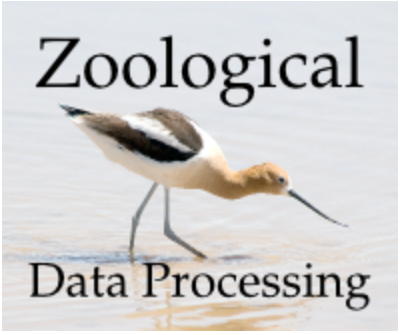
Tkinter 8.5 reference: a GUI for Python

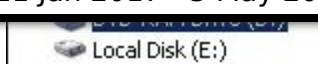


45. *ttk*.Treeview

The purpose of the *ttk*.Treeview widget is to present a hierarchical structure so that the user can use mouse actions to reveal or hide any part of the structure.

The association with the term “tree” is due to programming practice: tree structures are a commonplace in program design. Strictly speaking, the hierarchy shown in a Treeview widget is a forest: there is no one root, just a collection of top-level *nodes*, each of which may contain second-level nodes, each of which may contain third-level nodes, and so on.

You may have encountered this particular presentation as a way of browsing a directory or folder hierarchy. The entire hierarchy is displayed like an indented outline, where each directory is on a separate line, and the subdirectories of each directory are displayed underneath that line, indented:



The user can click on the icon for a directory to *collapse* (close) it, hiding all of the items in it. They can also click again on the icon to *expand* (open) it, so that the items in the directory or folder are shown.

The `Treeview` widget generalizes this concept so that you can use it to display any hierarchical structure, and the reader can collapse or expand subtrees of this structure with the mouse.

First, some definitions:

item

> One of the entities being displayed in the widget. For a file browser, an item might be either a directory or a file.
>
> Each item is associated with a textual label, and may also be associated with an image.

iid

> Every item in the tree has a unique identifier string called the *iid*. You can supply the iid values yourself, or you can let *ttk* generate them.

child

> The items directly below a given item in a hierarchy. A directory, for example, may have two kinds of children: files and subdirectories.

parent

ancestor

The ancestors of an item include its parent, its parent's parent, and so on up to the top level of the tree.

visible

Top-level items are always visible. Otherwise, an item is visible only if all its ancestors are expanded.

descendant

The descendants of an item include its children, its childrens' children, and so on. Another way of saying this is that the subtree of an item includes all its descendants.

tag

Your program can associate one or more *tag* strings with each item. You can use these tags to control the appearance of an item. For example, you could tag directories with the tag `'d'` and files with the tag `'f'`, and then specify that items with tag `'d'` use a boldface font.

You may also associate events with tags, so that certain events will cause certain handlers to be called for all items that have that tag. For example, you could set up a file browser so that when a user clicks on a directory, the browser updated its contents to reflect the current file structure.

Your `Treeview` widget will be structured with multiple columns. The first column, which we'll call the *icon column*, displays the icons that collapse or expand items. In the remaining columns,

columns, with the directory icons in the first column and the directory or file name in the second columns. Or you might wish to display file sizes, permissions, and other related data in additional columns.

The operations of the `Treeview` widget even allow you to use it as a tree editor. Your program can remove an entire subtree from its location in the main tree and then attach it later at an entirely different point.

Here is the general procedure for setting up a `Treeview` widget.

1. Create the widget with the *ttk*`.Treeview` constructor. Use the `columns` keyword argument to specify the number of columns to be displayed and to assign symbolic names to each column.

2. Use the `.column()` and `.heading()` methods to set up column headings (if you want them) and configure column properties such as size and stretchability.

3. Starting with the top-level entries, use the `.insert()` method to populate the tree. Each call to this method adds one item to the tree. Use the `open` keyword argument of this method to specify whether the item is initially expanded or collapsed.

   If you want to supply the iid value for this item, use the `iid` keyword argument. If you omit this argument, *ttk* will make one up and return it as the result of the `.insert()` method call.

   Use the `values` keyword argument of this method to specify what should appear in each column of this item

```
w = ttk.Treeview(parent, option=value, ...)
```

The constructor returns the new `Treeview` widget. Its options include:

`class_`	You may provide a widget class name when you create this widget. This name may be used to customize the widget's appearance; see Section 27, “Standardizing appearance”. Once the widget is created, the widget class name cannot be changed.
`columns`	A sequence of column identifier strings. These strings are used internally to identify the columns within the widget. The icon column, whose identifier is always `'#0'`, contains the collapse/expand icons and is always the first column. The columns you specify with the `columns` argument are in addition to the icon column. For example, if you specified `columns=('Name', 'Size')`, three columns would appear in the widget: first the icon column, then two more columns whose internal identifiers are `'Name'` and `'Size'`.
`cursor`	Use this option to specify the appearance of the mouse cursor when it is over the widget; see Section 5.8, “Cursors”. The default value (an empty string) specifies that the cursor is inherited from the parent widget.

- `'#all'` to select all columns and display them in the order defined by the `columns` argument.
- A list of column numbers (integer positions, counting from 0) or column identifiers from the `columns` argument.

  For example, suppose you specify `columns=('Name', 'Size', 'Date')`. This means each call to the `.insert()` method will require an argument `values=(`*`name`*`,` *`size`*`,` *`date`*`)` to supply the values that will be displayed. Let's call this sequence the *logical column sequence*.

  Further suppose that in the constructor you specify `columns=(2,0)`. The *physical column sequence*, the columns that will actually appear in the widget, will be three: the icon column will be first, followed by the date column (index 2 in the logical column sequence), followed by the name column (logical column index 0). The size column will not appear.

  You could get the same effect by specifying column identifiers instead of logical column positions: `columns=('Date', 'Name')`.

<table>
<tr><td></td><td>around the contents inside the widget. You may provide either a single dimension or a sequence of up to four dimensions, interpreted according to this table:
<table>
<tr><th>Values given</th><th>Left</th><th>Top</th><th>Right</th><th>Bottom</th></tr>
<tr><td>a</td><td>a</td><td>a</td><td>a</td><td>a</td></tr>
<tr><td>a b</td><td>a</td><td>b</td><td>a</td><td>b</td></tr>
<tr><td>a b c</td><td>a</td><td>c</td><td>b</td><td>c</td></tr>
<tr><td>a b c d</td><td>a</td><td>b</td><td>c</td><td>d</td></tr>
</table></td></tr>
<tr><td>selectmode</td><td>This option controls what the user is allowed to select with the mouse. Values can be:
<table>
<tr><td>selectmode='browse'</td><td>The user may select only one item at a time.</td></tr>
<tr><td>selectmode='extended'</td><td>The user may select multiple items at once.</td></tr>
<tr><td>selectmode='none'</td><td>The user cannot select items with the mouse.</td></tr>
</table></td></tr>
<tr><td>show</td><td>To suppress the labels at the top of each column, specify show='tree'. The default is to show the column labels.</td></tr>
</table>

`takefocus`	Use this option to specify whether a widget is visited during focus traversal; see Section 53, "Focus: routing keyboard input". Specify `takefocus=True` if you want the visit to accept focus; specify `takefocus=False` if the widget is not to accept focus. The default value is an empty string; by default, *`ttk`*`.Treeview` widgets do get focus.

Here are the methods available on a `Treeview` widget.

`.bbox(`*`item`*`, column=None)`

For the item with iid *`item`*, if the item is currently visible, this method returns a tuple `(`*`x`*`, `*`y`*`, `*`w`*`, `*`h`*`)`, where `(`*`x`*`, `*`y`*`)` are the coordinates of the upper left corner of that item relative to the widget, and *`w`* and *`h`* are the width and height of the item in pixels. If the item is not visible, the method returns an empty string.

If the optional `column` argument is omitted, you get the bounding box of the entire row. To get the bounding box of one specific column of the item's row, use `column=`*`C`* where *`C`* is either the integer index of the column or its column identifier.

`.column(`*`cid`*`, option=None, **`*`kw`*`)`

This method configures the appearance of the logical column specified by *`cid`*, which may be either a column index or a column identifier. To configure the icon column, use a *`cid`* value of `'#0'`.

`anchor`	The anchor that specifies where to position the content of the column. The default value is `'w'`.
`id`	The column name. This option is read-only and set when the constructor is called.
`minwidth`	Minimum width of the column in pixels; the default value is 20.
`stretch`	If this option is `True`, the column's width will be adjusted when the widget is resized. The default setting is 1.
`width`	Initial width of the column in pixels; the default is 200.

- If no *`option`* value or any other keyword argument is supplied, the method returns a dictionary of the column options for the specified column.
- To interrogate the current value of an option named *`X`* , use an argument `option=`*`X`*.
- To set one or more column options, you may pass keyword arguments using the option names shown above, e.g., `anchor=tk.CENTER` to center the column contents.

`.delete(*`*`items`*`)`

The arguments are iid values. All the items in the widget that have matching iid values are destroyed, along with all their descendants.

`.detach(*`*`items`*`)`

The items are not destroyed. You may reattach them to the visible tree using the `.move()` method described below.

`.exists(`*`iid`*`)`

Returns `True` if there exists an item in the widget with the given *`iid`*, or `False` otherwise. If an item is not currently visible because it was removed with the `.detach()` method, it is still considered to exist for the purposes of the `.exists()` method.

`.focus([`*`iid`*`])`

If you don't provide an argument to this method, you get back either the iid of the item that currently has focus, or `''` if no item has focus.

You can give focus to an item by passing its iid as the argument to this method.

`.get_children([`*`item`*`])`

Returns a tuple of the iid values of the children of the item specified by the *`item`* argument. If the argument is omitted, you get a tuple containing the iid values of the top-level items.

`.heading(`*`cid`*`, option=None, **`*`kw`*`)`

Use this method to configure the column heading that appears at the top of the widget for the column specified by *`cid`*, which may be either a column index or a column identifier. Use a *`cid`* argument value of `'#0'` to configure the heading over the icon column.

`anchor`	An anchor that specifies how the heading is aligned within the column; see Section 5.5, “Anchors”. The default value is `tk.W`.
`command`	A procedure to be called when the user clicks on this column heading.
`image`	To present a graphic in the column heading (either with or instead of a text heading), set this option to an image, as specified in Section 5.9, “Images”.
`text`	The text that you want to appear in the column heading.

- If you supply no keyword arguments, the method will return a dictionary showing the current settings of the column heading options.
- To interrogate the current value of some heading option *X*, use an argument of the form `option=`*X*; the method will return the current value of that option.
- You can set one or more heading options by supplying them as keyword arguments such as “`anchor=tk.CENTER`”.

`.identify_column(`*x*`)`

Given an *x* coordinate, this method returns a string of the form `'#`*n*`'` that identifies the column that contains that *x* coordinate.

Assuming that the icon column is displayed, the value of *n* is 0 for the icon column; 1 for the second physical column;

using the `displaycolumns` argument to the `Treeview` constructor.

If the icon column is not displayed, the value of *n* is 1 for the first physical column, 2 for the second, and so on.

`.identify_element(x, y)`

Returns the name of the element at location `(x, y)` relative to the widget, or `''` if no element appears at that position. Element names are discussed in Section 50, "The *ttk* element layer".

`.identify_region(x, y))`

Given the coordinates of a point relative to the widget, this method returns a string indicating what part of the widget contains that point. Return values may include:

`'nothing'`	The point is not within a functional part of the widget.
`'heading'`	The point is within one of the column headings.
`'separator'`	The point is located within the column headings row, but on the separator between columns. Use the `.identify_column()` method to determine which column is located just to the left of this separator.
`'tree'`	The point is located within the icon column.
`'cell'`	The point is located within an item row but not within the icon column.

returns the iid of that item. If that vertical coordinate is not within an item, this method returns an empty string.

.index(*iid*)

This method returns the index of the item with the specified *iid* relative to its parent, counting from zero.

.set_children(*item*, **newChildren*)

Use this method to change the set of children of the item whose iid is *item*. The *newChildren* argument is a sequence of iid strings. Any current children of *item* that are not in *newChildren* are removed.

.insert(*parent*, *index*, iid=None, **kw)

This method adds a new item to the tree, and returns the item's iid value. Arguments:

parent	To insert a new top-level item, make this argument an empty string. To insert a new item as a child of an existing item, make this argument the parent item's iid.
index	This argument specifies the position among this parent's children where you want the new item to be added. For example, to insert the item as the new first child, use a value of zero; to insert it after the parent's first child, use a value of 1; and so on. To add the new item as the last child of the parent, make this argument's value `'end'`.
iid	You may supply an iid for the item as a string

You may also specify a number of item options as keyword arguments to this method.

`image`	You may display an image just to the right of the icon for this item's row by providing an `image=`*`I`* argument, where *`I`* is an image as specified in Section 5.9, “Images”.
`open`	This option specifies whether this item will be open initially. If you supply `open=False`, this item will be closed. If you supply `open=True`, the item's children will be visible whenever the item itself is visible. The default value is `False`.
`tags`	You may supply one or more tag strings to be associated with this item. The value may be either a single string or a sequence of strings.
`text`	You may supply text to be displayed within the icon column of this item. If given, this text will appear just to the right of the icon, and also to the right of the image if provided.
`values`	This argument supplies the data items to be displayed in each column of the item. The values are supplied in logical column order. If too few values are supplied, the remaining columns will be blank in this item; if too many values are supplied, the extras will be discarded.

`.item(`*`iid`*`[, `*`option`*`[, **`*`kw`*`]])`

Use this method to set or retrieve the options within the item specified by *`iid`*. Refer to the `.insert()` method above

the option names and the corresponding values are the settings of those options. To retrieve the value of a given option, pass the option's name as its second argument. To set one or more options, pass them as keyword arguments to the method.

.move(*iid*, *parent*, *index*)

Move the item specified by *iid* to the values under the item specified by *parent* at position *index*. The *parent* and *index* arguments work the same as those arguments to the `.index()` method.

.next(*iid*)

If the item specified by *iid* is not the last child of its parent, this method returns the iid of the following child; if it is the last child of its parent, this method returns an empty string. If the specified item is a top-level item, the method returns the iid of the next top-level item, or an empty string if the specified item is the last top-level item.

.parent(*iid*)

If the item specified by *iid* is a top-level item, this method returns an empty string; otherwise it returns the iid of that item's parent.

.prev(*iid*)

If the item specified by *iid* is not the first child of its parent, this method returns the iid of the previous child; otherwise it returns an empty string. If the specified item is a top-level item, this method returns the iid of the previous top-level item, or an empty string if it is the first

This method ensures that the item specified by *iid* is visible. Any of its ancestors that are closed are opened. The widget is scrolled, if necessary, so that the item appears.

.selection_add(*items*)

In addition to any items already selected, add the specified *items*. The argument may be either a single iid or a sequence of iids.

.selection_remove(*items*)

Unselect any items specified by the argument, which may be a single iid or a sequence of iids.

.selection_set(*items*)

Only the specified *items* will be selected; if any other items were selected before, they will become unselected.

.selection_toggle(*items*)

The argument may be a single iid or a sequence of iids. For each item specified by the argument, if it was selected, unselect it; if it was unselected, select it.

.set(*iid*, column=None, value=None)

Use this method to retrieve or set the column values of the item specified by *iid*. With one argument, the method returns a dictionary: the keys are the column identifiers, and each related value is the text in the corresponding column.

the item's value for the specified column is set to the third argument.

.tag_bind(*tagName*, sequence=None, callback=None)

This method binds the event handler specified by the `callback` argument to all items that have tag *tagName*. The `sequence` and `callback` arguments work the same as the `sequence` and `func` arguments of the `.bind()` method described in Section 26, "Universal widget methods".

.tag_configure(*tagName*, option=None, ***kw*)

This method can either interrogate or set options that affect the appearance of all the items that have tag `tagName`. Tag options include:

`'background'`	The background color.
`'font'`	The text font.
`'foreground'`	The foreground color.
`'image'`	An image to be displayed in items with the given tag.

When called with one argument, it returns a dictionary of the current tag options. To return the value of a specific option *X*, use *X* as the second argument.

To set one or more options, use keyword arguments such as `foreground='red'`.

.tag_has(*tagName*[, *iid*])

True if the item with that iid has tag *tagName*, `False` otherwise.

.xview(*args)

This is the usual method for connecting a horizontal scrollbar to a scrollable widget. For details, see Section 22.1, "The `Scrollbar` *command* callback".

.yview(*args)

This is the usual method for connecting a vertical scrollbar to a scrollable widget. For details, see Section 22.1, "The `Scrollbar` *command* callback".

45.1. Virtual events for the *ttk*.`Treeview` widget

Certain state changes within a `Treeview` widget generate virtual events that you can use to respond to these changes; see Section 54.8, "Virtual events".

- Whenever there is a change in the selection, either by items becoming selected or becoming unselected, the widget generates a "`<<TreeviewSelect>>`" event.
- Whenever an item is opened, the widget generates a "`<<TreeviewOpen>>`" event.
- Whenever an item is closed, the widget generates a "`<<TreeviewClose>>`" event.

Next: 46. Methods common to all *ttk* widgets
Contents: *Tkinter* 8.5 reference: a GUI for Python

John W. Shipman
Comments welcome: john@nmt.edu
Last updated: 2013-09-09 22:58
URL: http://www.nmt.edu/~shipman/soft/tkinter/web/ttk-Treeview.html