el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **self.frame.widget_objeto** (se puede incorporar también en el root)
- informar **button** en el parámetro **tipo_widget_param**
- informar **self** en el parámetro **self_clase_gui_donde_call_rutina**

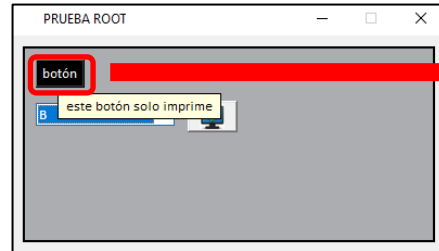
parámetros args

parámetros kwargs

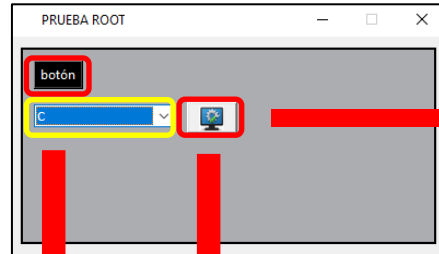
MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3 EJEMPLO 3 – creación botón

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*



aparece un mensaje en pantalla tras configurar **controltiptext** para el botón 1



aparece una imagen incrustada en el botón tras configurar **dicc_imagen_boton** para el botón 2

al pulsar el botón

```
def def_rutina_boton_2(self, opcion_boton_2):  
    if opcion_boton_2 == "A":  
        print("prueba ddd")  
    elif opcion_boton_2 == "B":  
        print("prueba eee")  
    elif opcion_boton_2 == "C":  
        print("prueba fff")
```

Ejecuta el args **dinámico** asociado al valor tomado por el combobox

```
PS C:\Users\user> & C:/U  
prueba fff
```

```
"parametros_args": (lambda widget: self.combobox.widget_objeto.get(),)
```

al pulsar el botón

```
def def_rutina_boton_1(self, opcion_boton_1, **kwargs):  
    kwargs_1 = kwargs.get("kwargs_1", None)  
    if opcion_boton_1 == "args_1":  
        print("prueba aaa")  
    elif opcion_boton_1 == "args_2":  
        print("prueba bbb")  
    if kwargs_1 == "prueba_1":  
        print("prueba ccc")
```

Ejecuta el args **estático** y el kwargs configurados

```
PS C:\Users\user> & C:/Us  
prueba aaa  
prueba ccc
```

```
"parametros_args": ("args_1",)
```

```
"parametros_kwargs": {"kwargs_1": "prueba_1", "kwargs_2": "prueba_1"}
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3 EJEMPLO 3 – creación botón

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

png_para_boton = (os.path.join(sys._MEIPASS, "png_para_boton.png")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "png_para_boton.png"))

kwargs_config_root = {"dicc_config_root":
    {
        "title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
        , "resizable": (0, 0)
    }
}

kwargs_config_frame = {"width": 380
    , "height": 180
    , "bg": "#ACADB1"
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_boton_1 = {"text": "botón"
    , "width": 5
    , "bg": "black"
    , "fg": "white"
    , "controlliptext": "este botón solo imprime"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    , "dicc_rutina":
        {
            "rutina": "def_rutina_boton_1"
            , "parametros_args": ("args_1",)
            , "parametros_kwargs": {"kwargs_1": "prueba_1", "kwargs_2": "prueba_1"}
        }
}

kwargs_config_combobox = {"font": ("Calibri", 10)
    , "width": 15
    , "bd": 2
    , "relief": "solid"
    , "justify": tk.LEFT
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 50}
    , "combobox_lista_opciones": ["A", "B", "C"]
}

kwargs_config_boton_2 = {"width": 40
    , "dicc_imagen": {"png_imagen": png_para_boton, "tupla_imagen_resize": (23, 23)}
    , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 48}
    , "dicc_rutina":
        {
            "rutina": "def_rutina_boton_2"
            , "parametros_args": (lambda widget: combobox.widget_objeto.get(),)
        }
}
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

3 EJEMPLO 3 – creación botón

CASO 2. Directamente en el módulo de *mi_proyecto*

```
modulo_python_actual = sys.modules[__name__]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)

def def_rutina_boton_1(opcion_boton_1, **kwargs):

    kwargs_1 = kwargs.get("kwargs_1", None)

    if opcion_boton_1 == "args_1":
        print("prueba aaa")

    elif opcion_boton_1 == "args_2":
        print("prueba bbb")

    if kwargs_1 == "prueba_1":
        print("prueba ccc")

def def_rutina_boton_2(opcion_boton_2):

    if opcion_boton_2 == "A":
        print("prueba ddd")

    elif opcion_boton_2 == "B":
        print("prueba eee")

    elif opcion_boton_2 == "C":
        print("prueba fff")

combobox = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "combobox", **kwargs_config_combobox)
boton_1 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton_1)
boton_2 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton_2)

root.widget_objeto.mainloop()
```

Hay pasar el entorno del módulo al parámetro **self_clase_gui_donde_call_rutina**.
Este es la expresión: **sys.modules[__name__]**
(requiere importar en el módulo la librería **sys**).

Cuando el código esta encapsulado en una clase, este se lee de arriba para abajo y de abajo para arriba (es decir una llamada o una asignación a un objeto o atributo se puede hacer antes de que dicho objeto o atributo se cree).

Cuando el código se ejecuta directamente en el módulo sin encapsularlo en una clase, este se lee solo de arriba para abajo:

- 1 Hay que crear las rutinas antes que crear los botones (el combobox se puede crear antes de las rutinas)
- 2 Hay que crear el combobox antes que los botones
- 3 Hay que crear los botones por último

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

Se usa la clase `frame_con_scrollbar`.

Se crean 4 frames scrollables. Se agrega un botón para configurar el frame 4 (**violeta**) sin scrollbars.



Como funciona el scrolling?

El scrolling se puede hacer usando las barras de scrolling (scrollbars) o bien clicando en las flechitas.

El scrolling con la rueda del ratón:

TIPO SCROLLING	EXPLICACIÓN
vertical y horizontal	Moviendo la rueda del ratón se hace scrolling vertical. Pulsando la teclas Shift + moviendo la rueda del ratón se hace scrolling horizontal.
vertical	Moviendo la rueda del ratón se hace scrolling vertical.
horizontal	Moviendo la rueda del ratón se hace scrolling horizontal.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_ventana_inicio:

    def __init__(self, master):

        self.master = master
```

```
{ "width": 5000
, "height": 5000
, "bg": "#ACADB1"
, "bd": 2
, "relief": "solid"
, "dicc_colocacion": { "metodo": "place", "coord_x": 100, "coord_y": 100 }
, "dicc_frame_scrollbar": { "width_visible": 450
, "width_total": 2000
, "height_visible": 150
, "height_total": 630
, "tupla_coord_place": (10, 10)
, "velocidad_scrolling": 3
}
}

, { "bg": "#88E271"
, "bd": 2
, "relief": "solid"
, "dicc_frame_scrollbar": { "width_visible": 450
, "width_total": 450
, "height_visible": 150
, "height_total": 2000
, "tupla_coord_place": (500, 10)
, "velocidad_scrolling": 3
}
}
```

Frame 1

No se aplican las dimensiones configuradas con atributos nativos (**amarillo**), solo las que aparecen en **dicc_frame_scrollbar**

width_visible < width_total ➡ scrollbar horizontal

height_visible < height_total ➡ scrollbar vertical

Frame 2

width_visible = width_total ➡ **NO** scrollbar horizontal

height_visible < height_total ➡ scrollbar vertical

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
, {"bg": "#DFC968",  
  "bd": 2,  
  "relief": "solid",  
  "dicc_frame_scrollbar": {"width_visible": 450  
                           , "width_total": 2000  
                           , "height_visible": 150  
                           , "height_total": 150  
                           , "tupla_coord_place": (10, 210)  
                           , "velocidad_scrolling": 3  
                           }  
},  
  
, {"bg": "#E3ABF0",  
  "bd": 2,  
  "relief": "solid",  
  "dicc_frame_scrollbar": {"width_visible": 450  
                           , "width_total": 2000  
                           , "height_visible": 150  
                           , "height_total": 630  
                           , "tupla_coord_place": (500, 210)  
                           , "velocidad_scrolling": 3  
                           }  
}
```

Frame 3

width_visible < width_total ➡ scrollbar horizontal

height_visible = height_total ➡ **NO** scrollbar vertical

Frame 4

width_visible < width_total ➡ scrollbar horizontal

height_visible < height_total ➡ scrollbar vertical

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
self.lista_objetos_frame = []
for ind_kwarg_label, kwarg_frame in enumerate(lista_kwarg_frame):

    kwarg_label = {"text": None
                  , "bg": kwarg_frame["bg"]
                  , "width": 20
                  , "font": ("Calibri", 12, "bold")
                  , "alineacion": "left"
                  , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                  }

    kwarg_label["text"] = ("vertical_y_horizontal" if ind_kwarg_label in [0, 3]
                          else "vertical" if ind_kwarg_label == 1
                          else "horizontal" if ind_kwarg_label == 2
                          else "")

    frame = mod_utils.frame_con_scrollbar(self.master.widget_objeto, **kwarg_frame)
    mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "label", **kwarg_label)

    self.lista_objetos_frame.append(frame)

    for coord in ["x", "y"]:

        kwarg_config_label_frame = {"text": None
                                    , "width": 15
                                    , "bd": 2
                                    , "bg": "black"
                                    , "fg": "white"
                                    , "relief": "solid"
                                    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 40}
                                    }

        for ind in range(1, 20, 1):

            kwarg_config_label_frame["text"] = f"label_{ind}" if ind == 1 else f"label_{ind}_{coord}"
            kwarg_config_label_frame["dicc_colocacion"][f"coord_{coord}"] = (ind - 1) * 150 + 10 if coord == "x" else (ind - 1) * 30 + 40

            mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "label", **kwarg_config_label_frame)

            kwarg_config_label_frame["text"] = None

        kwarg_config_boton = {"text": "modificar frame"
                              , "bg": "red"
                              , "fg": "white"
                              , "width": 15
                              , "dicc_colocacion": {"metodo": "place", "coord_x": 450, "coord_y": 380}
                              , "dicc_rutina": {"rutina": "def_rutina_boton"}
                              }

        self.boton = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **kwarg_config_boton)
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
def def_rutina_boton(self):
    kwargs_frame_modif = {"bg": "#CFE970"}
    # ...
    "dicc_frame_scrollbar": {"width_visible": 450
                             , "width_total": 450
                             , "height_visible": 150
                             , "height_total": 150
                             , "tupla_coord_place": (500, 210)
                             , "velocidad_scrolling": 3
                             }

    kwargs_frame_modif_label = {"text": "Frame cambiado de 'vertical y horizontal' a 'sin scrollbars'"
                                , "bg": "#CFE970"
                                , "fg": "red"
                                , "width": 50
                                , "font": ("Calibri", 12, "bold")
                                , "alineacion": "left"
                                , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                                }

    #se modifica el frame 4 (el de abajo - derecha)
    self.frame_por_modificar = self.lista_objetos_frame[3]
    self.frame_por_modificar.modificaciones(**kwargs_frame_modif)

    mod_utils.gui_tkinter_widgets(self.frame_por_modificar.widget_objeto, tipo_widget_param = "label", **kwargs_frame_modif_label)

    for coord in ["x", "y"]:
        kwargs_config_label_frame = {"text": None
                                     , "width": 15
                                     , "bd": 2
                                     , "bg": "black"
                                     , "fg": "white"
                                     , "relief": "solid"
                                     , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 40}
                                     }

        for ind in range(1, 20, 1):
            kwargs_config_label_frame["text"] = f"label_{ind}" if ind == 1 else f"label_{ind}_{coord}"
            kwargs_config_label_frame["dicc_colocacion"][f"coord_{coord}"] = (ind - 1) * 150 + 10 if coord == "x" else (ind - 1) * 30 + 40

            mod_utils.gui_tkinter_widgets(self.frame_por_modificar.widget_objeto, tipo_widget_param = "label", **kwargs_config_label_frame)

        kwargs_config_label_frame["text"] = None
```

se cambia el color de fondo del frame 4

width_visible = width_total ➡ **NO scrollbar horizontal**

height_visible = height_total ➡ **NO scrollbar vertical**

Se aplica el método propio **modificaciones** de la clase frame_con_scrollbar (dicho método tiene incorporado el método propio **config_atributos** de la clase madre gui_tkinter_widgets lo que habilita el cambio de color de fondo del frame usando el atributo nativo "bg" de tkinter)

```
if __name__ == "__main__":
    kwargs_config_root = {"dicc_config_root":
                          {"title": "PRUEBA ROOT"
                           , "iconbitmap": ico_tapar_pluma_tkinter
                           , "tupla_geometry": (1000, 450)
                           , "resizable": (0, 0)
                          }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_ventana_inicio(root)
    root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
    {"title": "PRUEBA ROOT"
    , "iconbitmap": ico_tapar_pluma_tkinter
    , "tupla_geometry": (1000, 450)
    , "resizable": (0, 0)
    }
}
```

```
{ "width": 5000
, "height": 5000
, "bg": "#ACADB1"
, "bd": 2
, "relief": "solid"
, "dicc_colocacion": { "metodo": "place", "coord_x": 100, "coord_y": 100 }
, "dicc_frame_scrollbar": { "width_visible": 450
    , "width_total": 2000
    , "height_visible": 150
    , "height_total": 630
    , "tupla_coord_place": (10, 10)
    , "velocidad_scrolling": 3
    }
}

{ "bg": "#88E271"
, "bd": 2
, "relief": "solid"
, "dicc_frame_scrollbar": { "width_visible": 450
    , "width_total": 450
    , "height_visible": 150
    , "height_total": 2000
    , "tupla_coord_place": (500, 10)
    , "velocidad_scrolling": 3
    }
}
```

Frame 1

No se aplican las dimensiones configuradas con atributos nativos (**amarillo**), solo las que aparecen en **dicc_frame_scrollbar**

width_visible < width_total ➡ scrollbar horizontal

height_visible < height_total ➡ scrollbar vertical

Frame 2

width_visible = width_total ➡ **NO** scrollbar horizontal

height_visible < height_total ➡ scrollbar vertical

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 2. Directamente en el módulo de *mi_proyecto*

```
, {"bg": "#DFC968",  
  "bd": 2,  
  "relief": "solid",  
  "dicc_frame_scrollbar": {"width_visible": 450  
                           , "width_total": 2000  
                           , "height_visible": 150  
                           , "height_total": 150  
                           , "tupla_coord_place": (10, 210)  
                           , "velocidad_scrolling": 3  
                           }  
},  
  
, {"bg": "#E3ABF0",  
  "bd": 2,  
  "relief": "solid",  
  "dicc_frame_scrollbar": {"width_visible": 450  
                           , "width_total": 2000  
                           , "height_visible": 150  
                           , "height_total": 630  
                           , "tupla_coord_place": (500, 210)  
                           , "velocidad_scrolling": 3  
                           }  
}
```

Frame 3

width_visible < width_total ➡ scrollbar horizontal

height_visible = height_total ➡ **NO** scrollbar vertical

Frame 4

width_visible < width_total ➡ scrollbar horizontal

height_visible < height_total ➡ scrollbar vertical

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 2. Directamente en el módulo de *mi_proyecto*

```
root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)

lista_objetos_frame = []
for ind_kwargs_label, kwargs_frame in enumerate(lista_kwargs_frame):

    kwargs_label = {"text": None
                    , "bg": kwargs_frame["bg"]
                    , "width": 20
                    , "font": ("Calibri", 12, "bold")
                    , "alineacion": "left"
                    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                    }

    kwargs_label["text"] = ("vertical_y_horizontal" if ind_kwargs_label in [0, 3]
                           else "vertical" if ind_kwargs_label == 1
                           else "horizontal" if ind_kwargs_label == 2
                           else "")

    frame = mod_utils.frame_con_scrollbar(root.widget_objeto, **kwargs_frame)
    mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "label", **kwargs_label)

    lista_objetos_frame.append(frame)

    for coord in ["x", "y"]:

        kwargs_config_label_frame = {"text": None
                                     , "width": 15
                                     , "bd": 2
                                     , "bg": "black"
                                     , "fg": "white"
                                     , "relief": "solid"
                                     , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 40}
                                     }

        for ind in range(1, 20, 1):

            kwargs_config_label_frame["text"] = f"label_{ind}" if ind == 1 else f"label_{ind}_{coord}"
            kwargs_config_label_frame["dicc_colocacion"][f"coord_{coord}"] = (ind - 1) * 150 + 10 if coord == "x" else (ind - 1) * 30 + 40

            mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "label", **kwargs_config_label_frame)

            kwargs_config_label_frame["text"] = None
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 2. Directamente en el módulo de *mi_proyecto*

```
def def_rutina_boton():  
    kwargs_frame_modif = {"bg": "#CFE970"  
        , "bd": 2  
        , "relief": "solid"  
        , "dicc_frame_scrollbar": {"width_visible": 450  
            , "width_total": 450  
            , "height_visible": 150  
            , "height_total": 150  
            , "tupla_coord_place": (500, 210)  
            , "velocidad_scrolling": 3  
        }  
    }  
  
    kwargs_frame_modif_label = {"text": "Frame cambiado de 'vertical y horizontal' a 'sin scrollbars'"  
        , "bg": "#CFE970"  
        , "fg": "red"  
        , "width": 50  
        , "font": ("Calibri", 12, "bold")  
        , "alineacion": "left"  
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}  
    }  
  
    #se modifica el frame 4 (el de abajo - derecha)  
    frame_por_modificar = lista_objetos_frame[3]  
    frame_por_modificar.modificaciones(**kwargs_frame_modif)  
  
    mod_utils.gui_tkinter_widgets(frame_por_modificar.widget_objeto, tipo_widget_param = "label", **kwargs_frame_modif_label)  
  
    for coord in ["x", "y"]:  
        kwargs_config_label_frame = {"text": None  
            , "width": 15  
            , "bd": 2  
            , "bg": "black"  
            , "fg": "white"  
            , "relief": "solid"  
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 40}  
        }  
  
        for ind in range(1, 20, 1):  
            kwargs_config_label_frame["text"] = f"label_{ind}" if ind == 1 else f"label_{ind}_{coord}"  
            kwargs_config_label_frame["dicc_colocacion"][f"coord_{coord}"] = (ind - 1) * 150 + 10 if coord == "x" else (ind - 1) * 30 + 40  
  
            mod_utils.gui_tkinter_widgets(frame_por_modificar.widget_objeto, tipo_widget_param = "label", **kwargs_config_label_frame)  
  
            kwargs_config_label_frame["text"] = None
```

se cambia el color de fondo del frame 4

width_visible = width_total ➡ **NO scrollbar horizontal**

height_visible = height_total ➡ **NO scrollbar vertical**

Se aplica el método propio **modificaciones** de la clase frame_con_scrollbar (dicho método tiene incorporado el método propio **config_atributos** de la clase madre gui_tkinter_widgets lo que habilita el cambio de color de fondo del frame usando el atributo nativo "bg" de tkinter)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

4 EJEMPLO 4 – frames scrollables dentro del root

CASO 2. Directamente en el módulo de *mi_proyecto*

```
kwargs_config_boton = {"text": "modificar frame"  
                        , "bg": "red"  
                        , "fg": "white"  
                        , "width": 15  
                        , "dicc_colocacion": {"metodo": "place", "coord_x": 450, "coord_y": 380}  
                        , "dicc_rutina":  
                          {"rutina": "def_rutina_boton"}  
                        }  
  
modulo_python_actual = sys.modules[__name__]  
  
boton = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton)  
  
root.widget_objeto.mainloop()
```

El botón se ha de declarar después de declarar la rutina que se le asocia (página anterior).

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

5 EJEMPLO 5 – creación label

Se usa la clase `gui_tkinter_widgets`.

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
#####
# clase propia
#####

ico_tapar_pluma_tkinter = (os.path.join(sys_MEIPASS, "ico_tapar_pluma_tkinter.ico")
                           if getattr(sys, 'frozen', False)
                           else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_ventana_inicio:
    def __init__(self, master):
        self.master = master

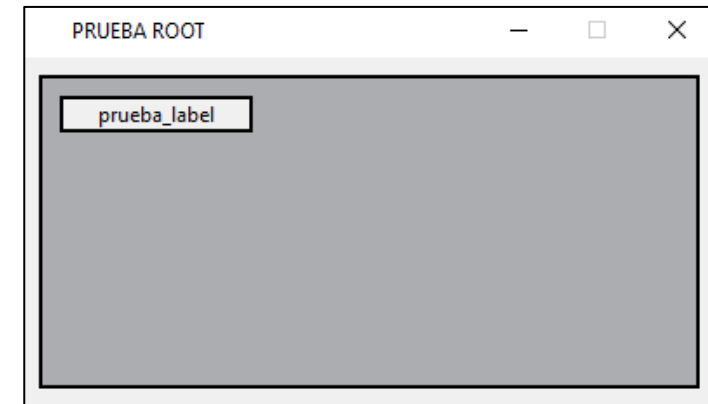
        self.kwargs_config_frame = {"width": 380
                                     , "height": 180
                                     , "bg": "#ACAD81"
                                     , "bd": 2
                                     , "relief": "solid"
                                     , "colocacion_dicc": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.kwargs_config_label = {"text": "prueba_label"
                                     , "width": 15
                                     , "bd": 2
                                     , "relief": "solid"
                                     , "colocacion_dicc": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)
        self.label = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "label", **self.kwargs_config_label)
        self.strvar_label = self.label.variable_enlace
        self.strvar_label.set("prueba stringvar")

    if __name__ == "__main__":
        kwargs_config_root = {"dicc": {"title": "PRUEBA ROOT"
                                       , "iconbitmap": ico_tapar_pluma_tkinter
                                       , "tuple_geometry": (400, 200)
                                       , "resizable": (0, 0)}}

        root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
        clase_ventana_inicio(root)
        root.widget_objeto.mainloop()
```



el **master** de la clase `gui_tkinter_widgets` tiene que ser el objeto del frame creado de aquí que salga ***self.frame.widget_objeto*** (se puede hacer también en el root)

informar **label**

variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

5 EJEMPLO 5 – creación label

CASO 2. Directamente en el módulo de *mi_proyecto*

```
#####
# directamente en el modulo
#####

ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                            if getattr(sys, 'frozen', False)
                            else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
                      {"title": "PRUEBA ROOT"
                        , "iconbitmap": ico_tapar_pluma_tkinter
                        , "tupla_geometry": (400, 200)
                        , "resizable": (0, 0)
                      }
                      }

kwargs_config_frame = {"width": 380
                       , "height": 180
                       , "bg": "#ACADB1"
                       , "bd": 2
                       , "relief": "solid"
                       , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                       }

kwargs_config_label = {"text": "prueba_label"
                       , "width": 15
                       , "bd": 2
                       , "relief": "solid"
                       , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                       }

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
label = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "label", **kwargs_config_label)

strvar_label = label.variable_enlace
strvar_label.set("prueba stringvar")

root.widget_objeto.mainloop()
```

informar **label**

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **frame.widget_objeto** (se puede incorporar también en el root)

variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

6 EJEMPLO 6 – creación entry

Se usa la clase `gui_tkinter_widgets`.

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
#####
# clase propia
#####

ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                            if getattr(sys, 'frozen', False)
                            else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_ventana_inicio:

    def __init__(self, master):

        self.master = master

        self.kwargs_config_frame = {"width": 380
                                     , "height": 180
                                     , "bg": "#ACAD81"
                                     , "bd": 2
                                     , "relief": "solid"
                                     , "colocacion_dicc": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

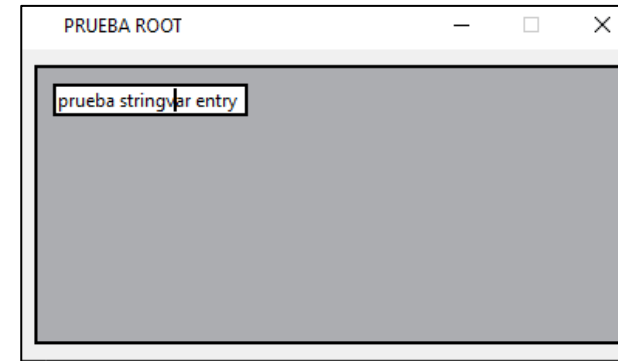
        self.kwargs_config_entry = {"width": 15
                                     , "bd": 2
                                     , "relief": "solid"
                                     , "colocacion_dicc": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.frame = mod_utils.gui_tkinter_widgets(self.master, tipo_widget_objeto = "frame", **self.kwargs_config_frame)
        self.entry = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "entry", **self.kwargs_config_entry)
        self.strvar_entry = self.entry.variable_enlace
        self.strvar_entry.set("prueba stringvar")

if __name__ == "__main__":

    kwargs_config_root = {"dicc_config_root":
                          {"title": "PRUEBA ROOT"
                           , "iconbitmap": ico_tapar_pluma_tkinter
                           , "tupla_geometry": (400, 200)
                           , "resizable": (0, 0)}

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_ventana_inicio(root)
    root.widget_objeto.mainloop()
```



el **master** de la clase `gui_tkinter_widgets` tiene que ser el objeto del frame creado de aquí que salga ***self.frame.widget_objeto*** (se puede hacer también en el root)

informar **entry**

variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

6 EJEMPLO 6 – creación entry

CASO 2. Directamente en el módulo de *mi_proyecto*

```
#####
# directamente en el modulo
#####

ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
{
"title": "PRUEBA ROOT"
, "iconbitmap": ico_tapar_pluma_tkinter
, "tupla_geometry": (400, 200)
, "resizable": (0, 0)
}
}

kwargs_config_frame = {"width": 380
, "height": 180
, "bg": "#ACADB1"
, "bd": 2
, "relief": "solid"
, "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_entry = {"width": 15
, "bd": 2
, "relief": "solid"
, "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
entry = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "entry", **kwargs_config_entry)

strvar_entry = entry.variable_enlace
strvar_entry.set("prueba stringvar entry")

root.widget_objeto.mainloop()
```

informar **entry**

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **frame.widget_objeto** (se puede incorporar también en el root)

variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

7 EJEMPLO 7 – creación entry con reglas de validación

Se usa la clase `entry_propio`.

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_ventana_inicio:

    def __init__(self, master):

        self.master = master

        self.kwargs_config_frame = {"width": 380
        , "height": 180
        , "bg": "#ACAD81"
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.kwargs_config_entry_1 = {"width": 15
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}}
        self.dicc_entry_1 = {"formato_validacion": "entero_positivo"
        , "titulo_messagebox_warning": "titulo messagebox"}

        self.kwargs_config_entry_2 = {"width": 15
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 10}}

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)

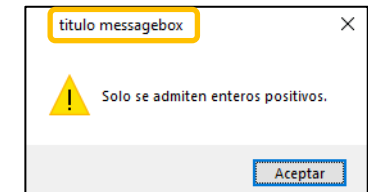
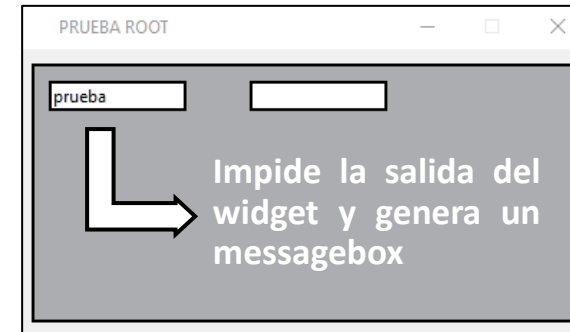
        self.entry_1 = mod_utils.entry_propio(self.frame.widget_objeto, **self.kwargs_config_entry_1)
        self.entry_2 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "entry", **self.kwargs_config_entry_2)

        self.strvar_entry = self.entry_1.variable_enlace
```

```
if __name__ == "__main__":

    kwargs_config_root = {"dicc_config": {"title": "PRUEBA ROOT"
    , "tuple_geometry": (400, 200)
    , "resizable": (0, 0)}

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_ventana_inicio(root)
    root.widget_objeto.mainloop()
```



se configura que el entry solo admita enteros positivos

clase `entry_propio`

el `master` de la clase `gui_tkinter_widgets` tiene que ser el objeto del frame creado de aquí que salga `self.frame.widget_objeto` (se puede incorporar también en el root)

Variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

7 EJEMPLO 7 – creación entry con reglas de validación

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
    {
        "title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
        , "resizable": (0, 0)
    }
}

kwargs_config_frame = {"width": 380
    , "height": 180
    , "bg": "#ACAD81"
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_entry_1 = {"width": 15
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    , "dicc_entry":
        {
            "formato_validacion": "entero_positivo"
            , "titulo_messagebox_warning": "titulo messagebox"
        }
}

kwargs_config_entry_2 = {"width": 15
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 10}
}

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
entry_1 = mod_utils.entry_propio(frame.widget_objeto, **kwargs_config_entry_1)
entry_2 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "entry", **kwargs_config_entry_2)

strvar_entry = entry_1.variable_enlace

strvar_entry.set("prueba")

root.widget_objeto.mainloop()
```

se configura que el entry solo admita enteros positivos

clase **entry_propio**

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **frame.widget_objeto** (se puede incorporar también en el root)

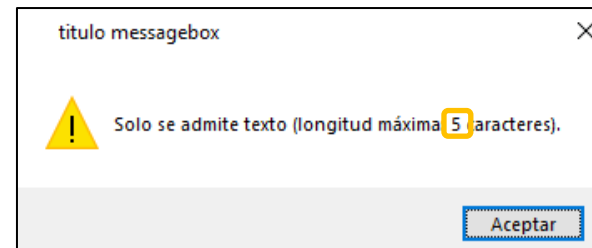
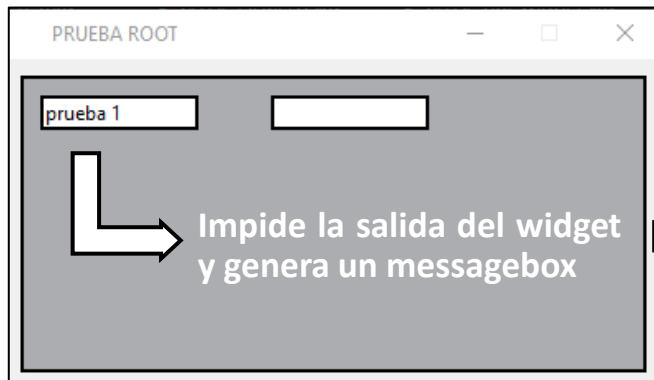
Variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

7 EJEMPLO 7 – creación entry con reglas de validación

Tanto para el CASO 1 como para el CASO 2, el kwargs de configuración de un entry donde se requiere un **texto de una longitud máxima** es el siguiente:

```
self.kwargs_config_entry_1 = {"width": 15
                              , "bd": 2
                              , "relief": "solid"
                              , "dicc_colocacion": {"metodo": "place", "coord x": 10, "coord y": 10}
                              , "dicc_entry": {
                                  {"formato_validacion": "texto"
                                   , "texto_longitud_maxima": 5
                                   , "titulo_messagebox_warning": "titulo messagebox"
                                  }
                              }
                              }
```



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

8 EJEMPLO 8 – creación entry fecha con inclusión botón calendario

Se usa la clase `entry_propio`.



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

8 EJEMPLO 8 – creación entry fecha con inclusión botón calendario

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                            if getattr(sys, 'frozen', False)
                            else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_ventana_inicio:

    def __init__(self, master):

        self.master = master

        self.kwarg_config_frame = {"width": 380
                                    , "height": 180
                                    , "bg": "#ACAD81"
                                    , "bd": 2
                                    , "relief": "solid"
                                    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.kwarg_config_entry = {"width": 15
                                    , "bd": 2
                                    , "relief": "solid"
                                    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                                    , "dicc_entry": {
                                        "formato_validacion": "fecha_ddmmaaaa"
                                        , "titulo_messagebox_warning": "titulo messagebox"
                                        , "calendario_tupla_coord_place_y_width": (120, 10, 5)
                                        , "calendario_iconbitmap": ico_tapar_pluma_tkinter
                                    }}

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwarg_config_frame)
        self.entry = mod_utils.entry_propio(self.frame.widget_objeto, **self.kwarg_config_entry)
        self.strvar_entry = self.entry.variable_enlace

if __name__ == "__main__":

    kwarg_config_root = {"dicc_config_root": {
        "title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
    }}

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwarg_config_root)
    clase_ventana_inicio(root.widget_objeto).mainloop()
```

ATRIBUTO PROPIO	OBLIGARIEDAD	DESCRIPCIÓN
formato_validacion	obligatorio	fecha_ddmmaaaa o fecha_yyyymmdd
titulo_messagebox_warning	opcional	el texto que el usuario desee
calendario_tupla_coord_place_y_width	obligatorio	tupla de 3 ítems numéricos positivos para colocar el calendario: (coordenadas x, coordenadas y, width)
calendario_iconbitmap	obligatorio	ruta archivo .ico para tapar la pluma tkinter en el calendario

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **self.frame.widget_objeto** (se puede incorporar también en el root)

clase **entry_propio**

Variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

8 EJEMPLO 8 – creación entry fecha con inclusión botón calendario

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
    {"title": "PRUEBA ROOT"
    , "iconbitmap": ico_tapar_pluma_tkinter
    , "tupla_geometry": (400, 200)
    , "resizable": (0, 0)
    }
}

kwargs_config_frame = {"width": 380
    , "height": 180
    , "bg": "#ACADB1"
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_entry = {"width": 15
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    , "dicc_entry":
        {"formato_validacion": "fecha_ddmmaaaa"
        , "titulo_messagebox_warning": "titulo messagebox"
        , "calendario_tupla_coord_place_y_width": (120, 10, 5)
        , "calendario_iconbitmap": ico_tapar_pluma_tkinter
        }
}

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
entry = mod_utils.entry_propio(frame.widget_objeto, **kwargs_config_entry)

strvar_entry = entry.variable_enlace

strvar_entry.set("prueba")

root.widget_objeto.mainloop()
```

ATRIBUTO PROPIO	OBLIGARIEDAD	DESCRIPCIÓN
formato_validacion	obligatorio	fecha_ddmmaaaa o fecha_yyyymmdd
titulo_messagebox_warning	opcional	el texto que el usuario desee
calendario_tupla_coord_place_y_width	obligatorio	tupla de 3 items numéricos positivos para colocar el calendario: (coordenadas x, coordenadas y, width)
calendario_iconbitmap	obligatorio	ruta archivo .ico para tapar la pluma tkinter en el calendario

clase **entry_propio**

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **frame.widget_objeto** (se puede incorporar también en el root)

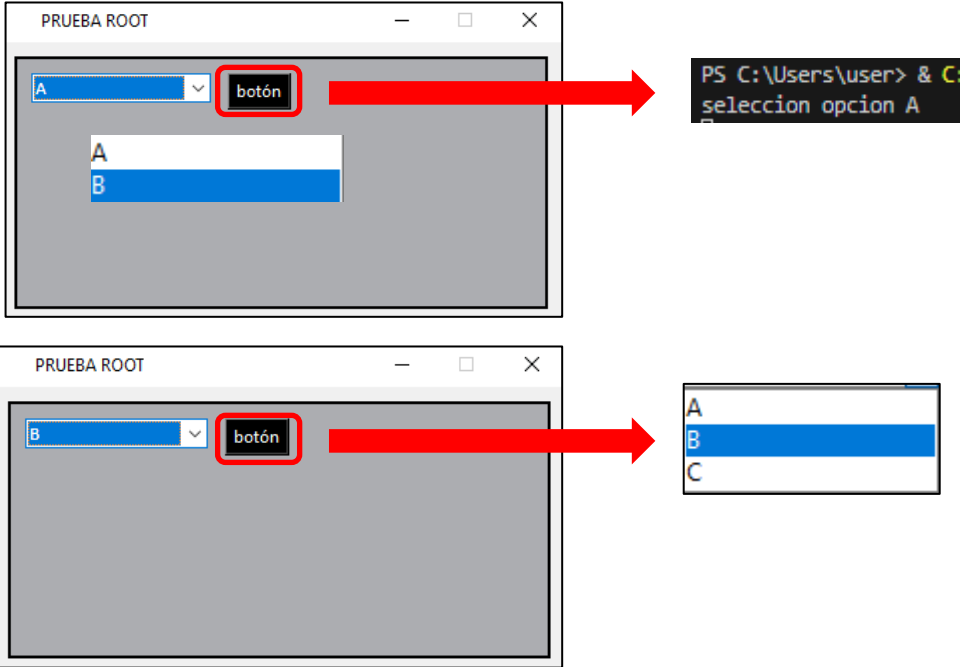
Variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

9

Se usa la clase `gui_tkinter_widgets`.

En el ejemplo se crea un combobox con 2 opciones posibles (A y B) y un botón que al pulsarlo imprime un texto relacionado con la opción seleccionada o re-actualiza la lista de opciones del combobox.



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

9 EJEMPLO 9 – creación combobox

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_ventana_inicio:
    def __init__(self, master):
        self.master = master

        self.kwargs_config_frame = {
            "width": 380
            , "height": 180
            , "bg": "#ACAD01"
            , "bd": 2
            , "relief": "solid"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwargs_config_combobox = {
            "font": ("Calibri", 10)
            , "width": 15
            , "bd": 2
            , "relief": "solid"
            , "justify": tk.LEFT
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
            , "combobox_lista_opciones": ["A", "B"]
        }

        self.kwargs_config_boton = {
            "text": "botón"
            , "width": 5
            , "bg": "black"
            , "fg": "white"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 10}
            , "dicc_rutina": {
                "rutina": "def_rutina_boton"
                , "parametros_args": (lambda widget: self.combobox.widget_objeto.get(),)
            }
        }

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)
        self.combobox = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "combobox", **self.kwargs_config_combobox)
        self.boton = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_boton)

        self.strvar_combobox = self.combobox.variable_enlace

    def def_rutina_boton(self, opcion_boton):
        if opcion_boton == "A":
            print("seleccion opcion A")

        elif opcion_boton == "B":
            self.combobox.config_tributos(**{"combobox_lista_opciones": ["A", "B", "C"]})

if __name__ == "__main__":
    kwargs_config_root = {
        "title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
        , "resizable": (0, 0)
    }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)

    clase_ventana_inicio(root)
    root.widget_objeto.mainloop()
```

informar **combobox**

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **self.frame.widget_objeto** (se puede incorporar también en el root)

Variable de enlace (stringvar)

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

9 EJEMPLO 9 – creación combobox

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
    {
        "title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
        , "resizable": (0, 0)
    }
}

kwargs_config_frame = {"width": 380
    , "height": 180
    , "bg": "#ACAD81"
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_combobox = {"font": ("Calibri", 10)
    , "width": 15
    , "bd": 2
    , "relief": "solid"
    , "justify": tk.LEFT
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    , "combobox_lista_opciones": ["A", "B"]
}

kwargs_config_boton = {"text": "botón"
    , "width": 5
    , "bg": "black"
    , "fg": "white"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 10}
    , "dicc_rutina":
        {
            "rutina": "def_rutina_boton"
            , "parametros_args": (lambda widget: combobox.config_textvariable())
        }
}

modulo_python_actual = sys.modules[__name__]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
combobox = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "combobox", **kwargs_config_combobox)

strvar_combobox = combobox.variable_enlace

def def_rutina_boton(opcion_boton):
    if opcion_boton == "A":
        print("seleccion opcion A")

    elif opcion_boton == "B":
        combobox.config_textvariable(**{"combobox_lista_opciones": ["A", "B", "C"]})

boton = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton)
root.widget_objeto.mainloop()
```

informar **combobox**

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **frame.widget_objeto** (se puede incorporar también en el root)

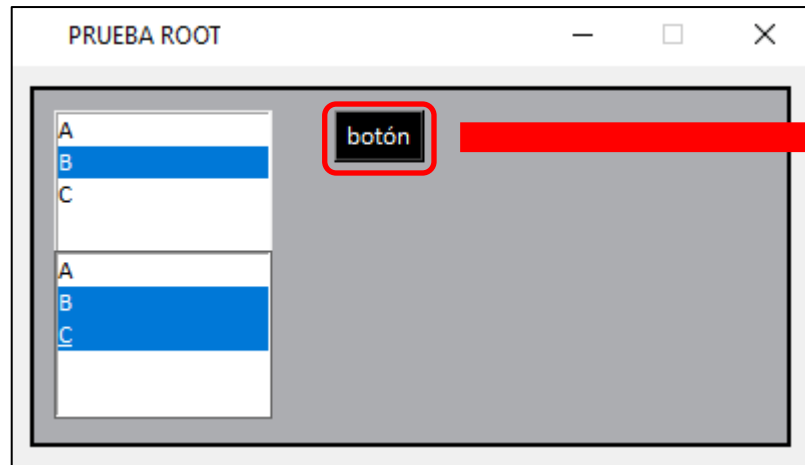
MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

10

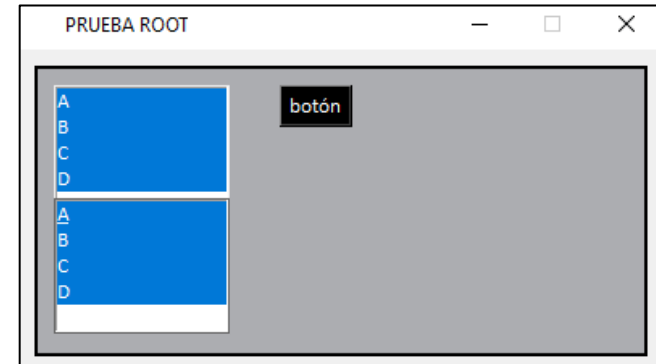
EJEMPLO 10 – creación listbox

Se usa la clase `gui_tkinter_widgets`.

En el ejemplo se crean 2 listbox (uno de selección simple y otro de selección múltiple, ambos con 3 opciones posibles: A, B o C) y un botón que al pulsarlo imprime las opciones seleccionadas en cada uno y re-actualiza sus listas de opciones agregando la opción D y selecciona todos los ítems de ambos.



```
PS C:\Users\user> & C:/U  
['B']  
['B', 'C']
```



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

10 EJEMPLO 10 – creación listbox

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **self.frame.widget_objeto** (se puede incorporar también en el root)

informar listbox

```
ico_tapar_pluma_tkinter = (os.path.join(sys.WEPATH, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, "frozen", False):
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class ClaseVentanaInicio:
    def __init__(self, master):
        self.master = master

        self.kwargs_config_root = {"dicc_config_root":
            {
                "title": "PRUEBA ROOT",
                "iconbitmap": ico_tapar_pluma_tkinter,
                "tuple_geometry": (400, 200),
                "resizable": (0, 0)
            }
        }

        self.kwargs_config_frame = {"width": 380,
            "height": 180,
            "bg": "#ACAD01",
            "bd": 2,
            "relief": "solid",
            "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwargs_config_listbox_seleccion_simple = {"font": ("Calibri", 10),
            "width": 15,
            "height": 5,
            "selectmode": "single",
            "exportselection": False,
            "listbox_lista_items": ["A", "B", "C"]
        }

        self.kwargs_config_listbox_seleccion_multiple = {"font": ("Calibri", 10),
            "width": 15,
            "height": 5,
            "selectmode": "multiple",
            "exportselection": False,
            "listbox_lista_items": ["A", "B", "C"]
        }

        self.kwargs_config_boton = {"text": "botón",
            "font": ("Calibri", 10),
            "width": 15,
            "height": 5,
            "bg": "black",
            "fg": "white",
            "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 10},
            "dicc_rutina": {"rutina": "def_rutina_listbox"}
        }

        if __name__ == "__main__":
            kwargs_config_root = {"dicc_config_root":
                {
                    "title": "PRUEBA ROOT",
                    "iconbitmap": ico_tapar_pluma_tkinter,
                    "tuple_geometry": (400, 200),
                    "resizable": (0, 0)
                }
            }

            root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
            ClaseVentanaInicio(root)
            root.widget_objeto.mainloop()
```

```
self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)

self.listbox_simple = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "listbox", **self.kwargs_config_listbox_seleccion_simple)
self.listbox_multiple = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "listbox", **self.kwargs_config_listbox_seleccion_multiple)

self.boton = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_boton)

def def_rutina_listbox(self):
    lista_items_seleccionados_simple = self.listbox_simple.config_tributos(**{"listbox_lista_items_seleccionados": True})
    lista_items_seleccionados_multiple = self.listbox_multiple.config_tributos(**{"listbox_lista_items_seleccionados": True})

    print(lista_items_seleccionados_simple)
    print(lista_items_seleccionados_multiple)

    self.listbox_simple.config_tributos(**{"listbox_lista_items": ["A", "B", "C", "D"]})
    self.listbox_multiple.config_tributos(**{"listbox_lista_items": ["A", "B", "C", "D"]})

    self.listbox_simple.config_tributos(**{"listbox_seleccionar_todo_o_nada": True, "exportselection": False})
    self.listbox_multiple.config_tributos(**{"listbox_seleccionar_todo_o_nada": True, "exportselection": False})
```

El método **config_tributos** permite:

- recuperar la lista de ítems seleccionados (atributo propio **listbox_lista_items_seleccionados**). Aquí funciona como **función**.
- configurar la lista de opciones del listbox (atributo propio **listbox_lista_items**).
- Seleccionar todo o nada en el listbox (atributo propio **listbox_seleccionar_todo_o_nada** marcando **True** para todo y **False** para nada).

(*) el atributo nativo **exportselection (False)** impide que se desmarquen las opciones de un listbox cuando se interactúa con otro widget.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

10

EJEMPLO 10 – creación listbox

CASO 2. Directamente en el módulo de *mi_proyecto*

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **frame.widget_objeto** (se puede incorporar también en el root)

informar listbox

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
    {"title": "PRUEBA ROOT"
    , "iconbitmap": ico_tapar_pluma_tkinter
    , "tupla_geometry": (400, 200)
    , "resizable": (0, 0)
    }
}

kwargs_config_frame = {"width": 380
    , "height": 180
    , "bg": "#ACAD81"
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_listbox_seleccion_simple = {"font": ("Calibri", 10)
    , "width": 15
    , "height": 5
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    , "selectmode": "single"
    , "exportselection": False
    , "listbox_lista_items": ["A", "B", "C"]}

kwargs_config_listbox_seleccion_multiple = {"font": ("Calibri", 10)
    , "width": 15
    , "height": 5
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 80}
    , "selectmode": "multiple"
    , "exportselection": False
    , "listbox_lista_items": ["A", "B", "C"]}

kwargs_config_boton = {"text": "botón"
    , "font": ("Calibri", 10)
    , "bg": "black"
    , "fg": "white"
    , "width": 5
    , "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 10}
    , "dicc_rutina": {"rutina": "def_rutina_listbox"}}
```

```
modulo_python_actual = sys.modules[__name__]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)

listbox_simple = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "listbox", **kwargs_config_listbox_seleccion_simple)
listbox_multiple = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "listbox", **kwargs_config_listbox_seleccion_multiple)

def def_rutina_listbox():
    lista_items_seleccionados_simple = listbox_simple.config_tributos(**{"listbox_lista_items_seleccionados": True})
    lista_items_seleccionados_multiplke = listbox_multiple.config_tributos(**{"listbox_lista_items_seleccionados": True})

    print(lista_items_seleccionados_simple)
    print(lista_items_seleccionados_multiplke)

    listbox_simple.config_tributos(**{"listbox_lista_items": ["A", "B", "C", "D"]})
    listbox_multiple.config_tributos(**{"listbox_lista_items": ["A", "B", "C", "D"]})

    listbox_simple.config_tributos(**{"listbox_seleccionar_todo_o_nada": True, "exportselection": False})
    listbox_multiple.config_tributos(**{"listbox_seleccionar_todo_o_nada": True, "exportselection": False})

boton = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton)
root.widget_objeto.mainloop()
```

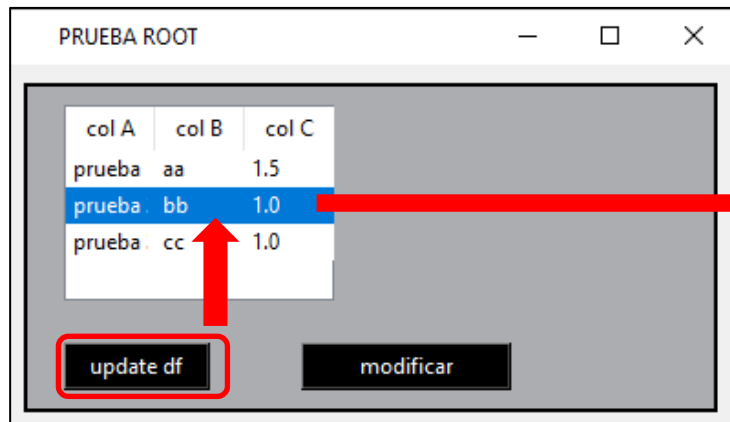
MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

11

EJEMPLO 11 – creación treeview

Se usa la clase `treeview_propio`.

En el ejemplo se crea un treeview y 2 botones, uno para actualizar el treeview desde un dataframe y otro para cambiar agregar una columna al treeview y cambiarle el tipo de selección de ítems de simple a multiple. Se asigna también una rutina que imprime el ítem seleccionado al clicar sobre el mismo.



```
{'lista_columnas_df': ['COLUMNA_1', 'COLUMNA_2', 'COLUMNA_3'],  
'lista_columnas_treeview': ['col A', 'col B', 'col C'],  
'lista_width_columnas_treeview': [50, 50, 50],  
'lista_tipo dato columna df': [dtype('O'), dtype('O'), dtype('float64')],  
'lista datos item seleccionado': [['prueba 2', 'bb', '1.0']]}
```



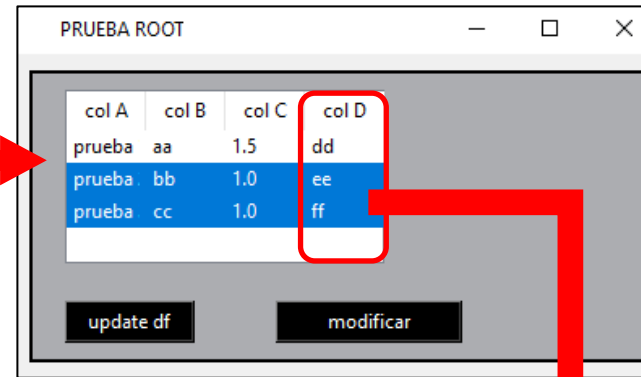
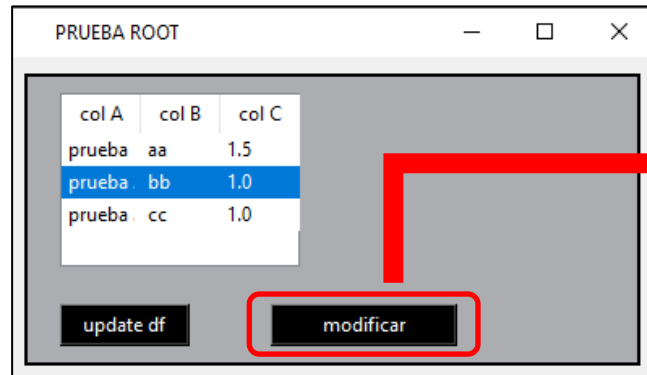
es lista de lista (una sola sublista porque el tipo de selección configurado es *simple*)

```
COLUMNA_1 COLUMNA_2 COLUMNA_3 COLUMNA_4  
0 prueba 1 aa 1.5 dd  
1 prueba 2 bb 1.0 ee  
2 prueba 3 cc 1.0 ff
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

11

EJEMPLO 11 – creación treeview



la selección multiple se hace presionando la tecla Alt Gr + seleccionado items

	COLUMNA_1	COLUMNA_2	COLUMNA_3	COLUMNA_4
0	prueba 1	aa	1.5	dd
1	prueba 2	bb	1.0	ee
2	prueba 3	cc	1.0	ff

```
{'lista_columnas_df': ['COLUMNA_1', 'COLUMNA_2', 'COLUMNA_3', 'COLUMNA_4'],  
'lista_columnas_treeview': ['col A', 'col B', 'col C', 'col D'],  
'lista_width_columnas_treeview': [50, 50, 50, 50],  
'lista_tipo_dato_columna_df': [dtype('O'), dtype('O'), dtype('float64'), dtype('O')],  
'lista_datos_item_seleccionado': [['prueba 2', 'bb', '1.0', 'ee'], ['prueba 3', 'cc', '1.0', 'ff']]}
```



es lista de lista (con 2 sublistas porque el tipo de selección se cambio a **multiple**)

La rutina que se asigna al hacer clic en el item en tkinter debe ser una rutina de evento. La clase **treeview_propio** dispone de un mecanismo interno para transformar la rutina que se configura en el kwargs como rutina de evento por lo que cuando se declara la rutina en la GUI de **mi_proyecto** ya no es necesario agregar event en los parámetros.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

11

EJEMPLO 11 – creación treeview

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, "frozen", False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class Clase_de_mi_proyecto:

    def __init__(self, master):
        self.master = master

        self.df_datos = pd.DataFrame({"COLUMNNA_1": ["prueba 1", "prueba 2", "prueba 3"],
                                      "COLUMNNA_2": ["aa", "bb", "cc"],
                                      "COLUMNNA_3": [1.5, 1, 1],
                                      "COLUMNNA_4": ["dd", "ee", "ff"]})

        self.kwarg_config_frame = {"width": 380,
                                   "height": 180,
                                   "bg": "#ACADB1",
                                   "bd": 2,
                                   "relief": "solid",
                                   "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.kwarg_config_treeview = {"dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 10},
                                      "dicc_treeview": {"seleccion_item": "simple",
                                                         "height": 4,
                                                         "columnas_df": ["COLUMNNA_1", "COLUMNNA_2", "COLUMNNA_3"],
                                                         "columnas_treeview": ["col A", "col B", "col C"],
                                                         "width_columnas_treeview": [50, 50, 50]},
                                      "dicc_rutina_click_item": {"rutina": "def_rutina_click_item"}}

        self.kwarg_config_boton_1 = {"text": "update df",
                                     "width": 10,
                                     "bg": "black",
                                     "fg": "white",
                                     "dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 140},
                                     "dicc_rutina": {"rutina": "def_rutina_boton_1"}}

        self.kwarg_config_boton_2 = {"text": "modificar",
                                     "width": 15,
                                     "bg": "black",
                                     "fg": "white",
                                     "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 140},
                                     "dicc_rutina": {"rutina": "def_rutina_boton_2"}}

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, self.kwarg_config_frame)
        self.treeview = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, self.kwarg_config_treeview)
        self.boton_1 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwarg_config_boton_1)
        self.boton_2 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwarg_config_boton_2)

        self.frame.mainloop()
```

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **self.frame.widget_objeto** (se puede incorporar también en el root)

informar **self**

```
def def_rutina_click_item(self):
    diccionario_datos_item_seleccionado = self.treeview.datos_item_seleccionado
    print(diccionario_datos_item_seleccionado)

def def_rutina_boton_1(self):
    self.treeview.modificaciones("actualizar_desde_df", df_datos = self.df_datos)

def def_rutina_boton_2(self):
    kwarg_config_treeview_update = {"dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 10},
                                    "dicc_treeview": {"seleccion_item": "multiple",
                                                         "height": 4,
                                                         "columnas_df": ["COLUMNNA_1", "COLUMNNA_2", "COLUMNNA_3", "COLUMNNA_4"],
                                                         "columnas_treeview": ["col A", "col B", "col C", "col D"],
                                                         "width_columnas_treeview": [50, 50, 50, 50]},
                                    "dicc_rutina_click_item": {"rutina": "def_rutina_click_item"}}

    self.treeview.modificaciones("modificar_objeto", **kwarg_config_treeview_update)
    self.treeview.config_tributos(**kwarg_config_treeview_update)
    self.treeview.modificaciones("actualizar_desde_df", df_datos = self.df_datos)
```

event no necesario

```
if __name__ == "__main__":
    kwarg_config_root = {"dicc_config_root": {"title": "PRUEBA ROOT",
                                              "iconbitmap": ico_tapar_pluma_tkinter,
                                              "tupla_geometry": (400, 200),
                                              "resizable": (0, 0)}}

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwarg_config_root)
    clase_ventana_inicio(root)
    root.mainloop()
```

Se recuperan los datos del item seleccionado mediante el atributo propio **datos_item_seleccionado**.

Se actualiza el treeview y se le aplica modificaciones después de su creación mediante el método propio **modificaciones** (opción **actualizar_desde_df** y **modificar_objeto** respectivamente).

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

11

EJEMPLO 11 – creación treeview

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys.MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

df_datos = pd.DataFrame({"COLUMNA_1": ["prueba 1", "prueba 2", "prueba 3"]
, "COLUMNA_2": ["aa", "bb", "cc"]
, "COLUMNA_3": [1.5, 1, 1]
, "COLUMNA_4": ["dd", "ee", "ff"]})

kwargs_config_root = {"dicc_config_root":
{
"title": "PRUEBA ROOT"
, "iconbitmap": ico_tapar_pluma_tkinter
, "tuple_geometry": (400, 200)
, "resizable": (1, 1)
}
}

kwargs_config_frame = {"width": 380
, "height": 180
, "bg": "#ACAD81"
, "bd": 2
, "relief": "solid"
, "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_treeview = {"dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 10}
, "dicc_treeview": {"seleccion_item": "simple"
, "height": 4
, "columnas_df": ["COLUMNA_1", "COLUMNA_2", "COLUMNA_3"]
, "columnas_treeview": ["col A", "col B", "col C"]
, "width_columnas_treeview": [50, 50, 50]
}
, "dicc_rutina_click_item": {"rutina": "def_rutina_click_item"}
}

kwargs_config_boton_1 = {"text": "update df"
, "width": 10
, "bg": "black"
, "fg": "white"
, "dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 140}
, "dicc_rutina": {"rutina": "def_rutina_boton_1"}
}

kwargs_config_boton_2 = {"text": "modificar"
, "width": 15
, "bg": "black"
, "fg": "white"
, "dicc_colocacion": {"metodo": "place", "coord_x": 150, "coord_y": 140}
, "dicc_rutina": {"rutina": "def_rutina_boton_2"}
}
```

```
modulo_python_actual = sys.modules[__name__]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root, tipo_widget_param = "frame", **kwargs_config_frame)

def def_rutina_click_item():
diccionario_datos_item_seleccionado = treeview.datos_item_seleccionado
print(diccionario_datos_item_seleccionado)

def def_rutina_boton_1():
treeview.modificaciones("actualizar_desde_df", df_datos = df_datos)

def def_rutina_boton_2():

kwargs_config_treeview_update = {"dicc_colocacion": {"metodo": "place", "coord_x": 20, "coord_y": 10}
, "dicc_treeview": {"seleccion_item": "multiple"
, "height": 4
, "columnas_df": ["COLUMNA_1", "COLUMNA_2", "COLUMNA_3", "COLUMNA_4"]
, "columnas_treeview": ["col A", "col B", "col C", "col D"]
, "width_columnas_treeview": [50, 50, 50, 50]
}
, "dicc_rutina_click_item": {"rutina": "def_rutina_click_item_2"}
}

treeview.modificaciones("modificar_objeto", **kwargs_config_treeview_update)
treeview.config_tributos(**kwargs_config_treeview_update)

treeview.modificaciones("actualizar_desde_df", df_datos = df_datos)

def def_rutina_click_item_2():
diccionario_datos_item_seleccionado = treeview.datos_item_seleccionado
print(diccionario_datos_item_seleccionado)

treeview = mod_utils.treeview_propio(frame.widget_objeto, self.clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_treeview)
boton_1 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self.clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton_1)
boton_2 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self.clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton_2)

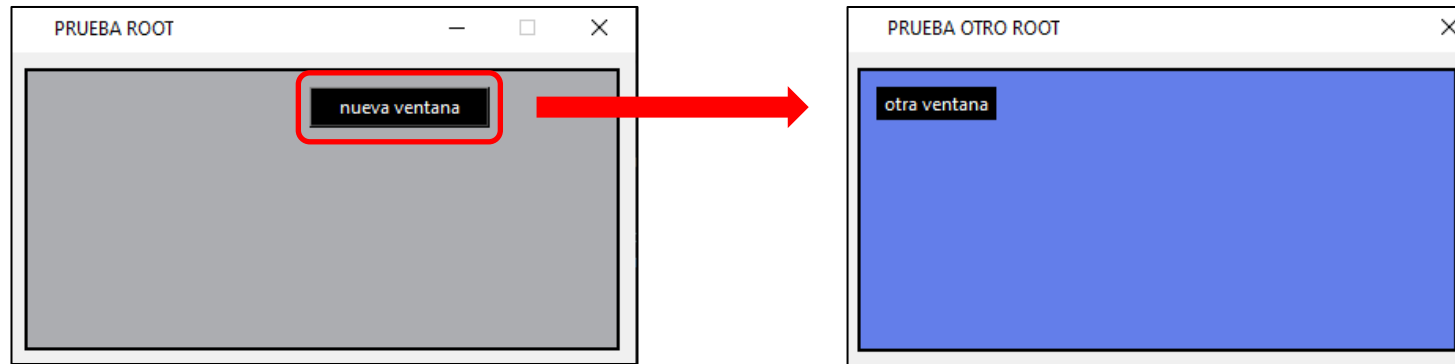
root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

12 EJEMPLO 12 – nuevo root tras pulsar un botón (toplevel)

Se usa la clase `gui_tkinter_widgets`.

En el ejemplo se crea un un root con un frame dentro del cual se encuentra un botón que al pulsarlo abre otra ventana donde se impide volver a la ventana anterior mientras esta 2nda ventana siga abierta.



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

12

EJEMPLO 12 – nuevo root tras pulsar un botón (toplevel)

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_de_mi_proyecto:

    def __init__(self, master):

        self.master = master

        self.kwargs_config_frame = {"width": 380
            , "height": 180
            , "bg": "#ACAD81"
            , "bd": 2
            , "relief": "solid"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwargs_config_boton = {"text": "nueva ventana"
            , "width": 15
            , "bg": "black"
            , "fg": "white"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 180, "coord_y": 10}
            , "dicc_rutina": {"rutina": "def_rutina_boton"}
        }

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)
        self.boton = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_boton)

    def def_rutina_boton(self):

        kwargs_config_root_otra_ventana = {"dicc_config_root":
            {
                "title": "PRUEBA OTRO ROOT"
                , "iconbitmap": ico_tapar_pluma_tkinter
                , "tuple_geometry": (400, 200)
                , "bloquear_interaccion_nueva_ventana_con_otras": True
                , "mantener_nueva_ventana_encima_otras": True
                , "resizable": (0, 0)
            }
        }

        self.otra_ventana = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "toplevel", **kwargs_config_root_otra_ventana["dicc_config_root"])
        self.otra_ventana.config_striducos(**kwargs_config_root_otra_ventana["dicc_config_root"])

        clase_de_mi_proyecto.otra_ventana(self.otra_ventana)
```

permite guardar el foco en la nueva ventana e impide volver a la anterior mientras no se cierre la nueva

informar toplevel

el **master** de la clase **gui_tkinter_widgets** tiene que ser el objeto del frame creado de aquí que salga **self.frame.widget_objeto** (se puede incorporar también en el root)

```
class clase_de_mi_proyecto.otra_ventana:

    def __init__(self, master):

        self.master = master

        self.kwargs_config_frame = {"width": 380
            , "height": 180
            , "bg": "#637EEA"
            , "bd": 2
            , "relief": "solid"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwargs_config_label = {"text": "otra ventana"
            , "width": 10
            , "bg": "black"
            , "fg": "white"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)
        self.label = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "label", **self.kwargs_config_label)

if __name__ == "__main__":

    kwargs_config_root = {"dicc_config_root":
        {
            "title": "PRUEBA ROOT"
            , "iconbitmap": ico_tapar_pluma_tkinter
            , "tuple_geometry": (400, 200)
            , "resizable": (0, 0)
        }
    }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_de_mi_proyecto(root)
    root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

12

EJEMPLO 12 – nuevo root tras pulsar un botón (toplevel)

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
    {"title": "PRUEBA ROOT"
    , "iconbitmap": ico_tapar_pluma_tkinter
    , "tupla_geometry": (400, 200)
    , "resizable": (0, 0)
    }
}

kwargs_config_frame = {"width": 380
    , "height": 180
    , "bg": "#ACADB1"
    , "bd": 2
    , "relief": "solid"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_boton = {"text": "nueva ventana"
    , "width": 15
    , "bg": "black"
    , "fg": "white"
    , "dicc_colocacion": {"metodo": "place", "coord_x": 180, "coord_y": 10}
    , "dicc_rutina": {"rutina": "def_rutina_boton"}
}

modulo_python_actual = sys.modules[__name__]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
```

```
def def_rutina_boton():

    kwargs_config_root_otra_ventana = {"dicc_config_root":
        {"title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
        , "bloquear_interaccion_nueva_ventana_con_otras": True
        , "mantener_nueva_ventana_encima_otras": True
        , "resizable": (0, 0)
        }
    }

    otra_ventana = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "toplevel", **kwargs_config_root_otra_ventana["dicc_config_root"])
    otra_ventana.config_tributos(**kwargs_config_root_otra_ventana["dicc_config_root"])

    kwargs_config_frame = {"width": 380
        , "height": 180
        , "bg": "#637EEA"
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    }

    kwargs_config_label = {"text": "otra ventana"
        , "width": 10
        , "bg": "black"
        , "fg": "white"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    }

    frame = mod_utils.gui_tkinter_widgets(otra_ventana.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
    label = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "label", **kwargs_config_label)

    boton = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_boton)

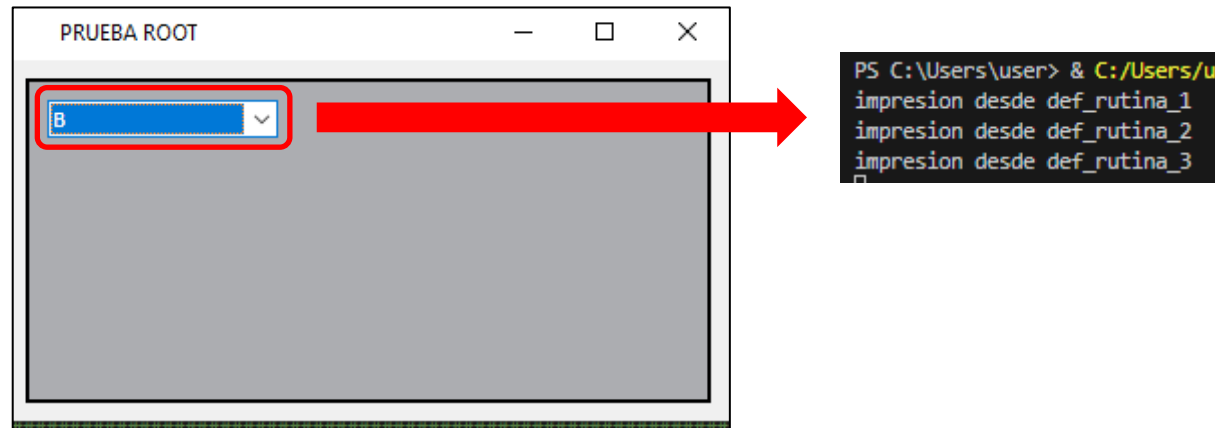
    root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

13 EJEMPLO 13 – Aplicar a un widget varias rutinas de evento bind (ejemplo combobox)

Se usa la clase `gui_tkinter_widgets`.

En el ejemplo, se crea un combobox al cual se agregan 3 rutinas de eventos bind (`<<ComboboxSelected>>`), cada rutina imprime algo distinto cada vez que se selecciona una opción en el combobox.



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

13

EJEMPLO 13 – Aplicar a un widget varias rutinas de evento bind (ejemplo combobox)

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys_MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class ClaseDeMiProyecto:

    def __init__(self, master):

        self.master = master

        self.kwarg_config_frame = {"width": 300
        , "height": 180
        , "bg": "#ACAD81"
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwarg_config_combobox = {"font": ("Calibri", 10)
        , "width": 15
        , "justify": tk.LEFT
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        , "combobox_list_options": ["A", "B", "C"]
        , "lista_dicc_rutina_aplicar_eventos_widget": [{"tipo_bind": "<<ComboboxSelected>>", "rutina": "def_rutina_1"}
        , {"tipo_bind": "<<ComboboxSelected>>", "rutina": "def_rutina_2"}
        , {"tipo_bind": "<<ComboboxSelected>>", "rutina": "def_rutina_3"}
        ]
        }

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwarg_config_frame)
        self.combobox = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "combobox", self_clase_gui_donde_call_rutina = self, **self.kwarg_config_combobox)

    def def_rutina_1(self):
        print("impresion desde def_rutina_1")

    def def_rutina_2(self):
        print("impresion desde def_rutina_2")

    def def_rutina_3(self):
        print("impresion desde def_rutina_3")

if __name__ == "__main__":

    kwarg_config_root = {"dicc_config_root":
        {"title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 200)
        , "resizable": (1, 1)
        }
        }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwarg_config_root)
    clase_de_mi_proyecto(root)
    root.widget_objeto.mainloop()
```

Al asignar rutinas de evento al combobox, se tiene que declarar **self** en el parámetro **self_clase_donde_call_rutina**

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

13

EJEMPLO 13 – Aplicar a un widget varias rutinas de evento bind (ejemplo combobox)

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                            if getattr(sys, 'frozen', False)
                            else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

kwargs_config_root = {"dicc_config_root":
                      {"title": "PRUEBA ROOT"
                       , "iconbitmap": ico_tapar_pluma_tkinter
                       , "tupla_geometry": (400, 200)
                       , "resizable": (1, 1)
                      }}

kwargs_config_frame = {"width": 380
                       , "height": 180
                       , "bg": "#ACADBD"
                       , "bd": 2
                       , "relief": "solid"
                       , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                       }

kwargs_config_combobox = {"font": ("Calibri", 10)
                          , "width": 15
                          , "justify": tk.LEFT
                          , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                          , "combobox_lista_opciones": ["A", "B", "C"]
                          , "lista_dicc_rutina_aplicar_eventos_widget": [{"tipo_bind": "<<ComboboxSelected>>", "rutina": "def_rutina_1"}
                                                                    , {"tipo_bind": "<<ComboboxSelected>>", "rutina": "def_rutina_2"}
                                                                    , {"tipo_bind": "<<ComboboxSelected>>", "rutina": "def_rutina_3"}
                          ]
                          }

modulo_python_actual = sys.modules[__name__]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)

def def_rutina_1():
    print("impresion desde def_rutina_1")

def def_rutina_2():
    print("impresion desde def_rutina_2")

def def_rutina_3():
    print("impresion desde def_rutina_3")

combobox = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "combobox", self_clase_gui_donde_call_rutina = modulo_python_actual, **kwargs_config_combobox)

root.widget_objeto.mainloop()
```

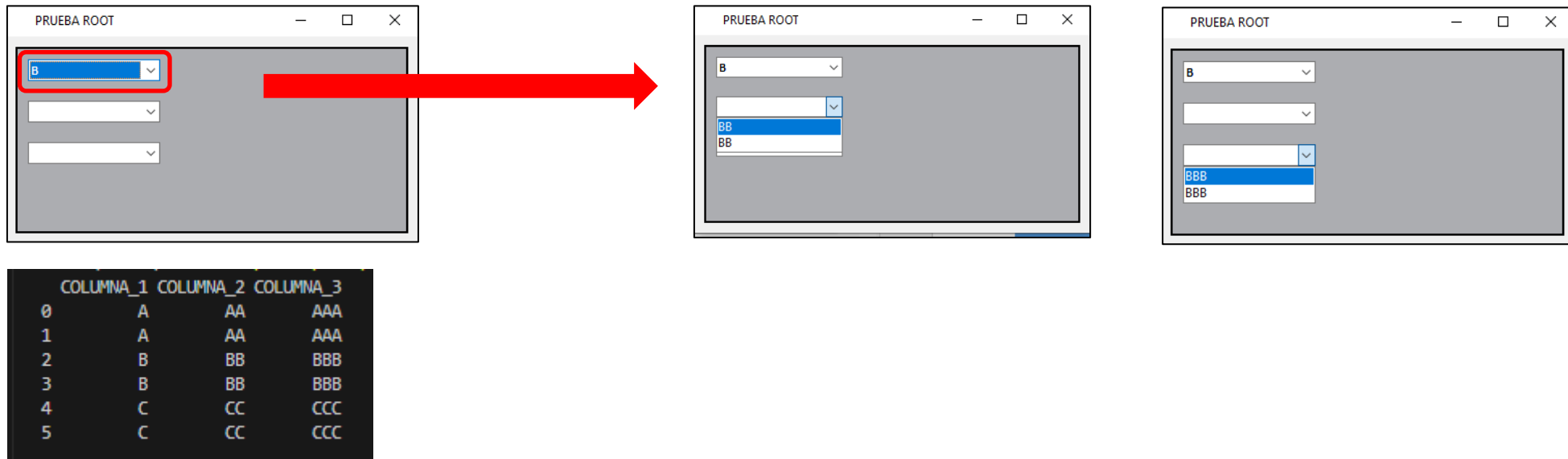
MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

14

EJEMPLO 14 – Aplicar a una variable enlace (stringvar) varias rutinas de evento en modo trace (ejemplo combobox)

Se usa la clase `gui_tkinter_widgets`.

En el ejemplo, se crean 3 combobox, el 1ero con las opciones A, B y C y los 2 otros sin opciones por escoger de inicio. En función de la opción seleccionada en el 1er combobox se actualizan las opciones de los 2 otros combobox en base a datos provenientes de un dataframe.



The diagram illustrates the dynamic update of combobox options based on a DataFrame. It shows three states of a Tkinter window titled "PRUEBA ROOT" with three comboboxes. A red arrow points from the first state to the second, and another from the second to the third.

State 1 (Left): The first combobox is selected with "B". The second and third comboboxes are empty.

State 2 (Middle): The first combobox is selected with "B". The second combobox is selected with "BB". The third combobox is empty.

State 3 (Right): The first combobox is selected with "B". The second combobox is selected with "BBB". The third combobox is empty.

DataFrame:

	COLUMNA_1	COLUMNA_2	COLUMNA_3
0	A	AA	AAA
1	A	AA	AAA
2	B	BB	BBB
3	B	BB	BBB
4	C	CC	CCC
5	C	CC	CCC

Lo que se obtiene aquí es lo mismo que hace el **método nativo `trace_add` de tkinter**. Las rutinas que se declaran con este método y se asocian a un stringvar (o intvar) tienen que **tener un parámetro `*args`**. La clase `gui_tkinter_widgets` tiene un mecanismo interno que integra este tipo de parámetros directamente cuando se asocia una rutina a una variable de enlace por lo que **ya no es necesario informar el `*args`**.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

14

EJEMPLO 14 – Aplicar a una variable enlace (stringvar) varias rutinas de evento en modo trace (ejemplo combobox)

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class Clase_de_mi_proyecto:

    def __init__(self, master):

        self.master = master

        self.df_datos = pd.DataFrame({"COLUMNA_1": ["A", "A", "B", "B", "C", "C"],
                                      "COLUMNA_2": ["AA", "AA", "BB", "BB", "CC", "CC"],
                                      "COLUMNA_3": ["AAA", "AAA", "BBB", "BBB", "CCC", "CCC"]
                                      })

        self.kwargs_config_frame = {"width": 380,
                                    "height": 180,
                                    "bg": "#ACADB1",
                                    "bd": 2,
                                    "relief": "solid",
                                    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}}

        self.kwargs_config_combobox_1 = {"font": ("Calibri", 10),
                                          "width": 15,
                                          "justify": tk.LEFT,
                                          "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10},
                                          "lista_dicc_rutina_trace_variable_enlace": [{"tipo_trace": "write", "rutina": "def_rutina_update_combobox_2"},
                                          {"tipo_trace": "write", "rutina": "def_rutina_update_combobox_3"}]}

        self.kwargs_config_combobox_2 = {"font": ("Calibri", 10),
                                          "width": 15,
                                          "justify": tk.LEFT,
                                          "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 50},
                                          "combobox_lista_opciones": []}

        self.kwargs_config_combobox_3 = {"font": ("Calibri", 10),
                                          "width": 15,
                                          "justify": tk.LEFT,
                                          "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 90},
                                          "combobox_lista_opciones": []}

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)
        self.combobox_1 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "combobox", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_combobox_1)
        self.combobox_2 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "combobox", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_combobox_2)
        self.combobox_3 = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "combobox", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_combobox_3)
```

```
def def_rutina_update_combobox_2(self):
    valor_combobox_1 = self.combobox_1.variable_enlace.get()

    df_datos_filtrado = self.df_datos.loc[self.df_datos["COLUMNA_1"] == valor_combobox_1, ["COLUMNA_2"]]
    lista_opciones = df_datos_filtrado.values.tolist()
    lista_opciones = [item[0] for item in lista_opciones]

    self.combobox_2.config_tributos(**{"combobox_lista_opciones": lista_opciones})

def def_rutina_update_combobox_3(self):
    valor_combobox_1 = self.combobox_1.variable_enlace.get()

    df_datos_filtrado = self.df_datos.loc[self.df_datos["COLUMNA_1"] == valor_combobox_1, ["COLUMNA_3"]]
    lista_opciones = df_datos_filtrado.values.tolist()
    lista_opciones = [item[0] for item in lista_opciones]

    self.combobox_3.config_tributos(**{"combobox_lista_opciones": lista_opciones})

if __name__ == "__main__":

    kwargs_config_root = {"dicc_config_root":
                        {"title": "PRUEBA ROOT",
                        "iconbitmap": ico_tapar_pluma_tkinter,
                        "tupla_geometry": (400, 200),
                        "resizable": (1, 1)}
                        }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_de_mi_proyecto(root)
    root.widget_objeto.mainloop()
```

Al asignar rutinas a la variable de enlace (stringvar) al combobox 1, se tiene que declarar **self** en el parámetro **self_clase_donde_call_rutina**

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

14

EJEMPLO 14 – Aplicar a una variable enlace (stringvar) varias rutinas de evento en modo trace (ejemplo combobox)

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

df_datos = pd.DataFrame({"COLUMNA_1": ["A", "A", "B", "B", "C", "C"],
    "COLUMNA_2": ["AA", "AA", "BB", "BB", "CC", "CC"],
    "COLUMNA_3": ["AAA", "AAA", "BBB", "BBB", "CCC", "CCC"]
})

kwargs_config_root = {"dicc_config_root":
    {"title": "PRUEBA ROOT",
    "iconbitmap": ico_tapar_pluma_tkinter,
    "tuple_geometry": (400, 200),
    "resizable": (1, 1)}
}

kwargs_config_frame = {"width": 380,
    "height": 180,
    "bg": "#ACAD01",
    "bd": 2,
    "relief": "solid",
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_combobox_1 = {"font": ("Calibri", 10),
    "width": 15,
    "justify": tk.LEFT,
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10},
    "lista_dicc_rutina_trace_variable_enlace": [{"tipo_trace": "write", "rutina": "def_rutina_update_combobox_2"},
    {"tipo_trace": "write", "rutina": "def_rutina_update_combobox_3"}
]

kwargs_config_combobox_2 = {"font": ("Calibri", 10),
    "width": 15,
    "justify": tk.LEFT,
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 50},
    "combobox_lista_opciones": []
}

kwargs_config_combobox_3 = {"font": ("Calibri", 10),
    "width": 15,
    "justify": tk.LEFT,
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 90},
    "combobox_lista_opciones": []
}
```

```
modulo_python_actual = sys.modules[ _name_ ]

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)

def def_rutina_update_combobox_2():
    valor_combobox_1 = combobox_1.variable_enlace.get()

    df_datos_filtrado = df_datos.loc[df_datos["COLUMNA_1"] == valor_combobox_1, ["COLUMNA_2"]]
    lista_opciones = df_datos_filtrado.values.tolist()
    lista_opciones = [item[0] for item in lista_opciones]

    combobox_2.config_tributos(**{"combobox_lista_opciones": lista_opciones})

def def_rutina_update_combobox_3():
    valor_combobox_1 = combobox_1.variable_enlace.get()

    df_datos_filtrado = df_datos.loc[df_datos["COLUMNA_1"] == valor_combobox_1, ["COLUMNA_3"]]
    lista_opciones = df_datos_filtrado.values.tolist()
    lista_opciones = [item[0] for item in lista_opciones]

    combobox_3.config_tributos(**{"combobox_lista_opciones": lista_opciones})

combobox_1 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "combobox",
    combobox_2 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "combobox", **kwargs_config_combobox_2)
    combobox_3 = mod_utils.gui_tkinter_widgets(frame.widget_objeto, tipo_widget_param = "combobox", **kwargs_config_combobox_3)

root.widget_objeto.mainloop()
```

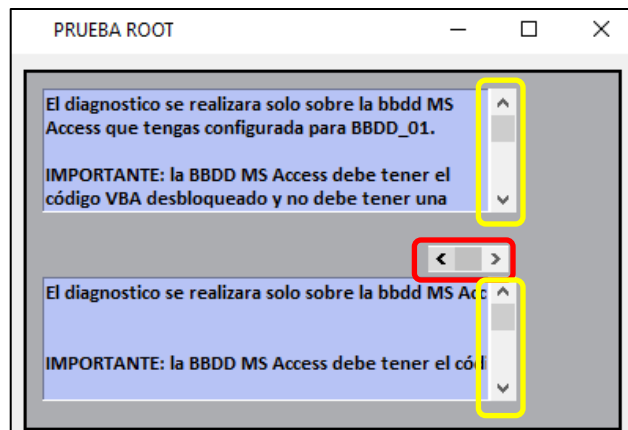
MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

15 EJEMPLO 15 – scrolledtext desde un string y desde un dataframe

Se usa la clase `scrolledtext_propio`.

En el ejemplo, se crean 2 scrolledtext:

- el 1ero se informa desde un string y se le aplica el atributo nativo **wrap** a WORD (corte por palabras de tal forma que cada línea del scrolledtext no supere nunca el width del mismo).
- el 2do se informa desde un dataframe y se le aplica el atributo nativo **wrap** a NONE (las líneas se establecen según los registros del dataframe). En este caso, el texto en muchas líneas excede el width del scrolledtext por lo que se le configura el atributo propio **colocacion_scrollbar_horizontal** para colocar una barra horizontal de scrolling.



En ambos casos, según el número final de líneas del scrolledtext informado si este número excede el atributo nativo height la clase `scrolledtext_propio` tiene un mecanismo interno para colocar automáticamente una barra de scrolling vertical dentro del scrolledtext.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

15 EJEMPLO 15 – scrolledtext desde un string y desde un dataframe

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class Clase_de_mi_proyecto:

    def __init__(self, master):

        self.master = master

        self.lista_texto = [
            "El diagnostico se realizara solo sobre la bbdd MS Access que tengas configurada para BBDD_01.\n\n",
            "¡IMPORTANTE! la BBDD MS Access debe tener el código VBA desbloqueado y no debe tener una macro AutoExec activada.\n\nAl finalizar el proceso, se generara un excel donde:\n\n",
            "LISTADO: se listan todos los objetos y se aportan diversas informaciones en función del tipo de objeto.\n\n",
            "DEPENDENCIAS: se listan para cada objeto en que modulos/rutinas VBA se usan.\n\nSIN DEPENDENCIAS: se listan que objetos no se usan modulos/rutinas VBA.\n\n",
            "TABLAS (CHECK MANUAL): se listan todas las tablas con las rutinas / funciones VBA en las que se usan como string encapsuladas entre comillas dobles",
            "pero que no parecen sentencias SQL o de manipulación de tablas via VBA. Sera el usuario quien ha de decidir si las tablas de este listado se usan o no en código VBA."
        ]

        self.string_scrolledtext = "".join(self.lista_texto)
        self.df_scrolledtext = pd.DataFrame([item for item in self.lista_texto], columns = ["TEXT0"])

        self.kwarg_config_frame = {"width": 380
            , "height": 230
            , "bg": "#ACAD81"
            , "bd": 2
            , "relief": "solid"
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwarg_config_scrolledtext_1 = {"font": ("Calibri", 10, "bold")
            , "width": 40
            , "height": 5
            , "state": tk.NORMAL
            , "bg": "#B7C3F5"
            , "wrap": tk.WORD
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
            , "justify": tk.LEFT
        }

        self.kwarg_config_scrolledtext_2 = {"font": ("Calibri", 10, "bold")
            , "width": 40
            , "height": 5
            , "state": tk.NORMAL
            , "bg": "#B7C3F5"
            , "wrap": tk.NONE
            , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 130}
            , "justify": tk.LEFT
            , "colocacion_scrollbar_horizontal": {"metodo": "place", "coord_x": 255, "coord_y": 110}
        }

        self.frame = mod_utils_mi_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwarg_config_frame)

        self.scrolledtext_1 = mod_utils_scrolledtext_propio(self.frame.widget_objeto, **self.kwarg_config_scrolledtext_1)
        self.scrolledtext_2 = mod_utils_scrolledtext_propio(self.frame.widget_objeto, **self.kwarg_config_scrolledtext_2)
```

```
self.scrolledtext_1.modificaciones("agregar_solo_contenido_desde_string"
    , string_texto_informar = self.string_scrolledtext
    , height_scrolledtext = self.kwarg_config_scrolledtext_1.get("height", 1))

self.scrolledtext_2.modificaciones("agregar_solo_contenido_desde_dataframe"
    , df_datos = self.df_scrolledtext
    , columna_df_para_informar = "TEXT0"
    , height_scrolledtext = self.kwarg_config_scrolledtext_2.get("height", 1))

if __name__ == "__main__":

    kwarg_config_root = {"dicc_config_root":
        {"title": "PRUEBA ROOT"
        , "iconbitmap": "ico_tapar_pluma_tkinter"
        , "tupla_geometry": (400, 250)
        , "resizable": (1, 1)
        }
    }

    root = mod_utils_mi_tkinter_widgets(None, tipo_widget_param = "root", **kwarg_config_root)

    clase_de_mi_proyecto(root)

    root.widget_objeto.mainloop()
```

La clase **scrolledtext_propio** dispone del método propio **modificaciones** para agregar contenido desde un string o desde un dataframe.

Según sea uno u otro cambian tanto el args como los kwargs necesarios.

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

15

EJEMPLO 15 – scrolledtext desde un string y desde un dataframe

CASO 2. Directamente en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
    if getattr(sys, 'frozen', False)
    else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

lista_texto = [
    "El diagnostico se realizara solo sobre la bbdd MS Access que tengas configurada para BDD081.\n\n",
    "IMPORTANTE: la BDD MS Access debe tener el código VBA desbloqueado y no debe tener una macro AutoExec activada.\n\nAl finalizar el proceso, se generara un excel donde:\n\n",
    "LISTADO: se listan todos los objetos y se aportan diversas informaciones en función del tipo de objeto.\n\n",
    "DEPENDENCIAS: se listan para cada objeto en que modulos/rutinas VBA se usan.\n\nSIN DEPENDENCIAS: se listan que objetos no se usan modulos/rutinas VBA.\n\n",
    "TABLAS (CHECK MANUAL): se listan todas las tablas con las rutinas / funciones VBA en las que se usan como string encapsuladas entre comillas dobles",
    "pero que no parecen sentencias SQL o de manipulación de tablas via VBA. Sera el usuario quien ha de decidir si las tablas de este listado se usan o no en código VBA."
]

string_scrolledtext = "".join(lista_texto)
df_scrolledtext = pd.DataFrame([item for item in lista_texto], columns = ["TEXT0"])

kwargs_config_root = {"dicc_config_root":
    {
        "title": "PRUEBA ROOT",
        "iconbitmap": ico_tapar_pluma_tkinter,
        "tuple_geometry": (400, 250),
        "resizable": (1, 1)
    }
}

kwargs_config_frame = {"width": 300,
    "height": 230,
    "bg": "#ACADDB",
    "bd": 2,
    "relief": "solid",
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
}

kwargs_config_scrolledtext_1 = {"font": ("Calibri", 10, "bold"),
    "width": 40,
    "height": 5,
    "state": tk.NORMAL,
    "bg": "#B7C3F5",
    "fg": "black",
    "wrap": tk.WORD,
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10},
    "justify": tk.LEFT
}

kwargs_config_scrolledtext_2 = {"font": ("Calibri", 10, "bold"),
    "width": 40,
    "height": 5,
    "state": tk.NORMAL,
    "bg": "#B7C3F5",
    "fg": "black",
    "wrap": tk.NONE,
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 130},
    "justify": tk.LEFT,
    "colocacion_scrollbar_horizontal": {"metodo": "place", "coord_x": 255, "coord_y": 110}
}

root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)
scrolledtext_1 = mod_utils.scrolledtext_propio(frame.widget_objeto, **kwargs_config_scrolledtext_1)
scrolledtext_2 = mod_utils.scrolledtext_propio(frame.widget_objeto, **kwargs_config_scrolledtext_2)
```

```
scrolledtext_1.modificaciones("agregar_solo_contenido_desde_string",
    , string_texto_informar = string_scrolledtext
    , height_scrolledtext = kwargs_config_scrolledtext_1.get("height", 1))

scrolledtext_2.modificaciones("agregar_solo_contenido_desde_dataframe",
    , df_datos = df_scrolledtext
    , column_df_para_informar = "TEXT0"
    , height_scrolledtext = kwargs_config_scrolledtext_2.get("height", 1))

root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

Se usa la clase `scrolledtext_propio`.

Al igual que en el caso `agregar_solo_contenido_desde_dataframe` (ejemplo 15), según el número final de líneas del `scrolledtext` informado si este número excede el atributo nativo `height` la clase `scrolledtext_propio` tiene un mecanismo interno para colocar automáticamente una barra de scrolling vertical dentro del `scrolledtext`.

Antes de enunciar el ejemplo, se comentan los atributos propios relacionados con los tags a aplicar (poner de color líneas enteras o fragmentos de caracteres de las mismas en colores según criterios configurables).y como se configuran.

ATRIBUTO PROPIO	DESCRIPCIÓN	KEYS DICCIONARIOS	DESCRIPCIÓN KEY	KEY OBLIGATORIA	TIPO DATO KEY
lista_dicc_tag_linea_completa	Lista de diccionarios donde cada diccionario contiene las keys de la columna siguiente. Sirve para aplicar tags sobre la línea completa de los registros de una columna del dataframe que sirve para informar el <code>scrolledtext</code> usando como criterio el valor tomado por la misma columna u otra del dataframe	nombre_tag	Nombre que se le quiere aplicar al tag (es interno al funcionamiento de la clase). Los distintos diccionarios que componen <code>lista_dicc_tag_linea_completa</code> pueden tener las keys <code>nombre_tag</code> <u>duplicadas</u> lo que permite <u>aplicar varios tags a un mismo criterio</u> (por ejemplo el <code>background</code> y luego el <code>foreground</code>)	Si	string o número
		columna_df_tag_aplicar	Es la columna del dataframe donde aplicar el tag	Si	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio <code>df_datos</code>)
		case_sensitive	Permite al aplicar el tag sobre la columna <code>columna_df_tag_aplicar</code> del dataframe discriminar o no por mayúsculas o minúsculas	No	True o False
		dicc_config	Es un diccionario con los <u>atributos nativos tkinter asociados a los <code>scrolledtext</code></u> . Se recomienda configurar un <u>atributo nativo a la vez</u> (por ejemplo <code>background</code> en un diccionario de <code>lista_dicc_tag_linea_completa</code> y luego el <code>foreground</code> en otro diccionario donde el valor de <code>nombre_tag</code> sea el mismo que en el anterior diccionario)	Si	diccionario de atributos nativos

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

ATRIBUTO PROPIO	DESCRIPCIÓN	KEYS DICCIONARIOS	DESCRIPCIÓN KEY	KEY OBLIGATORIA	TIPO DATO KEY
lista_dicc_tag_caracteres_cambiantes_comparativa	<p>Lista de diccionarios donde cada diccionario contiene las keys de la columna siguiente.</p> <p>Sirve para complementar los tags de lista_dicc_tag_linea_completa aplicando tags por fragmentos de texto comparando 2 columnas del dataframe que sirve para informar el scrolledtext (tipo de comparativa antes y despues).</p>	nombre_tag	Nombre que se le quiere aplicar al tag (es interno al funcionamiento de la clase). Los distintos diccionarios que componen lista_dicc_tag_caracteres_cambiantes_comparativa pueden tener las keys nombre_tag duplicadas lo que permite aplicar varios tags a un mismo criterio (por ejemplo el background y luego el foreground)	Si	string o número
		columna_df_filtro_registros_aplicar_tag	Es la columna en la cual se desea filtrar los registros que cumplan el valor configurado en columna_df_filtro_registros_aplicar_tag_valor para aplicarles el tag	Si	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
		columna_df_filtro_registros_aplicar_tag_valor	Es el valor en la columna columna_df_filtro_registros_aplicar_tag que sirve para filtrar los registros donde aplicarles el tag	Si	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
		columna_df_comparar_1	Es la columna del dataframe que sirve para comparar los registros con los de la columna del mismo dataframe informados en la columna configurada en columna_df_comparar_2	Si	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
		columna_df_comparar_2	Es la columna del dataframe que sirve para comparar los registros con los de la columna del mismo dataframe informados en la columna configurada en columna_df_comparar_1	Si	string o número (la columna ha de estar definida en el dataframe que se asigna al atributo propio df_datos)
		case_sensitive	Permite al aplicar el tag sobre la columna columna_df_tag_aplicar del dataframe discriminar o no por mayúsculas o minúsculas	No	True o False
		marcar_toda_linea_si_todo_varia	Permite marcar toda la línea si todo el registro varia en comparativa con el registro de la columna (esto es para evitar sobrecarga de colores en combinación con los tags configurados en lista_dicc_tag_linea_completa)	No	True o False
		dicc_config	Es un diccionario con los atributos nativos tkinter asociados a los scrolledtext. Se recomienda configurar un atributo nativo a la vez (por ejemplo 1ero el background en un diccionario de lista_dicc_tag_caracteres_cambiantes_comparativa y luego el foreground en otro diccionario donde el valor de nombre_tag sea el mismo que en el anterior diccionario)	Si	diccionario de atributos nativos

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

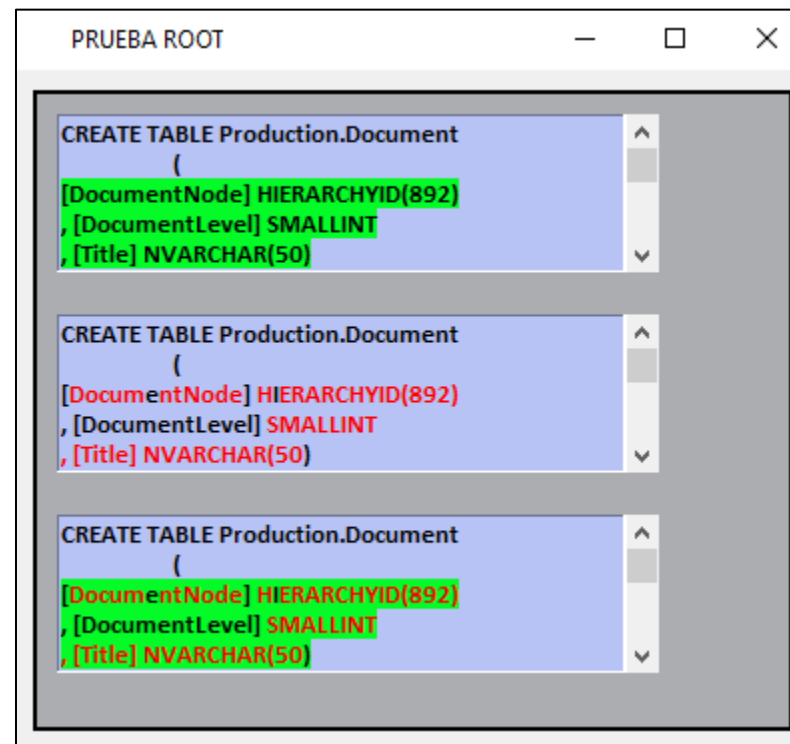
16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

En el ejemplo, se crean 3 scrolledtext que se informan desde un mismo dataframe:

	CODIGO	CODIGO_OTRA_BBDD	CONTROL_CAMBIOS_ACTUAL
0	CREATE TABLE Production.Document	CREATE TABLE Production.Document	None
1	\t(\	\t(\	None
2	[DocumentNode] HIERARCHYID(892)	[Owner] INT	CAMBIOS_LOCALIZADOS
3	, [DocumentLevel] SMALLINT	, [DocumentLevel] smallint	CAMBIOS_LOCALIZADOS
4	, [Title] NVARCHAR(50))	CAMBIOS_LOCALIZADOS
5	, [Owner] INT	None	CAMBIOS_LOCALIZADOS
6	, [FolderFlag] BIT	None	CAMBIOS_LOCALIZADOS
7	, [FileName] NVARCHAR(400)	None	CAMBIOS_LOCALIZADOS
8	, [FileExtension] NVARCHAR(8)	None	CAMBIOS_LOCALIZADOS
9	, [Revision] NCHAR(5)	None	CAMBIOS_LOCALIZADOS
10	, [ChangeNumber] INT	None	None
11	, [Status] TINYINT	None	CAMBIOS_LOCALIZADOS
12	, [DocumentSummary] NVARCHAR(-1)	None	CAMBIOS_LOCALIZADOS
13	, [Document] VARBINARY(-1)	None	CAMBIOS_LOCALIZADOS
14	, [rowguid] UNIQUEIDENTIFIER	None	CAMBIOS_LOCALIZADOS
15	, [ModifiedDate] DATETIME	None	CAMBIOS_LOCALIZADOS
16)	None	None

WIDGET	DESCRIPCIÓN
scrolledtext 1	se configura un solo tag con <code>lista_dicc_tag_linea_completa</code> : poner el background en verde en la columna CODIGO si en en la columna CONTROL_CAMBIOS_ACTUAL aparece el valor CAMBIOS_LOCALIZADOS. Se configura <code>case_sensitive</code> a <code>False</code> .
scrolledtext 2	se configura un solo tag con <code>lista_dicc_tag_caracteres_cambiantes_comparativa</code> : poner el foreground en rojo en la columna CODIGO comparando la columna CODIGO con la de CODIGO_OTRA_BBDD si en en la columna CONTROL_CAMBIOS_ACTUAL aparece el valor CAMBIOS_LOCALIZADOS. Se configura <code>case_sensitive</code> a <code>True</code> .
scrolledtext 3	se configura la combinación de las 2 anteriores. Se configura <code>case_sensitive</code> a <code>True</code> en <code>lista_dicc_tag_linea_completa</code> y en <code>lista_dicc_tag_caracteres_cambiantes_comparativa</code> .



MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
if getattr(sys, 'frozen', False)
else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class ClaseDeMiProyecto:

    def __init__(self, master):

        self.master = master

        self.lista_texto_para_df = [['CREATE TABLE Production.Document', 'CREATE TABLE Production.Document', None]
        , ['t', 't', None], [['DocumentNode] HIERARCHYID(892)', '[Owner] INT', 'CAMBIOS_LOCALIZADOS']
        , ['[DocumentLevel] SMALLINT', '[DocumentLevel] smallint', 'CAMBIOS_LOCALIZADOS']
        , ['[Title] NVARCHAR(50)', '[Title] NVARCHAR(50)', 'CAMBIOS_LOCALIZADOS']
        , ['[Owner] INT', None, 'CAMBIOS_LOCALIZADOS']
        , ['[FolderFlag] BIT', None, 'CAMBIOS_LOCALIZADOS']
        , ['[FileName] NVARCHAR(400)', None, 'CAMBIOS_LOCALIZADOS']
        , ['[FileExtension] NVARCHAR(8)', None, 'CAMBIOS_LOCALIZADOS']
        , ['[Revision] NCHAR(5)', None, 'CAMBIOS_LOCALIZADOS']
        , ['[ChangeNumber] INT', None, None]
        , ['[Status] TINYINT', None, 'CAMBIOS_LOCALIZADOS']
        , ['[DocumentSummary] NVARCHAR(-1)', None, 'CAMBIOS_LOCALIZADOS']
        , ['[Document] VARBINARY(-1)', None, 'CAMBIOS_LOCALIZADOS']
        , ['[rowguid] UNIQUEIDENTIFIER', None, 'CAMBIOS_LOCALIZADOS']
        , ['[ModifiedDate] DATETIME', None, 'CAMBIOS_LOCALIZADOS']
        , [''], None, None]]

        self.df_scrolledtext = pd.DataFrame(self.lista_texto_para_df, columns = ["CODIGO", "CODIGO_OTRA_BBDD", "CONTROL_CAMBIOS_ACTUAL"])

        self.kwargs_config_frame = {"width": 300
        , "height": 320
        , "bg": "#ACAD81"
        , "bd": 2
        , "relief": "solid"
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        }

        self.kwargs_config_scrolledtext_1 = {"font": ("Calibri", 10, "bold")
        , "width": 40
        , "height": 5
        , "state": tk.NORMAL
        , "bg": "#87C3F5"
        , "fg": "black"
        , "wrap": tk.NONE
        , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
        , "justif": tk.LEFT

        self.lista_dicc_tag_linea_completa = [{"nombre_tag": "CAMBIOS_LOCALIZADOS"
        , "columna_df_tag_aplicar": "CONTROL_CAMBIOS_ACTUAL"
        , "case_sensitive": False
        , "dicc_config": {"background": "#85FB27"}
        }
        ]
    }
```

```
self.kwargs_config_scrolledtext_2 = {"font": ("Calibri", 10, "bold")
, "width": 40
, "height": 5
, "state": tk.NONE
, "bg": "#87C3F5"
, "fg": "black"
, "wrap": tk.NONE
, "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 110}
, "justif": tk.LEFT

self.lista_dicc_tag_caracteres_cambiantes_comparativa = [{"nombre_tag": "CAMBIOS_LOCALIZADOS_POR_INDICES"
, "columna_df_filtro_registros_aplicar_tag": "CONTROL_CAMBIOS_ACTUAL"
, "columna_df_filtro_registros_aplicar_tag_valor": "CAMBIOS_LOCALIZADOS"
, "columna_df_comparar_1": "CODIGO"
, "columna_df_comparar_2": "CODIGO_OTRA_BBDD"
, "case_sensitive": True
, "marcar_toda_linea_si_todo_varia": False
, "dicc_config": {"foreground": "red"}
}
]

self.kwargs_config_scrolledtext_3 = {"font": ("Calibri", 10, "bold")
, "width": 40
, "height": 5
, "state": tk.NORMAL
, "bg": "#87C3F5"
, "fg": "black"
, "wrap": tk.NONE
, "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 210}
, "justif": tk.LEFT

self.lista_dicc_tag_linea_completa = [{"nombre_tag": "CAMBIOS_LOCALIZADOS"
, "columna_df_tag_aplicar": "CONTROL_CAMBIOS_ACTUAL"
, "case_sensitive": True
, "dicc_config": {"background": "#85FB27"}
}
]

self.lista_dicc_tag_caracteres_cambiantes_comparativa = [{"nombre_tag": "CAMBIOS_LOCALIZADOS_POR_INDICES"
, "columna_df_filtro_registros_aplicar_tag": "CONTROL_CAMBIOS_ACTUAL"
, "columna_df_filtro_registros_aplicar_tag_valor": "CAMBIOS_LOCALIZADOS"
, "columna_df_comparar_1": "CODIGO"
, "columna_df_comparar_2": "CODIGO_OTRA_BBDD"
, "case_sensitive": True
, "marcar_toda_linea_si_todo_varia": False
, "dicc_config": {"foreground": "red"}
}
]
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

CASO 1. Desde una clase propia en el módulo de *mi_proyecto*

```
self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)

self.scrolledtext_1 = mod_utils.scrolledtext_propio(self.frame.widget_objeto, **self.kwargs_config_scrolledtext_1)
self.scrolledtext_2 = mod_utils.scrolledtext_propio(self.frame.widget_objeto, **self.kwargs_config_scrolledtext_2)
self.scrolledtext_3 = mod_utils.scrolledtext_propio(self.frame.widget_objeto, **self.kwargs_config_scrolledtext_3)

self.scrolledtext_1.modificaciones("agregar_contenido_y_tags_desde_dataframe"
    , df_datos = self.df_scrolledtext
    , columna_df_para_informar = "CODIGO"
    , height_scrolledtext = self.kwargs_config_scrolledtext_1.get("height", 1)
    , lista_dicc_tag_linea_completa = self.kwargs_config_scrolledtext_1.get("lista_dicc_tag_linea_completa", None)

self.scrolledtext_2.modificaciones("agregar_contenido_y_tags_desde_dataframe"
    , df_datos = self.df_scrolledtext
    , columna_df_para_informar = "CODIGO"
    , height_scrolledtext = self.kwargs_config_scrolledtext_2.get("height", 1)
    , lista_dicc_tag_caracteres_cambiantes_comparativa = self.kwargs_config_scrolledtext_2.get("lista_dicc_tag_caracteres_cambiantes_comparativa", None)

self.scrolledtext_3.modificaciones("agregar_contenido_y_tags_desde_dataframe"
    , df_datos = self.df_scrolledtext
    , columna_df_para_informar = "CODIGO"
    , height_scrolledtext = self.kwargs_config_scrolledtext_3.get("height", 1)
    , lista_dicc_tag_linea_completa = self.kwargs_config_scrolledtext_3.get("lista_dicc_tag_linea_completa", None)
    , lista_dicc_tag_caracteres_cambiantes_comparativa = self.kwargs_config_scrolledtext_3.get("lista_dicc_tag_caracteres_cambiantes_comparativa", None)

if __name__ == "__main__":
    kwargs_config_root = {"dicc_config_root":
        ("title": "PRUEBA ROOT"
        , "iconbitmap": ico_tapar_pluma_tkinter
        , "tupla_geometry": (400, 340)
        , "resizable": (1, 1)
        )
    }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_de_mi_proyecto(root)
    root.widget_objeto.mainloop()
```

La clase **scrolledtext_propio** dispone del método propio **modificaciones** para agregar contenido desde un dataframe aplicando tags (opción **agregar_contenido_y_tags_desde_dataframe**).

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

CASO 2. Directamente en el módulo de *mi_proyecto*

```

ico_tapar_pluma_tkinter = (os.path.join(sys._MEIPASS, "ico_tapar_pluma_tkinter.ico")
                             if getattr(sys, 'frozen', False)
                             else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

lista_texto_para_df = [['CREATE TABLE Production.Document', 'CREATE TABLE Production.Document', None]
                        , ['t(' , 't(' , None], ['[DocumentNode] HIERARCHYID(892)', '[Owner] INT', 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [DocumentLevel] SMALLINT', ' , [DocumentLevel] smallint', 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [Title] NVARCHAR(50)', ' , ' , 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [Owner] INT', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [FolderFlag] BIT', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [FileName] NVARCHAR(400)', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [FileExtension] NVARCHAR(8)', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [Revision] NCHAR(5)', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [ChangeNumber] INT', None, None ]
                        , [' , [Status] TINYINT', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [DocumentSummary] NVARCHAR(-1)', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [Document] VARBINARY(-1)', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [rowguid] UNIQUEIDENTIFIER', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' , [ModifiedDate] DATETIME', None, 'CAMBIOS_LOCALIZADOS' ]
                        , [' )', None, None]]

df_scrolledtext = pd.DataFrame(lista_texto_para_df, columns = ["CODIGO", "CODIGO_OTRA_BBDD", "CONTROL_CAMBIOS_ACTUAL"])

kwargs_config_root = {"dicc_config_root":
                      {
                        "title": "PRUEBA ROOT"
                        , "iconbitmap": ico_tapar_pluma_tkinter
                        , "tuple_geometry": (400, 340)
                        , "resizable": (1, 1)
                      }
                      }

kwargs_config_frame = {"width": 380
                      , "height": 320
                      , "bg": "#ACADB1"
                      , "bd": 2
                      , "relief": "solid"
                      , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                      }

```

```
kwidgets_config_scrolledtext_1 = {"font": ("Calibri", 10, "bold")
    , "width": 40
    , "height": 5
    , "state": tk.NORMAL
    , "bg": "#B7C3F5"
    , "fg": "black"
    , "wrap": tk.NONE
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
    , "duracion": tk.LEFT

    lista_dicc_tag_linea_completa: [
        [{"nombre_tag": "CAMBIOS_LOCALIZADOS"}
         , "columna_df_tag_aplicar": "CONTROL_CAMBIOS_ACTUAL"
         , "case_sensitive": False
         , "dicc_config": {"background": "#05FB27"}]
    ]
}

kwidgets_config_scrolledtext_2 = {"font": ("Calibri", 10, "bold")
    , "width": 40
    , "height": 5
    , "state": tk.NORMAL
    , "bg": "#B7C3F5"
    , "fg": "black"
    , "wrap": tk.NONE
    , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 110}
    , "duracion": tk.LEFT

    lista_dicc_tag_caracteres_cambiantes_comparativa: [
        [{"nombre_tag": "CAMBIOS_LOCALIZADOS_POR_INDICES"}
         , "columna_df_filtro_registros_aplicar_tag": "CONTROL_CAMBIOS_ACTUAL"
         , "columna_df_filtro_registros_aplicar_tag_valor": "CAMBIOS_LOCALIZADOS"
         , "columna_df_comparar_1": "COORDINO"
         , "columna_df_comparar_2": "COORDINO_OTRA_BBDDO"
         , "case_sensitive": True
         , "marcar_toda_linea_si_todo_varia": False
         , "dicc_config": {"foreground": "red"}]
    ]
}
```

```
kwidgets_config scrolledtext_3 = {"font": ("Calibri", 10, "bold"),  
    "width": 40,  
    "height": 5,  
    "state": tk.NORMAL,  
    "bg": "#B7C9F5",  
    "fg": "black",  
    "wrap": tk.NONE,  
    "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 210}  
},  
  
def crear_diccionario():  
    lista_dic_tag_lineas_completa=[  
        [{"nombre_tag": "CAMBIOS_LOCALIZADOS"  
            , "columna_df_filtro_registros_aplicar_tag": "CONTROL_CAMBIOS_ACTUAL"  
            , "case_sensitive": True  
            , "dicc_config": {"background": "#05FB27"}  
        }]  
    )  
  
    lista_dic_tag_caracteres_cambiantes_comparativo=[  
        [{"nombre_tag": "CAMBIOS_LOCALIZADOS_POR_INDICES"  
            , "columna_df_filtro_registros_aplicar_tag": "CONTROL_CAMBIOS_ACTUAL"  
            , "columna_df_filtro_registros_aplicar_tag_valor": "CAMBIOS_LOCALIZADOS"  
            , "columna_df_comparar_1": "CODIGO"  
            , "columna_df_comparar_2": "CODIGO_OTRA_BECO"  
            , "case_sensitive": True  
            , "marcar_toda_linea_si_todo_varia": false  
            , "dicc_config": {"foreground": "red"}  
        }]  
    )  
}
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

16

EJEMPLO 16 – scrolledtext desde un dataframe aplicando tags

CASO 2. Directamente en el módulo de *mi_proyecto*

```
root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
frame = mod_utils.gui_tkinter_widgets(root.widget_objeto, tipo_widget_param = "frame", **kwargs_config_frame)

scrolledtext_1 = mod_utils.scrolledtext_propio(frame.widget_objeto, **kwargs_config_scrolledtext_1)
scrolledtext_2 = mod_utils.scrolledtext_propio(frame.widget_objeto, **kwargs_config_scrolledtext_2)
scrolledtext_3 = mod_utils.scrolledtext_propio(frame.widget_objeto, **kwargs_config_scrolledtext_3)

scrolledtext_1.modificaciones("agregar_contenido_y_tags_desde_dataframe"
                              , df_datos = df_scrolledtext
                              , columna_df_para_informar = "CODIGO"
                              , height_scrolledtext = kwargs_config_scrolledtext_1.get("height", 1)
                              , lista_dicc_tag_linea_completa = kwargs_config_scrolledtext_1.get("lista_dicc_tag_linea_completa", None)
                              )

scrolledtext_2.modificaciones("agregar_contenido_y_tags_desde_dataframe"
                              , df_datos = df_scrolledtext
                              , columna_df_para_informar = "CODIGO"
                              , height_scrolledtext = kwargs_config_scrolledtext_2.get("height", 1)
                              , lista_dicc_tag_caracteres_cambiantes_comparativa = kwargs_config_scrolledtext_2.get("lista_dicc_tag_caracteres_cambiantes_comparativa", None)
                              )

scrolledtext_3.modificaciones("agregar_contenido_y_tags_desde_dataframe"
                              , df_datos = df_scrolledtext
                              , columna_df_para_informar = "CODIGO"
                              , height_scrolledtext = kwargs_config_scrolledtext_3.get("height", 1)
                              , lista_dicc_tag_linea_completa = kwargs_config_scrolledtext_3.get("lista_dicc_tag_linea_completa", None)
                              , lista_dicc_tag_caracteres_cambiantes_comparativa = kwargs_config_scrolledtext_3.get("lista_dicc_tag_caracteres_cambiantes_comparativa", None)
                              )

root.widget_objeto.mainloop()
```

MANUAL PARA USAR EL SISTEMA DE CLASES TKINTER EN OTROS PROYECTOS

17

EJEMPLO 17 – Configuración de atributos tkinter no integrados en `mi_sistema_tkinter` (messagebox, filedialog etc) dentro de clases propias en `mi_proyecto`

```
ico_tapar_pluma_tkinter = (os.path.join(sys.MEIPASS, "ico_tapar_pluma_tkinter.ico")
                           if getattr(sys, 'frozen', False)
                           else os.path.join(pathlib.Path(__file__).parent.absolute(), "ico_tapar_pluma_tkinter.ico"))

class clase_de_mi_proyecto:

    def __init__(self, master):

        self.master = master

        self.kwargs_config_frame = {"width": 380
                                     , "height": 180
                                     , "bg": "#ACADB1"
                                     , "bd": 2
                                     , "relief": "solid"
                                     , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                                    }

        self.kwargs_config_boton = {"text": "botón"
                                     , "width": 5
                                     , "bg": "black"
                                     , "fg": "white"
                                     , "controltip": "esto botón solo imprime"
                                     , "dicc_colocacion": {"metodo": "place", "coord_x": 10, "coord_y": 10}
                                     , "dicc_rutina": {"rutina": "def_rutina_boton"}
                                    }

        self.frame = mod_utils.gui_tkinter_widgets(self.master.widget_objeto, tipo_widget_param = "frame", **self.kwargs_config_frame)
        self.boton = mod_utils.gui_tkinter_widgets(self.frame.widget_objeto, tipo_widget_param = "button", self_clase_gui_donde_call_rutina = self, **self.kwargs_config_boton)

    def def_rutina_boton(self):

        filedialog = fd.askdirectory(parent = self.master.widget_objeto, title = "Selecciona en que directorio quieres que se guarde la guía de usuario:")

        self.master.widget_objeto.config(cursor = "wait")

        self.master.widget_objeto.config(cursor = "")

if __name__ == "__main__":

    kwargs_config_root = {"dicc_config_root":
                          {"title": "PRUEBA ROOT"
                           , "iconbitmap": ico_tapar_pluma_tkinter
                           , "tuple_geometry": (400, 340)
                           , "resizable": (1, 1)
                          }
                          }

    root = mod_utils.gui_tkinter_widgets(None, tipo_widget_param = "root", **kwargs_config_root)
    clase_de_mi_proyecto(root)
    root.widget_objeto.mainloop()
```

Los objetos tkinter nativos no configurados en `mi_sistema_tkinter` que se puedan integrar en la GUI de `mi_proyecto` en una clase propia han de tener el ***master*** seguido del atributo ***widget_objeto***.