ttkwidgets Documentation

Release 0.11.0

ttwidgets developpers

Apr 08, 2020

Contents:

A collection of widgets for Tkinter's ttk extensions by various authors

1 Authors

List of all the authors of widgets in this repository. Please note that this list only mentions the original creators of the widgets, and the widgets may have been edited and/or improved or otherwise modified by other authors.

- RedFantom
  - *ScrolledFrame*, based on an Unpythonic idea
  - *ToggledFrame*, based on an idea by Onlyjus
  - *LinkLabel*, based on an idea by Nelson Brochado
  - *ScrolledListbox*

  - *FontChooser*, based on an idea by Nelson Brochado
  - *FontSelectFrame*
  - *Tooltip*
  - *ItemsCanvas*
  - *TimeLine*
- The Python Team
  - *Calendar*, found here
- Mitja Martini
  - *AutocompleteEntry*, found here
- Russell Adams
  - *AutocompleteCombobox*, found here
- Juliette Monsel
  - *CheckboxTreeview*
  - *Table*
  - *TickScale*
  - *AutoHideScrollbar* based on an idea by Fredrik Lundh
  - All color widgets: *askcolor()*, *ColorPicker*, *GradientBar* and *ColorSquare*, `LimitVar`, `Spinbox`, *AlphaBar* and supporting functions in `functions.py`.
  - *AutocompleteEntryListbox*
- Multiple authors:
  - *ScaleEntry* (RedFantom and Juliette Monsel)

2 Installation

- With pip:

```
pip install ttkwidgets
```

- Ubuntu: ttkwidgets is available in the PPA ppa:j-4321-i/ttkwidgets.

```
sudo add-apt-repository ppa:j-4321-i/ttkwidgets
sudo apt-get update
sudo apt-get install python(3)-ttkwidgets
```

- Archlinux: ttkwidgets is available in AUR.

3 Documentation

Note: Only the widgets' specific options and methods are documented here, to get information about the options and methods inherited from standard tk/ttk widgets, consult tkinter's documentation.

3.1 ttkwidgets

Classes

AutoHideScrollbar	Scrollbar that automatically hides when not needed.
Calendar	ttk Widget that enables a calender within a frame, allowing the user to select dates.
CheckboxTreeview	`ttk.Treeview` widget with checkboxes left of each item.
DebugWindow	A Toplevel that shows sys.stdout and sys.stderr for Tkinter applications
ItemsCanvas	A `ttk.Frame` containing a Canvas upon which text items can be placed with a coloured background.
LinkLabel	A `ttk.Label` that can be clicked to open a link with a default blue color, a purple color when clicked and a bright blue color when hovering over the Label.
ScaleEntry	A simple combination of a Scale and an Entry widget suitable for use with int ranges.
ScrolledListbox	Simple `tk.Listbox` with an added scrollbar.
Table	Table widget displays a table with options to drag rows and columns and to sort columns.
TickScale	A `ttk.Scale` that can display the current value next to the slider and supports ticks.
TimeLine	A Frame containing a Canvas and various buttons to manage a timeline that can be marked with certain events, allowing the binding of commands to hovering over certain elements and creating texts inside the elements.

AutoHideScrollbar

class ttkwidgets.**AutoHideScrollbar**(*master=None*, ***kwargs*)

Bases: `ttk.Scrollbar`

Scrollbar that automatically hides when not needed.

__init__(*master=None*, ***kwargs*)

Create a scrollbar.

Parameters

- **master** (*widget*) – master widget
- **kwargs** – options to be passed on to the `ttk.Scrollbar` initializer

grid(***kw*)

Position a widget in the parent widget in a grid.

Parameters

- **column** (*int*) – use cell identified with given column (starting with 0)
- **columnspan** (*int*) – this widget will span several columns
- **in_** (*widget*) – widget to use as container
- **ipadx** (*int*) – add internal padding in x direction

- **ipady** (*int*) – add internal padding in y direction
- **padx** (*int*) – add padding in x direction
- **pady** (*int*) – add padding in y irection
- **row** (*int*) – use cell identified with given row (starting with 0)
- **rowspan** (*int*) – this widget will span several rows
- **sticky** (*str*) – “n”, “s”, “e”, “w” or combinations: if cell is larger on which sides will this widget stick to the cell boundary

pack (***kw*)

Pack a widget in the parent widget.

Parameters

- **after** (*widget*) – pack it after you have packed widget
- **anchor** (*str*) – position anchor according to given direction (“n”, “s”, “e”, “w” or combinations)
- **before** (*widget*) – pack it before you will pack widget
- **expand** (*bool*) – expand widget if parent size grows
- **fill** (*str*) – “none” or “x” or “y” or “both”: fill widget if widget grows
- **in_** (*widget*) – widget to use as container
- **ipadx** (*int*) – add internal padding in x direction
- **ipady** (*int*) – add internal padding in y direction
- **padx** (*int*) – add padding in x direction
- **pady** (*int*) – add padding in y irection
- **side** (*str*) – “top” (default), “bottom”, “left” or “right”: where to add this widget

place (***kw*)

Place a widget in the parent widget.

Parameters

- **in_** (*widget*) – master relative to which the widget is placed
- **x** (*int*) – locate anchor of this widget at position x of master
- **y** (*int*) – locate anchor of this widget at positiony of master
- **relx** (*float*) – locate anchor of this widget between 0 and 1 relative to width of master (1 is right edge)
- **rely** (*float*) – locate anchor of this widget between 0 and 1 relative to height of master (1 is bottom edge)
- **anchor** (*str*) – position anchor according to given direction (“n”, “s”, “e”, “w” or combinations)
- **width** (*int*) – width of this widget in pixel
- **height** (*int*) – height of this widget in pixel
- **relwidth** (*float*) – width of this widget between 0.0 and 1.0 relative to width of master (1.0 is the same width as the master)

- **relheight** (*float*) – height of this widget between 0.0 and 1.0 relative to height of master (1.0 is the same height as the master)
- **bordermode** (*str*) – "inside" or "outside": whether to take border width of master widget into account

set(*lo*, *hi*)
Set the fractional values of the slider position.

Parameters

- **lo** (*float*) – lower end of the scrollbar (between 0 and 1)
- **hi** (*float*) – upper end of the scrollbar (between 0 and 1)

Calendar

class ttkwidgets.**Calendar**(*master=None*, ***kw*)
Bases: ttk.Frame

ttk Widget that enables a calender within a frame, allowing the user to select dates.

Credits to: The Python team
Source: The Python/ttk samples
License: The Python GPL-compatible license

__init__(*master=None*, ***kw*)
Create a Calendar.

Parameters

- **master** (*widget*) – master widget
- **locale** (*str*) – calendar locale (defines the language, date formatting)
- **firstweekday** (*int*) – first day of the week, 0 is monday
- **year** (*int*) – year to display
- **month** (*int*) – month to display
- **selectbackground** (*str*) – background color of the selected day
- **selectforeground** (*str*) – selectforeground color of the selected day
- **kw** – options to be passed on to the ttk.Frame initializer

class datetime(*year*, *month*, *day*[, *hour*[, *minute*[, *second*[, *microsecond*[, *tzinfo*]]]]])
Bases: datetime.date

The year, month and day arguments are required. tzinfo may be None, or an instance of a tzinfo subclass. The remaining arguments may be ints or longs.

astimezone()
tz -> convert to local time in new timezone tz

combine()
date, time -> datetime with same date and time fields

ctime()
Return ctime() style string.

date()
Return date object with same year, month and day.

dst()
Return self.tzinfo.dst(self).

fromtimestamp()
timestamp[, tz] -> tz's local time from POSIX timestamp.

isoformat()
[sep] -> string in ISO 8601 format, YYYY-MM-DDTHH:MM:SS[.mmmmmm][+HH:MM].

sep is used to separate the year from the time, and defaults to 'T'.

now()
[tz] -> new datetime with tz's local day and time.

replace()
Return datetime with new specified fields.

strptime()
string, format -> new datetime parsed from a string (like time.strptime()).

time()
Return time object with same time but with tzinfo=None.

timetuple()
Return time tuple, compatible with time.localtime().

timetz()
Return time object with same time and tzinfo.

tzname()
Return self.tzinfo.tzname(self).

utcfromtimestamp()
timestamp -> UTC datetime from a POSIX timestamp (like time.time()).

utcnow()
Return a new datetime representing UTC day and time.

utcoffset()
Return self.tzinfo.utcoffset(self).

utctimetuple()
Return UTC time tuple, compatible with time.localtime().

selection
Return the currently selected date.

Return type *datetime*

class timedelta
Bases: `object`

Difference between two datetime values.

days
Number of days.

microseconds
Number of microseconds (>= 0 and less than 1 second).

seconds
Number of seconds (>= 0 and less than 1 day).

total_seconds()
Total seconds in the duration.

CheckboxTreeview

class ttkwidgets.**CheckboxTreeview**(*master=None*, ***kw*)
Bases: ttk.Treeview

ttk.Treeview widget with checkboxes left of each item.

Note: The checkboxes are done via the image attribute of the item, so to keep the checkbox, you cannot add an image to the item.

__init__(*master=None*, ***kw*)
Create a CheckboxTreeview.

Parameters

- **master** (*widget*) – master widget
- **kw** – options to be passed on to the ttk.Treeview initializer

change_state(*item*, *state*)
Replace the current state of the item.

i.e. replace the current state tag but keeps the other tags.

Parameters

- **item** (*str*) – item id
- **state** (*str*) – "checked", "unchecked" or "tristate": new state of the item

collapse_all()
Collapse all items.

expand_all()
Expand all items.

get_checked()
Return the list of checked items that do not have any child.

insert(*parent*, *index*, *iid=None*, ***kw*)
Creates a new item and return the item identifier of the newly created item.

Parameters

- **parent** (*str*) – identifier of the parent item
- **index** (*int or "end"*) – where in the list of parent's children to insert the new item
- **iid** (*None or str*) – item identifier, iid must not already exist in the tree. If iid is None a new unique identifier is generated.
- **kw** – other options to be passed on to the ttk.Treeview.insert() method

Returns the item identifier of the newly created item

Return type str

Note: Same method as for the standard `ttk.Treeview` but add the tag for the box state accordingly to the parent state if no tag among ('checked', 'unchecked', 'tristate') is given.

state(*statespec=None*)
Modify or inquire widget state.

Parameters **statespec** (*None or sequence[str]*) – Widget state is returned if *statespec* is None, otherwise it is set according to the statespec flags and then a new state spec is returned indicating which flags were changed.

tag_add(*item*, *tag*)
Add tag to the tags of item.

Parameters

- **item** (*str*) – item identifier
- **tag** (*str*) – tag name

tag_del(*item*, *tag*)
Remove tag from the tags of item.

Parameters

- **item** (*str*) – item identifier
- **tag** (*str*) – tag name

DebugWindow

class ttkwidgets.**DebugWindow**(*master=None*, *title='Debug window'*, *stdout=True*, *stderr=False*, *width=70*, *autohidescrollbar=True*, ***kwargs*)
Bases: `Tkinter.Toplevel`

A Toplevel that shows sys.stdout and sys.stderr for Tkinter applications

__init__(*master=None*, *title='Debug window'*, *stdout=True*, *stderr=False*, *width=70*, *autohidescrollbar=True*, ***kwargs*)
Create a Debug window.

Parameters

- **master** (*widget*) – master widget
- **stdout** (*bool*) – whether to redirect stdout to the widget
- **stderr** (*bool*) – whether to redirect stderr to the widget
- **width** (*int*) – window width (in characters)
- **autohidescrollbar** (*bool*) – whether to use an *AutoHideScrollbar* or a `ttk.Scrollbar`
- **kwargs** – options to be passed on to the `tk.Toplevel` initializer

quit()
Restore previous stdout/stderr and destroy the window.

save()
Save widget content.

write(*line*)
Write line at the end of the widget.

Parameters **line** (*str*) – text to insert in the widget

ItemsCanvas

class ttkwidgets.**ItemsCanvas**(**args*, ***kwargs*)

Bases: ttk.Frame

A ttk.Frame containing a Canvas upon which text items can be placed with a coloured background.

The items can be moved around and deleted. A background can also be set.

__init__(**args*, ***kwargs*)

Create an ItemsCanvas.

Parameters

- **canvaswidth** (*int*) – width of the canvas in pixels
- **canvasheight** – height of the canvas in pixels
- **callback_add** (*function*) – callback for when an item is created, *(int item, int rectangle)
- **callback_del** (*function*) – callback for when an item is deleted, *(int item, int rectangle)
- **callback_move** (*function*) – callback for when an item is moved, *(int item, int rectangle, int x, int y)
- **function_new** (*function*) – user defined function for when an item is created, *(self.add_item)

add_item(*text*, *font=('default'*, *12*, *'bold')*, *backgroundcolor='yellow'*, *textcolor='black'*, *highlightcolor='blue'*)

Add a new item on the Canvas.

Parameters

- **text** (*str*) – text to display
- **font** (tuple or Font) – font of the text
- **backgroundcolor** (*str*) – background color
- **textcolor** (*str*) – text color
- **highlightcolor** (*str*) – the color of the text when the item is selected

cget(*key*)

Query widget option.

Parameters **key** (*str*) – option name

Returns value of the option

To get the list of options for this widget, call the method keys().

config(***kwargs*)

Configure resources of the widget.

To get the list of options for this widget, call the method keys(). See *__init__()* for a description of the widget specific option.

configure(***kwargs*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

del_item()
Delete the current item on the Canvas.

grid_widgets()
Put the widgets in the correct position.

left_motion(*event*)
Callback for the B1-Motion event, or the dragging of an item.

Moves the item to the desired location, but limits its movement to a place on the actual Canvas. The item cannot be moved outside of the Canvas.

Parameters **event** – Tkinter event

left_press(*event*)
Callback for the press of the left mouse button.

Selects a new item and sets its highlightcolor.

Parameters **event** – Tkinter event

left_release(*event*)
Callback for the release of the left button.

Parameters **event** – Tkinter event

right_press(*event*)
Callback for the right mouse button event to pop up the correct menu.

Parameters **event** – Tkinter event

set_background(*image=None*, *path=None*, *resize=True*)
Set the background image of the Canvas.

Parameters

- **image** (*PhotoImage*) – background image
- **path** (*str*) – background image path
- **resize** (*bool*) – whether to resize the image to the Canvas size

LinkLabel

class ttkwidgets.**LinkLabel**(*master=None*, ***kwargs*)
Bases: `ttk.Label`

A `ttk.Label` that can be clicked to open a link with a default blue color, a purple color when clicked and a bright blue color when hovering over the Label.

__init__(*master=None*, ***kwargs*)
Create a LinkLabel.

Parameters

- **master** – master widget
- **link** (*str*) – link to be opened

- **normal_color** (*str*) – text color when widget is created
- **hover_color** (*str*) – text color when hovering over the widget
- **clicked_color** (*str*) – text color when link is clicked
- **kwargs** – options to be passed on to the `ttk.Label` initializer

cget(*key*)
Query widget option.

Parameters **key** (*str*) – option name

Returns value of the option

To get the list of options for this widget, call the method *keys()*.

configure(***kwargs*)
Configure resources of the widget.

To get the list of options for this widget, call the method *keys()*. See *__init__()* for a description of the widget specific option.

keys()
Return a list of all resource names of this widget.

open_link(**args*)
Open the link in the web browser.

reset()
Reset Label to unclicked status if previously clicked.

ScaleEntry

class ttkwidgets.**ScaleEntry**(*master=None*, *scalewidth=50*, *entrywidth=5*, *from_=0*, *to=50*, *orient='horizontal'*, *compound='right'*, *entryscalepad=0*, ***kwargs*)
Bases: `ttk.Frame`

A simple combination of a Scale and an Entry widget suitable for use with int ranges.

__init__(*master=None*, *scalewidth=50*, *entrywidth=5*, *from_=0*, *to=50*, *orient='horizontal'*, *compound='right'*, *entryscalepad=0*, ***kwargs*)
Create a ScaleEntry.

Parameters

- **master** (*widget*) – master widget
- **scalewidth** (*int*) – width of the Scale in pixels
- **entrywidth** (*int*) – width of the Entry in characters
- **from_** (*int*) – start value of the scale
- **to** (*int*) – end value of the scale
- **orient** (*str*) – scale orientation. Supports `tk.HORIZONTAL` and `tk.VERTICAL`
- **compound** (*str*) – side the Entry must be on. Supports `tk.LEFT`, `tk.RIGHT`, `tk.TOP` and `tk.BOTTOM`
- **entryscalepad** (*int*) – space between the entry and the scale
- **kwargs** – keyword arguments passed on to the `ttk.Frame` initializer

class LimitedIntVar(*low*, *high*)
Bases: `Tkinter.IntVar`

Subclass of `tk.IntVar` that allows limits in the value of the variable stored.

configure(***kwargs*)
Configure the limits of the LimitedIntVar.

set(*value*)
Set a new value.

Check whether value is in limits first. If not, return False and set the new value to either be the minimum (if value is smaller than the minimum) or the maximum (if the value is larger than the maximum). Both str and int are supported as value types, as long as the str contains an int.

Parameters **value** (*int*) – new value

cget(*key*)
Query widget option.

Parameters **key** (*str*) – option name

Returns value of the option

To get the list of options for this widget, call the method `keys()`.

cget_entry(*key*)
Query the Entry widget's option.

Parameters **key** (*str*) – option name

Returns value of the option

cget_scale(*key*)
Query the Scale widget's option.

Parameters **key** (*str*) – option name

Returns value of the option

config(*cnf={}*, ***kw*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

config_entry(*cnf={}*, ***kwargs*)
Configure resources of the Entry widget.

config_scale(*cnf={}*, ***kwargs*)
Configure resources of the Scale widget.

configure(*cnf={}*, ***kw*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

value
Get the value of the *LimitedIntVar* instance of the class.

ScrolledListbox

class ttkwidgets.**ScrolledListbox**(*master=None*, *compound='right'*, *autohidescrollbar=True*, ***kwargs*)

Bases: `ttk.Frame`

Simple `tk.Listbox` with an added scrollbar.

__init__(*master=None*, *compound='right'*, *autohidescrollbar=True*, ***kwargs*)

Create a Listbox with a vertical scrollbar.

Parameters

- **master** (*widget*) – master widget
- **compound** (*str*) – side for the Scrollbar to be on (`tk.LEFT` or `tk.RIGHT`)
- **autohidescrollbar** (*bool*) – whether to use an *AutoHideScrollbar* or a `ttk.Scrollbar`
- **kwargs** – keyword arguments passed on to the `tk.Listbox` initializer

config_listbox(**args*, ***kwargs*)

Configure resources of the Listbox widget.

Table

class ttkwidgets.**Table**(*master=None*, *show='headings'*, *drag_cols=True*, *drag_rows=True*, *sortable=True*, *class_='Table'*, ***kwargs*)

Bases: `ttk.Treeview`

Table widget displays a table with options to drag rows and columns and to sort columns.

This widget is based on the `ttk.Treeview` and shares many options and methods with it.

__init__(*master=None*, *show='headings'*, *drag_cols=True*, *drag_rows=True*, *sortable=True*, *class_='Table'*, ***kwargs*)

Create a Table.

Parameters

- **master** (*widget*) – master widget
- **drag_cols** (*bool*) – whether columns are draggable
- **drag_rows** (*bool*) – whether rows are draggable
- **sortable** (*bool*) – whether columns are sortable by clicking on their headings. The sorting order depends on the type of data (str, float, ...) which can be set with the column method.
- **show** (*str*) – which parts of the treeview to show (same as the Treeview option)
- **kwargs** – options to be passed on to the `ttk.Treeview` initializer

cget(*key*)

Query widget option.

Parameters **key** (*str*) – option name

Returns value of the option

To get the list of options for this widget, call the method `keys()`.

column (*column*, *option=None*, ***kw*)
Query or modify the options for the specified column.

If *kw* is not given, returns a dict of the column option values. If *option* is specified then the value for that option is returned. Otherwise, sets the options to the corresponding values.

Parameters

- **id** – the column's identifier (read-only option)
- **anchor** – "n", "ne", "e", "se", "s", "sw", "w", "nw", or "center": alignment of the text in this column with respect to the cell
- **minwidth** (*int*) – minimum width of the column in pixels
- **stretch** (*bool*) – whether the column's width should be adjusted when the widget is resized
- **width** (*int*) – width of the column in pixels
- **type** (*type*) – column's content type (for sorting), default type is *str*

configure (*cnf=None*, ***kw*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

delete (**items*)
Delete all specified items and all their descendants. The root item may not be deleted.

Parameters **items** (*sequence[str]*) – list of item identifiers

detach (**items*)
Unlinks all of the specified items from the tree.

The items and all of their descendants are still present, and may be reinserted at another point in the tree, but will not be displayed. The root item may not be detached.

Parameters **items** (*sequence[str]*) – list of item identifiers

heading (*column*, *option=None*, ***kw*)
Query or modify the heading options for the specified column.

If *kw* is not given, returns a dict of the heading option values. If *option* is specified then the value for that option is returned. Otherwise, sets the options to the corresponding values.

Parameters

- **text** (*str*) – text to display in the column heading
- **image** (*PhotoImage*) – image to display to the right of the column heading
- **anchor** (*str*) – "n", "ne", "e", "se", "s", "sw", "w", "nw", or "center": alignement of the heading text
- **command** (*function*) – callback to be invoked when the heading label is pressed.

insert (*parent*, *index*, *iid=None*, ***kw*)
Creates a new item and return the item identifier of the newly created item.

Parameters

- **parent** (*str*) – identifier of the parent item
- **index** (*int or "end"*) – where in the list of parent's children to insert the new item

- **iid** (*None or str*) – item identifier, iid must not already exist in the tree. If iid is None a new unique identifier is generated.
- **kw** – item's options: see *item()*

Returns the item identifier of the newly created item

Return type str

item(*item*, *option=None*, ***kw*)

Query or modify the options for the specified item.

If no options are given, a dict with options/values for the item is returned. If option is specified then the value for that option is returned. Otherwise, sets the options to the corresponding values as given by *kw*.

Parameters

- **text** (*str*) – item's label
- **image** (*PhotoImage*) – image to be displayed on the left of the item's label
- **values** (*sequence*) – values to put in the columns
- **open** (*bool*) – whether the item's children should be displayed
- **tags** (*sequence[str]*) – list of tags associated with this item

move(*item*, *parent*, *index*)

Moves item to position index in parent's list of children.

It is illegal to move an item under one of its descendants. If index is less than or equal to zero, item is moved to the beginning, if greater than or equal to the number of children, it is moved to the end. If item was detached it is reattached.

Parameters

- **item** (*str*) – item's identifier
- **parent** (*str*) – new parent of item
- **index** (*int of "end"*) – where in the list of parent's children to insert item

reattach(*item*, *parent*, *index*)

Moves item to position index in parent's list of children.

It is illegal to move an item under one of its descendants. If index is less than or equal to zero, item is moved to the beginning, if greater than or equal to the number of children, it is moved to the end. If item was detached it is reattached.

Parameters

- **item** (*str*) – item's identifier
- **parent** (*str*) – new parent of item
- **index** (*int of "end"*) – where in the list of parent's children to insert item

set(*item*, *column=None*, *value=None*)

Query or set the value of given item.

With one argument, return a dictionary of column/value pairs for the specified item. With two arguments, return the current value of the specified column. With three arguments, set the value of given column in given item to the specified value.

Parameters

- **item** (*str*) – item's identifier

- **column** (*str, int or None*) – column's identifier
- **value** – new value

set_children (*item*, **newchildren*)
Replaces item's children with newchildren.

Children present in item that are not present in newchildren are detached from tree. No items in newchildren may be an ancestor of item.

Parameters **newchildren** (*sequence[str]*) – new item's children (list of item identifiers)

TickScale

class ttkwidgets.**TickScale** (*master=None*, ***kwargs*)
Bases: ttk.Frame

A ttk.Scale that can display the current value next to the slider and supports ticks.

__init__ (*master=None*, ***kwargs*)
Create a TickScale with parent master.

Parameters

- **master** (*widget*) – master widget
- **digits** (*int*) – number of digits after the comma to display, if negative use the %g format
- **labelpos** (*str*) – "n", "s", "e or "w": if showvalue is True, position of the label
- **resolution** (*float*) – increment by which the slider can be moved. 0 means continuous sliding.
- **showvalue** (*bool*) – whether to display current value next to the slider
- **tickinterval** (*float*) – if not 0, display ticks with the given interval
- **tickpos** (*str*) – "w" or "e" (vertical scale), "n" or "s" (horizontal scale): if tickinterval is not 0, position of the ticks
- **kwargs** – options to be passed on to the ttk.Scale initializer (class, cursor, style, takefocus, command, from, length, orient, to, value, variable)

Note: The style must derive from "Vertical.TScale" or "Horizontal.TScale" depending on the orientation. Depending on the theme, the default slider length might not be correct. If it is the case, this can be solve by setting the 'sliderlength' through ttk.Style.

---

cget (*key*)
Query widget option.

Parameters **key** (*str*) – option name

Returns value of the option

To get the list of options for this widget, call the method keys().

config (*cnf={}*, ***kw*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

configure(*cnf={}*, ***kw*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

convert_to_pixels(*value*)
Convert value in the scale's unit into a position in pixels.

Parameters **value** (*float*) – value to convert

Returns the corresponding position in pixels

Return type float

TimeLine

class ttkwidgets.**TimeLine**(*master=None*, ***kwargs*)
Bases: `ttk.Frame`

A Frame containing a Canvas and various buttons to manage a timeline that can be marked with certain events, allowing the binding of commands to hovering over certain elements and creating texts inside the elements.

Each marker is pretty much a coloured rectangle with some optional text, that can be assigned tags. Tags may specify the colors of the marker, but tags can also be assigned callbacks that can be called with the identifier of the tag as well as a Tkinter event instance that was generated upon clicking. For example, the markers may be moved, or the user may want to add a menu that shows upon right-clicking. See the *create_marker()* function for more details on the markers.

The markers are put into a Canvas, which contains rows for each category. The categories are indicated by Labels and separated by black separating lines. Underneath the rows of categories, there is a second Canvas containing markers for the ticks of the time unit. Some time units get special treatment, such as "h" and "m", displayed in an appropriate H:M and M:S format respectively.

The height of the row for each category is automatically adjusted to the height of its respective Label to give a uniform appearance. All markers are redrawn if *draw_timeline()* is called, and therefore it should be called after any size change. Depending on the number of markers to draw, it may take a long time.

The TimeLine can be scrolled in two ways: horizontally (with `_scrollbar_timeline`) and vertically (with `_scrollbar_timeline_v`), which both use a class function as a proxy to allow for other functions to be called upon scrolling. The horizontal scrollbar makes a small pop-up window appear to indicate the time the cursor is currently pointing at on the timeline.

The markers can be retrieved from the class using the markers property, and they can be saved and then the markers can be recreated by calling *create_marker()* again for each marker. This functionality is not built into the class, if the user wants to do something like this, he or she should write the code required, as it can be done in different ways.

Some of the code has been inspired by the *ItemsCanvas*, as that is also a Canvas that supports the manipulation of items, but as this works in a fundamentally different way, the classes do not share any common parent class.

> **Warning:** This widget is *absolutely not* thread-safe, and it was not designed as such. It may work in some situations, but nothing is guaranteed when using this widget from multiple threads, even with Tkinter compiled with thread-safe flags or when using mtTkinter for Python 2.

Note: Some themes may conflict with this widget, for example because it makes the default font bigger for the category Labels. This should be fixed by the user by modifying the "TimeLine.T(Widget)" style.

__init__ (*master=None*, ***kwargs*)

Create a TimeLine widget

The style of the buttons can be modified by using the "TimeLine.TButton" style. The style of the surrounding Frame can be modified by using the "TimeLine.TFrame" style, or by specifying another style in the keyword arguments. The style of the category Labels can be modified by using the "TimeLine.TLabel" style.

Base TimeLine Widget Options

Parameters

- **width** (*int*) – Width of the timeline in pixels
- **height** (*int*) – Height of the timeline in pixels
- **extend** (*bool*) – Whether to extend when an item is moved out of range
- **start** (*float*) – Value to start the timeline at
- **finish** (*float*) – Value to end the timeline at
- **resolution** (*int*) – Amount of time per pixel [s/pixel]
- **tick_resolution** (*int*) – Amount of time between ticks on the timeline
- **unit** (*str*) – Unit of time. Some units have predefined properties, such as minutes ('m') and hours ('h'), which make the tick markers have an appropriate format.
- **zoom_enabled** (*bool*) – Whether to allow zooming on the timeline using the zoom buttons
- **categories** (*dict[Any, dict]*) – A dictionary with the names of the categories as the keys and the keyword argument dictionaries as values. Use an `OrderedDict` in order to preserve category order.
- **background** (*str*) – Background color for the Canvas widget
- **style** (*str*) – Style to apply to the Frame widget
- **zoom_factors** (*tuple[float]*) – Tuple of allowed zoom levels. For example: (1.0, 2.0, 5.0). The order of zoom levels is preserved.
- **zoom_default** (*float*) – Default zoom level to apply to the timeline
- **snap_margin** (*int*) – Amount of pixels between start and/or finish of a marker and a tick on the timeline before the marker is snapped into place
- **menu** (*tk.Menu*) – Menu to show when a right-click is performed somewhere on the TimeLine without a marker being active
- **autohidescrollbars** (*bool*) – whether to use *AutoHideScrollbar* or `ttk.Scrollbar` for the scrollbars

Marker Default Options

Parameters

- **marker_font** (*tuple*) – Font tuple to specify the default font for the markers
- **marker_background** (*str*) – Default background color for markers

- **marker_foreground** (*str*) – Default foreground color for markers
- **marker_outline** (*str*) – Default outline color for the markers
- **marker_border** (*int*) – Border width in pixels
- **marker_move** (*bool*) – Whether markers are allowed to move by default
- **marker_change_category** (*bool*) – Whether markers are allowed to change category by being dragged vertically
- **marker_allow_overlap** (*bool*) – Whether the markers are allowed to overlap. This setting is only enforced on the marker being moved. This means that when inserting markers, no errors will be raised, even with overlaps, and when an overlap-allowing marker is moved over an overlap-disallowing marker and overlap will still occur.
- **marker_snap_to_ticks** – Whether the markers should snap to the ticks when moved close to ticks automatically

active
Currently selected marker

Return type str

static calculate_text_coords(*rectangle_coords*)
Calculate Canvas text coordinates based on rectangle coords

call_callbacks(*iid*, *type*, *args*)
Call the available callbacks for a certain marker

Parameters

- **iid** (*str*) – marker identifier
- **type** (*str*) – type of callback (key in tag dictionary)
- **args** (*tuple*) – arguments for the callback

Returns amount of callbacks called

Return type int

cget(*item*)
Return the value of an option

static check_kwargs(*kwargs*)
Check the type and values of keyword arguments to *__init__()*

Parameters **kwargs** (*dict[str, Any]*) – Dictionary of keyword arguments

Raises TypeError, ValueError

check_marker_kwargs(*kwargs*)
Check the types of the keyword arguments for marker creation

Parameters **kwargs** (*dict*) – dictionary of options for marker creation

Raises TypeError, ValueError

clear_timeline()
Clear the contents of the TimeLine Canvas

Does not modify the actual markers dictionary and thus after redrawing all markers are visible again.

config(*cnf={}*, ***kwargs*)
Update options of the TimeLine widget

configure(*cnf={}*, ***kwargs*)
Update options of the TimeLine widget

create_marker(*category*, *start*, *finish*, *marker=None*, ***kwargs*)
Create a new marker in the TimeLine with the specified options

Parameters

- **category** (*Any*) – Category identifier, key as given in categories dictionary upon initialization
- **start** (*float*) – Start time for the marker
- **finish** (*float*) – Finish time for the marker
- **marker** (*dict[str, Any]*) – marker dictionary (replaces kwargs)

Marker Options

Options can be given either in the marker dictionary argument, or as keyword arguments. Given keyword arguments take precedence over tag options, which take precedence over default options.

Parameters

- **text** (*str*) – Text to show in the marker. Text may not be displayed fully if the zoom level does not allow the marker to be wide enough. Updates when resizing the marker.
- **background** (*str*) – Background color for the marker
- **foreground** (*str*) – Foreground (text) color for the marker
- **outline** (*str*) – Outline color for the marker
- **border** (*int*) – The width of the border (for which outline is the color)
- **font** (*tuple*) – Font tuple to set for the marker
- **iid** (*str*) – Unique marker identifier to use. A marker is generated if not given, and its value is returned. Use this option if keeping track of markers in a different manner than with auto-generated iid's is necessary.
- **tags** (*tuple[str]*) – Set of tags to apply to this marker, allowing callbacks to be set and other options to be configured. The option precedence is from the first to the last item, so the options of the last item overwrite those of the one before, and those of the one before that, and so on.
- **move** (*bool*) – Whether the marker is allowed to be moved

Additionally, all the options with the `marker_` prefix from *__init__()*, but without the prefix, are supported. Active state options are also available, with the `active_` prefix for `background`, `foreground`, `outline`, `border`. These options are also available for the hover state with the `hover_` prefix.

Returns identifier of the created marker

Return type str

Raises **ValueError** – One of the specified arguments is invalid

create_scroll_region()
Setup the scroll region on the Canvas

current
Currently active item on the _timeline Canvas

Return type str

current_iid
Currently active item's iid

Return type str

delete_marker(*iid*)
Delete a marker from the TimeLine

Parameters **iid** (*str*) – marker identifier

draw_categories()
Draw the category labels on the Canvas

draw_markers()
Draw all created markers on the TimeLine Canvas

draw_separators()
Draw the lines separating the categories on the Canvas

draw_ticks()
Draw the time tick markers on the TimeLine Canvas

draw_time_marker()
Draw the time marker on the TimeLine Canvas

draw_timeline()
Draw the contents of the whole TimeLine Canvas

get_position_time(*position*)
Get time for x-coordinate

Parameters **position** (*int*) – X-coordinate position to determine time for

Returns Time for the given x-coordinate

Return type float

get_time_position(*time*)
Get x-coordinate for given time

Parameters **time** (*float*) – Time to determine x-coordinate on Canvas for

Returns X-coordinate for the given time

Return type int

Raises ValueError

static get_time_string(*time*, *unit*)
Create a properly formatted string given a time and unit

Parameters

- **time** (*float*) – Time to format
- **unit** (*str*) – Unit to apply format of. Only supports hours ('h') and minutes ('m').

Returns A string in format '{whole}:{part}'

Return type str

grid_widgets()
Configure all widgets using the grid geometry manager

Automatically called by the *__init__()* method. Does not have to be called by the user except in extraordinary cases.

itemconfigure(*iid*, *rectangle_options*, *text_options*)
Configure options of items drawn on the Canvas

Low-level access to the individual elements of markers and other items drawn on the timeline Canvas. All modifications are overwritten when the TimeLine is redrawn.

marker_options
List of available options to create_marker

marker_tags(*iid*)
Generator for all the tags of a certain marker

markers
Return a dictionary with categories as keys

Return type dict[str, dict[str, Any]]

options
List of available options to *__init__()*

pixel_width
Width of the whole TimeLine in pixels

Return type int

static range(*start*, *finish*, *step*)
Like built-in `range()`, but with float support

set_time(*time*)
Set the time marker to a specific time

Parameters **time** (*float*) – Time to set for the time marker on the TimeLine

set_zoom_factor(*factor*)
Manually set a custom zoom factor

Parameters **factor** (*float*) – Custom zoom factor

tag_configure(*tag_name*, ***kwargs*)
Create a marker tag

Parameters

- **tag_name** – Identifier for the tag
- **move_callback** (*callable*) – Callback to be called upon moving a marker. Arguments to callback: `(iid: str, (old_start: float, old_finish: float), (new_start: float, new_finish: float))`
- **left_callback** (*callable*) – Callback to be called upon left clicking a marker. Arguments to callback: `(iid: str, x_coord: int, y_coord: int)`
- **right_callback** (*callable*) – Callback to be called upon right clicking a marker. Arguments to callback: `(iid: str, x_coord: int, y_coord: int)`
- **menu** (*tk.Menu*) – A Menu widget to show upon right click. Can be used with the right_callback option simultaneously.

In addition, supports all options supported by markers. Note that tag options are applied to markers upon marker creation, and thus is a tag is updated, the markers are not automatically updated as well.

time
Current value the time marker is pointing to

Return type float

update_active()
Update the active marker on the marker Canvas

update_marker(*iid*, ***kwargs*)
Change the options for a certain marker and redraw the marker

Parameters

- **iid** (*str*) – identifier of the marker to change
- **kwargs** (*dict*) – Dictionary of options to update

Raises ValueError

update_state(*iid*, *state*)
Set a custom state of the marker

Parameters

- **iid** (*str*) – identifier of the marker to set the state of
- **state** (*str*) – supports "active", "hover", "normal"

zoom_factor
Return the current zoom factor

Return type float

zoom_in()
Increase zoom factor and redraw TimeLine

zoom_out()
Decrease zoom factor and redraw TimeLine

zoom_reset()
Reset the zoom factor to default and redraw TimeLine

3.2 ttkwidgets.autocomplete

AutocompleteCombobox	`ttk.Combobox` widget that features autocompletion.
AutocompleteEntry	Subclass of `ttk.Entry` that features autocompletion.
AutocompleteEntryListbox	`ttk.Entry` that features autocompletion combined with a `tk.Listbox` to display the completion list.

AutocompleteCombobox

class ttkwidgets.autocomplete.**AutocompleteCombobox**(*master=None*, *completevalues=None*, ***kwargs*)

Bases: `ttk.Combobox`

`ttk.Combobox` widget that features autocompletion.

__init__(*master=None*, *completevalues=None*, ***kwargs*)
Create an AutocompleteCombobox.

Parameters

- **master** (*widget*) – master widget
- **completevalues** (*list*) – autocompletion values
- **kwargs** – keyword arguments passed to the `ttk.Combobox` initializer

autocomplete(*delta=0*)
Autocomplete the Combobox.

Parameters **delta** (*int*) – 0, 1 or -1: how to cycle through possible hits

cget(*key*)
Return value for widget specific keyword arguments

config(***kwargs*)
Alias for configure

configure(***kwargs*)
Configure widget specific keyword arguments in addition to `ttk.Combobox` keyword arguments.

handle_keyrelease(*event*)
Event handler for the keyrelease event on this widget.

Parameters **event** – Tkinter event

handle_return(*event*)
Function to bind to the Enter/Return key so if Enter is pressed the selection is cleared

Parameters **event** – Tkinter event

keys()
Return a list of all resource names of this widget.

set_completion_list(*completion_list*)
Use the completion list as drop down selection menu, arrows move through menu.

Parameters **completion_list** (*list*) – completion values

AutocompleteEntry

class ttkwidgets.autocomplete.**AutocompleteEntry**(*master=None*, *completevalues=None*, ***kwargs*)

Bases: `ttk.Entry`

Subclass of `ttk.Entry` that features autocompletion.

To enable autocompletion use *set_completion_list()* to define a list of possible strings to hit. To cycle through hits use down and up arrow keys.

__init__(*master=None*, *completevalues=None*, ***kwargs*)
Create an AutocompleteEntry.

Parameters

- **master** (*widget*) – master widget
- **completevalues** (*list*) – autocompletion values
- **kwargs** – keyword arguments passed to the `ttk.Entry` initializer

autocomplete(*delta=0*)
Autocomplete the Entry.

Parameters **delta** (*int*) – 0, 1 or -1: how to cycle through possible hits

cget(*key*)
Return value for widget specific keyword arguments

config(***kwargs*)
Alias for configure

configure(***kwargs*)
Configure widget specific keyword arguments in addition to `ttk.Entry` keyword arguments.

handle_keyrelease(*event*)
Event handler for the keyrelease event on this widget.

Parameters **event** – Tkinter event

handle_return(*event*)
Function to bind to the Enter/Return key so if Enter is pressed the selection is cleared.

Parameters **event** – Tkinter event

keys()
Return a list of all resource names of this widget.

set_completion_list(*completion_list*)
Set a new auto completion list

Parameters **completion_list** (*list*) – completion values

AutocompleteEntryListbox

class ttkwidgets.autocomplete.**AutocompleteEntryListbox**(*master=None*, *completevalues=[]*, *allow_other_values=False*, *autohidescrollbar=True*, ***kwargs*)

Bases: `ttk.Frame`

`ttk.Entry` that features autocompletion combined with a `tk.Listbox` to display the completion list.

__init__(*master=None*, *completevalues=[]*, *allow_other_values=False*, *autohidescrollbar=True*, ***kwargs*)
Create an Entry + Listbox widget with autocompletion.

Parameters

- **master** (*widget*) – master widget
- **completevalues** (*list*) – autocompletion values
- **allow_other_values** (*bool*) – whether the user is allowed to enter values not in the list
- **width** (*int*) – widget width (in characters)
- **exportselection** (*bool*) – whether to automatically export selected text to the clipboard
- **justify** (*str*) – text alignment in entry and listbox
- **font** – font in entry and listbox
- **autohidescrollbar** (*bool*) – whether to use an *AutoHideScrollbar* or a `ttk.Scrollbar`
- **kwargs** – keyword arguments passed to the `ttk.Frame` initializer

get()
Return the text in the entry.

3.3 ttkwidgets.color

Functions

ttkwidgets.color.**askcolor**(*color='red'*, *parent=None*, *title='Color Chooser'*, *alpha=False*)

Open a ColorPicker dialog and return the chosen color.

Returns the selected color in RGB(A) and hexadecimal #RRGGBB(AA) formats. (None, None) is returned if the color selection is cancelled.

Parameters

- **color** (*sequence[int] or str*) – initially selected color (RGB(A), HEX or tkinter color name)
- **parent** (*widget*) – parent widget
- **title** (*str*) – dialog title
- **alpha** (*bool*) – whether to display the alpha channel

Classes

AlphaBar	Bar to select alpha value.
ColorPicker	Color picker dialog.
ColorSquare	Square color gradient with selection cross.
GradientBar	HSV gradient colorbar with selection cursor.

AlphaBar

class ttkwidgets.color.**AlphaBar**(*parent*, *alpha=255*, *color=(255, 0, 0)*, *height=11*, *width=256*, *variable=None*, ***kwargs*)

Bases: Tkinter.Canvas

Bar to select alpha value.

__init__(*parent*, *alpha=255*, *color=(255, 0, 0)*, *height=11*, *width=256*, *variable=None*, ***kwargs*)

Create a bar to select the alpha value.

Parameters

- **parent** (*widget*) – parent widget
- **alpha** (*int*) – initially selected alpha value (between 0 and 255)
- **color** (*tuple[int]*) – gradient color in RGB format
- **variable** (*IntVar*) – variable linked to the alpha value
- **height** (*int*) – height of the widget in pixels
- **width** (*int*) – width of the widget in pixels
- **kwargs** – options to be passed on to the tk.Canvas initializer

get()

Return alpha value of color under cursor.

set(*alpha*)

Set cursor position on the color corresponding to the alpha value.

Parameters **alpha** (*int*) – new alpha value (between 0 and 255)

set_color(*color*)
Change gradient color and change cursor position if an alpha value is supplied.

Parameters **color** (*tuple[int]*) – new gradient color in RGB(A) format

ColorPicker

class ttkwidgets.color.**ColorPicker**(*parent=None*, *color=(255*, *0*, *0)*, *alpha=False*, *title='Color Chooser'*)
Bases: Tkinter.Toplevel

Color picker dialog.

__init__(*parent=None*, *color=(255*, *0*, *0)*, *alpha=False*, *title='Color Chooser'*)
Create a ColorPicker dialog.

Parameters

- **parent** (*widget*) – parent widget
- **color** (*sequence[int] or str*) – initially selected color (RGB(A), HEX or tkinter color name)
- **alpha** (*bool*) – whether to display the alpha channel
- **title** (*str*) – dialog title

get_color()
Return selected color, return an empty string if no color is selected.

Returns selected color as a (RGB, HSV, HEX) tuple or “”

ok()
Validate color selection and destroy dialog.

ColorSquare

class ttkwidgets.color.**ColorSquare**(*parent*, *hue*, *color=None*, *height=256*, *width=256*, ***kwargs*)
Bases: Tkinter.Canvas

Square color gradient with selection cross.

__init__(*parent*, *hue*, *color=None*, *height=256*, *width=256*, ***kwargs*)
Create a ColorSquare.

Parameters

- **parent** (*widget*) – parent widget
- **hue** (*int*) – hue (between 0 and 360) of the color square gradient (color in top right corner is (hue, 100, 100) in HSV)
- **color** (*tuple[int]*) – initially selected color given in HSV format
- **height** (*int*) – height of the widget in pixels
- **width** (*int*) – width of the widget in pixels
- **kwargs** – options to be passed on to the tk.Canvas initializer

get()
Get selected color.

Returns color under cursor as a (RGB, HSV, HEX) tuple

get_hue()
Return current hue.

set_hsv(*sel_color*)
Put cursor on sel_color given in HSV.

Parameters **sel_color** (*sequence(int)*) – color in HSV format

set_hue(*value*)
Change hue.

Parameters **value** (*int*) – new hue value (between 0 and 360)

set_rgb(*sel_color*)
Put cursor on sel_color given in RGB.

Parameters **sel_color** (*sequence(int)*) – color in RBG format

GradientBar

class ttkwidgets.color.**GradientBar**(*parent*, *hue=0*, *height=11*, *width=256*, *variable=None*, ***kwargs*)

Bases: Tkinter.Canvas

HSV gradient colorbar with selection cursor.

__init__(*parent*, *hue=0*, *height=11*, *width=256*, *variable=None*, ***kwargs*)
Create a GradientBar.

Parameters

- **parent** (*widget*) – parent widget
- **hue** (*int*) – initially selected hue value (between 0 and 360)
- **variable** (*IntVar*) – variable linked to the hue value
- **height** (*int*) – height of the widget in pixels
- **width** (*int*) – width of the widget in pixels
- **kwargs** – options to be passed on to the tk.Canvas initializer

get()
Return hue of color under cursor.

set(*hue*)
Set cursor position on the color corresponding to the hue value.

Parameters **hue** (*int*) – new hue value (between 0 and 360)

3.4 ttkwidgets.font

Functions

ttkwidgets.font.**askfont**()
Opens a *FontChooser* toplevel to allow the user to select a font

Returns font tuple (family_name, size, *options), `Font` object

Classes

FontChooser	A Toplevel to choose a `Font` from a list.
FontFamilyDropdown	A dropdown menu to select a font family with callback support and selection property.
FontFamilyListbox	*ScrolledListbox* listing all font families available on the system with a Scrollbar on the right with the option of a callback when double clicked and a property to get the font family name.
FontPropertiesFrame	Simple frame with buttons for Bold, Italic and Underline font types.
FontSelectFrame	A frame to use in your own application to let the user choose a font.
FontSizeDropdown	A dropdown with default font sizes

FontChooser

class `ttkwidgets.font.`**FontChooser**(*master=None*, ***kwargs*)

Bases: `Tkinter.Toplevel`

A Toplevel to choose a `Font` from a list. Should only be used through *askfont()*.

__init__(*master=None*, ***kwargs*)

Create a FontChooser.

Parameters

- **master** (*widget*) – master window
- **kwargs** – keyword arguments passed to `tk.Toplevel` initializer

font

Selected font.

Returns font tuple (family_name, size, *options), `Font` object

FontFamilyDropdown

class `ttkwidgets.font.`**FontFamilyDropdown**(*master=None*, *callback=None*, ***kwargs*)

Bases: `ttkwidgets.autocomplete.autocompletecombobox.AutocompleteCombobox`

A dropdown menu to select a font family with callback support and selection property.

__init__(*master=None*, *callback=None*, ***kwargs*)

Create a FontFamilyDropdown.

Parameters

- **master** (*widget*) – master widget
- **callback** (*function*) – callable object with single argument: font family name
- **kwargs** – keyword arguments passed on to the *AutocompleteCombobox* initializer

selection

Selection property.

Returns None if no font is selected and font family name if one is selected.

Return type None or str

FontFamilyListbox

class ttkwidgets.font.**FontFamilyListbox**(*master=None*, *callback=None*, ***kwargs*)

Bases: ttkwidgets.scrolledlistbox.ScrolledListbox

ScrolledListbox listing all font families available on the system with a Scrollbar on the right with the option of a callback when double clicked and a property to get the font family name.

__init__(*master=None*, *callback=None*, ***kwargs*)

Create a FontFamilyListbox.

Parameters

- **master** (*widget*) – master widget
- **callback** (*function*) – callable object with one argument: the font family name
- **kwargs** – keyword arguments passed to *ScrolledListbox*, in turn passed to tk.Listbox

selection

Selection property.

Returns None if no font is selected and font family name if one is selected.

Return type None or str

FontPropertiesFrame

class ttkwidgets.font.**FontPropertiesFrame**(*master=None*, *callback=None*, *label=True*, *fontsize=11*, ***kwargs*)

Bases: ttk.Frame

Simple frame with buttons for Bold, Italic and Underline font types.

__init__(*master=None*, *callback=None*, *label=True*, *fontsize=11*, ***kwargs*)

Create a FontPropertiesFrame.

Parameters

- **master** (*widget*) – master widget
- **callback** (*function*) – callback with argument (*bool* bold, *bool* italic, *bool* underline, *bool* overstrike)
- **label** (*str*) – show a header label
- **fontsize** (*int*) – size of the font on the buttons
- **kwargs** – keyword arguments passed on to the ttk.Frame initializer

bold

Bold property.

Returns True if bold is selected

Return type bool

italic
Italic property.

Returns True if italic is selected

Return type bool

overstrike
Overstrike property.

Returns True if overstrike is selected

Return type bool

underline
Underline property.

Returns True if underline is selected

Return type bool

FontSelectFrame

class ttkwidgets.font.**FontSelectFrame**(*master=None*, *callback=None*, ***kwargs*)
Bases: ttk.Frame

A frame to use in your own application to let the user choose a font.

For Font object, use *font* property.

__init__(*master=None*, *callback=None*, ***kwargs*)

Parameters

- **master** (*widget*) – master widget
- **callback** (*function*) – callback passed argument (*str* family, *int* size, *bool* bold, *bool* italic, *bool* underline)
- **kwargs** – keyword arguments passed on to the ttk.Frame initializer

font
Font property.

Returns a Font object if family is set, else None

Return type Font or None

FontSizeDropdown

class ttkwidgets.font.**FontSizeDropdown**(*master=None*, *callback=None*, ***kwargs*)
Bases: ttkwidgets.autocomplete.autocompletecombobox.AutocompleteCombobox

A dropdown with default font sizes

__init__(*master=None*, *callback=None*, ***kwargs*)

Parameters

- **master** (*widget*) – master widget
- **callback** (*function*) – callback on click with single argument: *int* size
- **kwargs** – keyword arguments passed on to the *AutocompleteCombobox* initializer

selection
Selection property.

Returns None if no value is selected and size if selected.

Return type None or int

3.5 ttkwidgets.frames

Classes

Tooltip	Simple help hover balloon.
ScrolledFrame	A frame that sports a vertically oriented scrollbar for scrolling.
ToggledFrame	A frame that can be toggled to open and close.

Tooltip

class ttkwidgets.frames.**Tooltip**(*master=None*, *headertext='Help'*, *text='Some great help is displayed here.'*, *width=200*, *timeout=1*, *background='#fef9cd'*, *offset=(2, 2)*, *showheader=True*, *static=False*, ***kwargs*)

Bases: ttk.Frame

Simple help hover balloon.

__init__(*master=None*, *headertext='Help'*, *text='Some great help is displayed here.'*, *width=200*, *timeout=1*, *background='#fef9cd'*, *offset=(2, 2)*, *showheader=True*, *static=False*, ***kwargs*)

Create a Tooltip

Parameters

- **master** (*widget*) – widget to bind the Tooltip to
- **headertext** (*str*) – text to show in window header
- **text** (*str*) – text to show as help text
- **width** (*int*) – width of the window
- **timeout** (*float*) – timeout in seconds to wait until the Tooltip is shown
- **background** (*str*) – background color of the Tooltip
- **offset** (*Tuple[int, int]*) – The offset from the mouse position the Ballon shows up
- **showheader** (*bool*) – Whether to display the header with image
- **static** (*bool*) – Whether to display the tooltip with static position. When the position is set to static, the balloon will always appear an offset from the bottom right corner of the widget.
- **kwargs** – keyword arguments passed on to the ttk.Frame initializer

cget(*key*)

Query widget option.

Parameters **key** (*str*) – option name

Returns value of the option

To get the list of options for this widget, call the method `keys()`.

config(***kwargs*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

configure(***kwargs*)
Configure resources of the widget.

To get the list of options for this widget, call the method `keys()`. See *__init__()* for a description of the widget specific option.

show()
Create the Toplevel and its children to show near the cursor

This is the callback for the delayed `<Enter>` event (see `_on_enter()`).

ScrolledFrame

class ttkwidgets.frames.**ScrolledFrame**(*master=None*, *compound='right'*, *canvasheight=400*, *canvaswidth=400*, *canvasborder=0*, *autohidescrollbar=True*, ***kwargs*)
Bases: `ttk.Frame`

A frame that sports a vertically oriented scrollbar for scrolling.

Variables **interior** – `ttk.Frame` in which to put the widgets to be scrolled with any geometry manager.

__init__(*master=None*, *compound='right'*, *canvasheight=400*, *canvaswidth=400*, *canvasborder=0*, *autohidescrollbar=True*, ***kwargs*)
Create a ScrolledFrame.

Parameters

- **master** (*widget*) – master widget
- **compound** (*str*) – "right" or "left": side the scrollbar should be on
- **canvasheight** (*int*) – height of the internal canvas
- **canvaswidth** (*int*) – width of the internal canvas
- **canvasborder** (*int*) – border width of the internal canvas
- **autohidescrollbar** (*bool*) – whether to use an *AutoHideScrollbar* or a `ttk.Scrollbar`
- **kwargs** – keyword arguments passed on to the `ttk.Frame` initializer

resize_canvas(*height=400*, *width=400*)
Function for the user to resize the internal Canvas widget if desired.

Parameters

- **height** (*int*) – new height in pixels
- **width** (*int*) – new width in pixels

ToggledFrame

class ttkwidgets.frames.**ToggledFrame**(*master=None*, *text=''*, *width=20*, *compound='left'*, ***kwargs*)

Bases: `ttk.Frame`

A frame that can be toggled to open and close.

Variables **interior** – `ttk.Frame` in which to put the widgets to be toggled with any geometry manager.

__init__(*master=None*, *text=''*, *width=20*, *compound='left'*, ***kwargs*)

Create a ToggledFrame.

Parameters

- **master** (*widget*) – master widget
- **text** (*str*) – text to display next to the toggle arrow
- **width** (*int*) – width of the closed ToggledFrame (in characters)
- **compound** (*str*) – "center", "none", "top", "bottom", "right" or "left": position of the toggle arrow compared to the text
- **kwargs** – keyword arguments passed on to the `ttk.Frame` initializer

toggle()

Toggle `ToggledFrame.interior` opened or closed.

Package structure

```
ttkwidgets
├── autocomplete
├── color
├── font
└── frames
```

Package	Content
ttkwidgets	Miscellanous widgets.
ttkwidgets.autocomplete	Autocompletion widgets.
ttkwidgets.color	Color choosing widgets.
ttkwidgets.font	Font choosing widgets.
ttkwidgets.frames	Frame based widgets.

4 Examples

4.1 ttkwidgets

Example: AutoHideScrollbar

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE
```

(continues on next page)

(continued from previous page)

```
from ttkwidgets import AutoHideScrollbar
import tkinter as tk

window = tk.Tk()
listbox = tk.Listbox(window, height=5)
scrollbar = AutoHideScrollbar(window, command=listbox.yview)
listbox.configure(yscrollcommand=scrollbar.set)

for i in range(10):
    listbox.insert('end', 'item %i' % i)

tk.Label(window, text="Increase the window's height\nto make the scrollbar vanish.").
↪pack(side='top', padx=4, pady=4)
scrollbar.pack(side='right', fill='y')
listbox.pack(side='left', fill='both', expand=True)

window.mainloop()
```

Example: Calendar

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets import Calendar
import tkinter as tk

def validate():
    sel = calendar.selection
    if sel is not None:
        label.configure(text='Selected date: %s' % sel.strftime('%x'))

window = tk.Tk()
calendar = Calendar(window, year=2015, month=3, selectforeground='white',
                    selectbackground='red')
calendar.pack()

tk.Button(window, text='Select', command=validate).pack()
label = tk.Label(window, text='Selected date:')
label.pack()
window.mainloop()
```

Example: CheckboxTreeview

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2017
# For license see LICENSE

from ttkwidgets import CheckboxTreeview
import tkinter as tk

```

(continues on next page)

(continued from previous page)

```
root = tk.Tk()

tree = CheckboxTreeview(root)
tree.pack()

tree.insert("", "end", "1", text="1")
tree.insert("1", "end", "11", text="11")
tree.insert("1", "end", "12",  text="12")
tree.insert("11", "end", "111", text="111")
tree.insert("", "end", "2", text="2")

root.mainloop()
```

Example: DebugWindow

```
# -*- coding: utf-8 -*-

# Copyright (c) RedFantom 2017
# Copyright (c) Juliette Monsel 2017
# For license see LICENSE

from ttkwidgets import DebugWindow
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
ttk.Button(root, text="Print ok", command=lambda: print('ok')).pack()
DebugWindow(root)
root.mainloop()
```

Example: ItemsCanvas

```
# -*- coding: utf-8 -*-

# Copyright (c) RedFantom 2017
# For license see LICENSE
from ttkwidgets import ItemsCanvas
import tkinter as tk
from tkinter import ttk


root = tk.Tk()

canvas = ItemsCanvas(root)
canvas.pack()

canvas.add_item("Example", font=("default", 13, "italic"), backgroundcolor="green",␣
↪textcolor="darkblue",
                highlightcolor="blue")

root.mainloop()
```

Example: LinkLabel

```
# -*- coding: utf-8 -*-

# Copyright (c) RedFantom 2017
# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets import LinkLabel
import tkinter as tk

window = tk.Tk()
LinkLabel(window, text="ttkwidgets repository",
          link="https://github.com/RedFantom/ttkwidgets",
          normal_color='royal blue',
          hover_color='blue',
          clicked_color='purple').pack()
window.mainloop()
```

Example: ScaleEntry

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets import ScaleEntry
import tkinter as tk


window = tk.Tk()
scaleentry = ScaleEntry(window, scalewidth=200, entrywidth=3, from_=0, to=20)
scaleentry.config_entry(justify='center')
scaleentry.pack()
window.mainloop()
```

Example: ScrolledListbox

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets import ScrolledListbox
import tkinter as tk

window = tk.Tk()
listbox = ScrolledListbox(window, height=5)

for i in range(10):
    listbox.listbox.insert('end', 'item {}'.format(i))

listbox.pack(fill='both', expand=True)
window.mainloop()
```

Example: Table

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets import Table
import tkinter as tk
from tkinter import ttk

root = tk.Tk()

root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)

style = ttk.Style(root)
style.theme_use('alt')
sortable = tk.BooleanVar(root, False)
drag_row = tk.BooleanVar(root, False)
drag_col = tk.BooleanVar(root, False)

columns = ["A", "B", "C", "D", "E", "F", "G"]
table = Table(root, columns=columns, sortable=sortable.get(), drag_cols=drag_col.
↪get(),
              drag_rows=drag_row.get(), height=6)
for col in columns:
    table.heading(col, text=col)
    table.column(col, width=100, stretch=False)

# sort column A content as int instead of strings
table.column('A', type=int)

for i in range(12):
    table.insert('', 'end', iid=i,
                 values=(i, i) + tuple(i + 10 * j for j in range(2, 7)))

# add scrollbars
sx = tk.Scrollbar(root, orient='horizontal', command=table.xview)
sy = tk.Scrollbar(root, orient='vertical', command=table.yview)
table.configure(yscrollcommand=sy.set, xscrollcommand=sx.set)

table.grid(sticky='ewns')
sx.grid(row=1, column=0, sticky='ew')
sy.grid(row=0, column=1, sticky='ns')
root.update_idletasks()


# toggle table properties
def toggle_sort():
    table.config(sortable=sortable.get())


def toggle_drag_col():
    table.config(drag_cols=drag_col.get())
```

(continues on next page)

(continued from previous page)

```
def toggle_drag_row():
    table.config(drag_rows=drag_row.get())


frame = tk.Frame(root)
tk.Checkbutton(frame, text='sortable', variable=sortable, command=toggle_sort).
↪pack(side='left')
tk.Checkbutton(frame, text='drag columns', variable=drag_col, command=toggle_drag_
↪col).pack(side='left')
tk.Checkbutton(frame, text='drag rows', variable=drag_row, command=toggle_drag_row).
↪pack(side='left')
frame.grid()
root.geometry('400x200')

root.mainloop()
```

Example: TickScale

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2017
# For license see LICENSE
from ttkwidgets import TickScale
import tkinter as tk
from tkinter import ttk


root = tk.Tk()
style = ttk.Style(root)
style.theme_use('clam')
style.configure('my.Vertical.TScale', sliderlength=50, background='white',
                foreground='red')
style.configure('my.Horizontal.TScale', sliderlength=10,
                font='TkDefaultFont 20 italic')
s1 = TickScale(root, orient='vertical', style='my.Vertical.TScale',
               tickinterval=0.2, from_=-1, to=1, showvalue=True, digits=2,
               length=400, labelpos='e')
s2 = TickScale(root, orient='horizontal', style='my.Horizontal.TScale',
               from_=0, to=10, tickinterval=2, resolution=1,
               showvalue=True, length=400)
s3 = TickScale(root, orient='horizontal', from_=0.25, to=1, tickinterval=0.1,
               resolution=0.1)

s1.pack(fill='y')
s2.pack(fill='x')
s3.pack(fill='x')

root.mainloop()
```

Example: TimeLine

```
# -*- coding: utf-8 -*-
```

(continues on next page)

(continued from previous page)

```
# Copyright (c) RedFantom 2017
# For license see LICENSE

import tkinter as tk
from ttkwidgets import TimeLine

window = tk.Tk()
timeline = TimeLine(
    window,
    categories={str(key): {"text": "Category {}".format(key)} for key in range(0, 5)},
    height=100, extend=True
)
menu = tk.Menu(window, tearoff=False)
menu.add_command(label="Some Action", command=lambda: print("Command Executed"))
timeline.tag_configure("1", right_callback=lambda *args: print(args), menu=menu,␣
↪foreground="green",
                       active_background="yellow", hover_border=2, move_
↪callback=lambda *args: print(args))
timeline.create_marker("1", 1.0, 2.0, background="white", text="Change Color", tags=(
↪"1",), iid="1")
timeline.create_marker("2", 2.0, 3.0, background="green", text="Change Category",␣
↪foreground="white", iid="2",
                       change_category=True)
timeline.create_marker("3", 1.0, 2.0, text="Show Menu", tags=("1",))
timeline.create_marker("4", 4.0, 5.0, text="Do nothing", move=False)
timeline.draw_timeline()
timeline.grid()
window.after(2500, lambda: timeline.configure(marker_background="cyan"))
window.after(5000, lambda: timeline.update_marker("1", background="red"))
window.after(5000, lambda: print(timeline.time))
window.mainloop()
```

4.2 ttkwidgets.autocomplete

Example: AutocompleteCombobox

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets.autocomplete import AutocompleteCombobox
import tkinter as tk

window = tk.Tk()
tk.Label(window, text="Combobox with autocompletion for the Tk instance's methods:").
↪pack(side='left')
entry = AutocompleteCombobox(window, width=20, completevalues=dir(window))
entry.pack(side='right')
window.mainloop()
```

Example: AutocompleteEntry

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets.autocomplete import AutocompleteEntry
import tkinter as tk

window = tk.Tk()
tk.Label(window, text="Entry with autocompletion for the Tk instance's methods:").
↪pack(side='left')
entry = AutocompleteEntry(window, width=20, completevalues=dir(window))
entry.pack(side='right')
window.mainloop()
```

Example: AutocompleteEntryListbox

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2019
# For license see LICENSE
import tkinter as tk
from ttkwidgets.autocomplete import AutocompleteEntryListbox

window = tk.Tk()
tk.Label(window, text="Entry + Listbox with autocompletion for the Tk instance's␣
↪methods:").pack()
entry = AutocompleteEntryListbox(window, width=20, completevalues=dir(window))
entry.pack()
window.mainloop()
```

4.3 ttkwidgets.color

Example: askcolor

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets.color import askcolor
import tkinter as tk
from tkinter import ttk
from PIL import Image, ImageTk


def pick(alpha=False):
    global im  # to avoid garbage collection of image
    res = askcolor('sky blue', parent=window, title='Pick a color', alpha=alpha)
    canvas.delete('image')
    if res[1] is not None:
```

(continues on next page)

(continued from previous page)

```
        im = ImageTk.PhotoImage(Image.new('RGBA', (100, 100), res[1]), master=window)
        canvas.create_image(60, 60, image=im, tags='image', anchor='center')
    print(res)


window = tk.Tk()
canvas = tk.Canvas(window, width=120, height=120)
canvas.create_text(60, 60, text='Background', anchor='center')
canvas.pack()
ttk.Button(window, text="Pick a color (No alpha channel)", command=pick).pack(fill='x
↪')
ttk.Button(window, text="Pick a color (With alpha channel)", command=lambda:␣
↪pick(True)).pack(fill='x')
window.mainloop()
```

4.4 ttkwidgets.font

Example: FontSelectFrame

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets.font import FontSelectFrame
import tkinter as tk
from tkinter import ttk

def update_preview(font_tuple):
    print(font_tuple)
    font = font_selection.font[0]
    if font is not None:
        label.configure(font=font)

window = tk.Tk()
label = ttk.Label(window, text='Sample text rendered in the chosen font.')
label.pack(padx=10, pady=10)
font_selection = FontSelectFrame(window, callback=update_preview)
font_selection.pack()
window.mainloop()
```

Example: askfont

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets.font import askfont
import tkinter as tk
from tkinter import ttk

```

(continues on next page)

(continued from previous page)

```
def font():
    res = askfont()
    if res[0] is not None:
        label.configure(font=res[0])
    print(res)

window = tk.Tk()
label = ttk.Label(window, text='Sample text rendered in the chosen font.')
label.pack(padx=10, pady=10)
ttk.Button(window, text="Pick a font", command=font).pack()
window.mainloop()
```

4.5 ttkwidgets.frames

Example: ScrolledFrame

```
# -*- coding: utf-8 -*-

# Copyright (c) Juliette Monsel 2018
# For license see LICENSE

from ttkwidgets.frames import ScrolledFrame
import tkinter as tk
from tkinter import ttk

window = tk.Tk()
frame = ScrolledFrame(window, compound=tk.RIGHT, canvasheight=200)
frame.pack(fill='both', expand=True)

for i in range(20):
    ttk.Label(frame.interior, text='Label %i' % i).pack()
window.mainloop()
```

Example: ToggledFrame

```
# -*- coding: utf-8 -*-

# Copyright (c) RedFantom 2017
# For license see LICENSE

from ttkwidgets.frames import ToggledFrame
import tkinter as tk
from tkinter import ttk

window = tk.Tk()
frame = ToggledFrame(window, text="Value", width=10)
frame.pack()
button = ttk.Button(frame.interior, text="Button", command=window.destroy)
button.grid()
frame.toggle()
window.mainloop()
```

Example: Tooltip

```
# -*- coding: utf-8 -*-

# Copyright (c) RedFantom 2017
# For license see LICENSE
from ttkwidgets.frames import Tooltip
import tkinter as tk


window = tk.Tk()
button = tk.Button(window, text="Button", command=window.destroy)
button.pack()
balloon = Tooltip(button)
window.mainloop()
```

5 Index

6 License

ttkwidgets: A collection of widgets for Tkinter's ttk extensions by various authors.

- Copyright (C) RedFantom 2017
- Copyright (C) The Python Team
- Copyright (C) Mitja Martini 2008
- Copyright (C) Russell Adams 2011
- Copyright (C) Juliette Monsel 2017

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

7 Contributing

If you have created a widget that you think is worth adding, then feel free to fork the repository and create a Pull Request when you've added the widget to your copy of the repository. You will be credited for your work, and you can add headers to your files. You will also be added to the AUTHORS.md file.

8 Issues

If you find any bugs or have any ideas, feel free to open an issue in the repository, and it will be looked at.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z