Remi Documentation

Release 1.0.0

Davide Rosa

Dec 15, 2018

Contents

Contents:

CHAPTER 1

remi package

1.1 Submodules

1.2 remi.gui module

class remi.gui.**BODY**(**args*, ***kwargs*)

Bases: *remi.gui.Widget*

EVENT_ONERROR = 'onerror'

EVENT_ONLOAD = 'onload'

EVENT_ONONLINE = 'ononline'

EVENT_ONPAGEHIDE = 'onpagehide'

EVENT_ONPAGESHOW = 'onpageshow'

EVENT_ONRESIZE = 'onresize'

onerror(*message*, *source*, *lineno*, *colno*)

Called when an error occurs.

onload()

Called when page gets loaded.

ononline()

onpagehide()

onpageshow()

onresize(*width*, *height*)

class remi.gui.**Button**(*text=''*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

The Button widget. Have to be used in conjunction with its event onclick. Use Widget.onclick.connect in order to register the listener.

class remi.gui.**CheckBox**(*checked=False*, *user_data=''*, ***kwargs*)
Bases: *remi.gui.Input*

check box widget useful as numeric input field implements the onchange event.

get_value()

Returns

Return type bool

onchange(*value*)

set_value(*checked*, *update_ui=1*)

class remi.gui.**CheckBoxLabel**(*label=''*, *checked=False*, *user_data=''*, ***kwargs*)
Bases: *remi.gui.Widget*

onchange(*widget*, *value*)

set_on_change_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**ClassEventConnector**(*event_source_instance*, *event_name*, *event_method_bound*)
Bases: object

This class allows to manage the events. Decorating a method with *decorate_event* decorator The method gets the __is_event flag. At runtime, the methods that has this flag gets replaced by a ClassEventConnector. This class overloads the __call__ method, where the event method is called, and after that the listener method is called too.

connect(*callback*, **userdata*)
The callback and userdata gets stored, and if there is some javascript to add the js code is appended as attribute for the event source

class remi.gui.**ColorPicker**(*default_value='#995500'*, ***kwargs*)
Bases: *remi.gui.Input*

class remi.gui.**Date**(*default_value='2015-04-13'*, ***kwargs*)
Bases: *remi.gui.Input*

class remi.gui.**DropDown**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

Drop down selection widget. Implements the onchange(value) event. Register a listener for its selection change by means of the function DropDown.onchange.connect.

append(*value*, *key=''*)
Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

empty()
remove all children from the widget

get_item()

Returns The selected item or None.

Return type *DropDownItem*

get_key()

Returns The unique string identifier of the selected item or None.

Return type str

get_value()

Returns The value of the selected item or None.

Return type str

classmethod new_from_list(*items*, ***kwargs*)

onchange(*value*)
Called when a new DropDownItem gets selected.

select_by_key(*key*)
Selects an item by its unique string identifier.

Parameters **key** (*str*) – Unique string identifier of the DropDownItem that have to be selected.

select_by_value(*value*)
Selects a DropDownItem by means of the contained text-

Parameters **value** (*str*) – Textual content of the DropDownItem that have to be selected.

set_on_change_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_value(*value*)

class remi.gui.**DropDownItem**(*text*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

item widget for the DropDown

get_value()

set_value(*text*)

class remi.gui.**EventSource**(**args*, ***kwargs*)
Bases: object

setup_event_methods()

class remi.gui.**FileDownloader**(*text*, *filename*, *path_separator='/'*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

FileDownloader widget. Allows to start a file download.

download()

class remi.gui.**FileFolderItem**(*text*, *is_folder=False*, ***kwargs*)
Bases: *remi.gui.Widget*

FileFolderItem widget for the FileFolderNavigator

get_text()

onclick(*widget*)
Called when the Widget gets clicked by the user with the left mouse button.

onselection(*widget*)

set_on_click_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_selection_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_selected(*selected*)

set_text(*t*)

class remi.gui.**FileFolderNavigator**(*multiple_selection*, *selection_folder*, *allow_file_selection*, *allow_folder_selection*, ***kwargs*)
Bases: *remi.gui.Widget*

FileFolderNavigator widget.

chdir(*directory*)

dir_go(*widget*)

dir_go_back(*widget*)

get_selected_filefolders()

get_selection_list()

on_folder_item_click(*folderitem*)

on_folder_item_selected(*folderitem*)

populate_folder_items(*directory*)

class remi.gui.**FileSelectionDialog**(*title='File dialog'*, *message='Select files and folders'*, *multiple_selection=True*, *selection_folder='.'*, *allow_file_selection=True*, *allow_folder_selection=True*, ***kwargs*)
Bases: *remi.gui.GenericDialog*

file selection dialog, it opens a new webpage allows the OK/CANCEL functionality implementing the "confirm_value" and "cancel_dialog" events.

confirm_value(*widget*)
event called pressing on OK button. propagates the string content of the input field

set_on_confirm_value_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**FileUploader**(*savepath='./'*, *multiple_selection_allowed=False*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

FileUploader widget: allows to upload multiple files to a specified folder. implements the onsuccess and onfailed events.

ondata(*filedata*, *filename*)

onfailed(*filename*)

onsuccess(*filename*)

set_on_data_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_failed_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_success_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**GenericDialog**(*title=''*, *message=''*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

Generic Dialog widget. It can be customized to create personalized dialog windows. You can setup the content adding content widgets with the functions add_field or add_field_with_label. The user can confirm or dismiss the dialog with the common buttons Cancel/Ok. Each field added to the dialog can be retrieved by its key, in order to get back the edited value. Use the function

get_field(key) to retrieve the field.

The Ok button emits the 'confirm_dialog' event. Register the listener to it with set_on_confirm_dialog_listener. The Cancel button emits the 'cancel_dialog' event. Register the listener to it with set_on_cancel_dialog_listener.

add_field(*key*, *field*)
Adds a field to the dialog with a unique identifier.

Note: You can access to the fields content calling the function GenericDialog.get_field(key).

Parameters

- **key** (*str*) – The unique identifier for the field.
- **field** (*Widget*) – The widget to be added to the dialog, TextInput or any Widget for example.

add_field_with_label(*key*, *label_description*, *field*)
Adds a field to the dialog together with a descriptive label and a unique identifier.

Note: You can access to the fields content calling the function GenericDialog.get_field(key).

Parameters

- **key** (*str*) – The unique identifier for the field.
- **label_description** (*str*) – The string content of the description label.

- **field** (*Widget*) – The instance of the field Widget. It can be for example a TextInput or maybe
- **custom widget.** (*a*) –

cancel_dialog(*emitter*)
Event generated by the Cancel button click.

confirm_dialog(*emitter*)
Event generated by the OK button click.

get_field(*key*)

Parameters **key** (*str*) – The unique string identifier of the required field.

Returns Widget field instance added previously with methods GenericDialog.add_field or GenericDialog.add_field_with_label.

hide()

set_on_cancel_dialog_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, **user-data)

set_on_confirm_dialog_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, **user-data)

show(*base_app_instance*)

class remi.gui.**GenericObject**(*filename*, ***kwargs*)
Bases: *remi.gui.Widget*

GenericObject widget - allows to show embedded object like pdf,swf..

class remi.gui.**GridBox**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

It contains widgets automatically aligning them to the grid. Does not permit children absolute positioning.

In order to add children to this container, use the append(child, key) function. The key have to be string and determines the children positioning in the layout.

Note: If you would absolute positioning, use the Widget container instead.

append(*value*, *key=''*)
Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param. The key have to correspond to a an element provided in the 'define_grid' method param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

define_grid(*matrix*)
Populates the Table with a list of tuples of strings.

Parameters **matrix** (*list*) – list of iterables of strings (lists or something else). Items in the matrix have to correspond to a key for the children.

remove_child(*child*)
Removes a child instance from the Tag's children.

Parameters **child** (Tag) – The child to be removed.

set_column_gap(*value*)
Sets the gap value between columns

Parameters **value** (*int or str*) – gap value (i.e. 10 or "10px")

set_column_sizes(*values*)
Sets the size value for each column

Parameters **values** (*iterable of int or str*) – values are treated as percentage.

set_row_gap(*value*)
Sets the gap value between rows

Parameters **value** (*int or str*) – gap value (i.e. 10 or "10px")

set_row_sizes(*values*)
Sets the size value for each row

Parameters **values** (*iterable of int or str*) – values are treated as percentage.

class remi.gui.**HBox**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

The purpose of this widget is to automatically horizontally aligning the widgets that are appended to it.

Does not permit children absolute positioning.

In order to add children to this container, use the append(child, key) function. The key have to be numeric and determines the children order in the layout.

Note: If you would absolute positioning, use the Widget container instead.

append(*value*, *key=''*)
It allows to add child widgets to this. The key allows to access the specific child in this way widget.children[key]. The key have to be numeric and determines the children order in the layout.

Parameters

- **value** (Widget) – Child instance to be appended.
- **key** (*str*) – Unique identifier for the child. If key.isdigit()==True '0' '1'.. the value determines the order
- **the layout** (*in*) –

class remi.gui.**HEAD**(*title*, **args*, ***kwargs*)
Bases: *remi.gui.Tag*

repr(*changed_widgets=None*)
It is used to automatically represent the object to HTML format packs all the attributes, children and so on.

Parameters **changed_widgets** (*dict*) – A dictionary containing a collection of tags that have to be updated. The tag that have to be updated is the key, and the value is its textual repr.

set_internal_js(*net_interface_ip*, *pending_messages_queue_length*, *websocket_timeout_timer_ms*)

set_title(*title*)

class remi.gui.**HTML**(**args*, ***kwargs*)
Bases: *remi.gui.Tag*

repr(*changed_widgets=None*)
It is used to automatically represent the object to HTML format packs all the attributes, children and so on.

Parameters **changed_widgets** (*dict*) – A dictionary containing a collection of tags that have to be updated. The tag that have to be updated is the key, and the value is its textual repr.

class remi.gui.**Image**(*filename*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

image widget.

set_image(*filename*)

Parameters **filename** (*str*) – an url to an image

class remi.gui.**Input**(*input_type=''*, *default_value=''*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

get_value()
returns the new text value.

onchange(*value*)

set_on_change_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_read_only(*readonly*)

set_value(*value*)

class remi.gui.**InputDialog**(*title='Title'*, *message='Message'*, *initial_value=''*, **args*, ***kwargs*)
Bases: *remi.gui.GenericDialog*

Input Dialog widget. It can be used to query simple and short textual input to the user. The user can confirm or dismiss the dialog with the common buttons Cancel/Ok. The Ok button click or the ENTER key pression emits the 'confirm_dialog' event. Register the listener to it with set_on_confirm_dialog_listener. The Cancel button emits the 'cancel_dialog' event. Register the listener to it with set_on_cancel_dialog_listener.

confirm_value(*widget*)
Event called pressing on OK button.

on_keydown_listener(*widget*, *value*, *keycode*)
event called pressing on ENTER key.

propagates the string content of the input field

set_on_confirm_value_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**Label**(*text*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

Non editable text label widget. Set its content by means of set_text function, and retrieve its content with the function get_text.

class remi.gui.**Link**(*url*, *text*, *open_new_window=True*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

get_url()

class remi.gui.**ListItem**(*text*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

List item widget for the ListView.

ListItems are characterized by a textual content. They can be selected from the ListView. Do NOT manage directly its selection by registering set_on_click_listener, use instead the events of the ListView.

get_value()

Returns The text content of the ListItem

Return type str

class remi.gui.**ListView**(*selectable=True*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

List widget it can contain ListItems. Add items to it by using the standard append(item, key) function or generate a filled list from a string list by means of the function new_from_list. Use the list in conjunction of its onselection event. Register a listener with ListView.onselection.connect.

append(*value*, *key=''*)
Appends child items to the ListView. The items are accessible by list.children[key].

Parameters

- **value** (*ListItem, or iterable of ListItems*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

empty()
Removes all children from the list

get_item()

Returns The selected item or None

Return type *ListItem*

get_key()

Returns The key of the selected item or None if no item is selected.

Return type str

get_value()

Returns The value of the selected item or None

Return type str

classmethod new_from_list(*items*, ***kwargs*)
Populates the ListView with a string list.

Parameters **items** (*list*) – list of strings to fill the widget with.

onselection(*widget*)
Called when a new item gets selected in the list.

select_by_key(*key*)
Selects an item by its key.

Parameters **key** (*str*) – The unique string identifier of the item that have to be selected.

select_by_value(*value*)
Selects an item by the text content of the child.

Parameters **value** (*str*) – Text content of the item that have to be selected.

set_on_selection_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_value(*value*)

class remi.gui.**Menu**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

Menu widget can contain MenuItem.

class remi.gui.**MenuBar**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

class remi.gui.**MenuItem**(*text*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

MenuItem widget can contain other MenuItem.

append(*value*, *key=''*)
Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

class remi.gui.**Progress**(*value=0*, *_max=100*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

Progress bar widget.

set_max(*_max*)

Parameters **max** (*int*) – The maximum progress value.

set_value(*value*)

Parameters **value** (*int*) – The actual progress value.

class remi.gui.**Slider**(*default_value=''*, *min=0*, *max=10000*, *step=1*, ***kwargs*)
Bases: *remi.gui.Input*

oninput(*value*)

set_oninput_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**SpinBox**(*default_value=100*, *min_value=100*, *max_value=5000*, *step=1*, *allow_editing=True*, ***kwargs*)
Bases: *remi.gui.Input*

spin box widget useful as numeric input field implements the onchange event.

class remi.gui.**Svg**(*width*, *height*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

svg widget - is a container for graphic widgets such as SvgCircle, SvgLine and so on.

set_viewbox(*x*, *y*, *w*, *h*)
Sets the origin and size of the viewbox, describing a virtual view area.

Parameters

- **x** (*int*) – x coordinate of the viewbox origin
- **y** (*int*) – y coordinate of the viewbox origin
- **w** (*int*) – width of the viewbox
- **h** (*int*) – height of the viewbox

class remi.gui.**SvgCircle**(*x*, *y*, *radius*, **args*, ***kwargs*)
Bases: *remi.gui.SvgShape*

svg circle - a circle represented filled and with a stroke.

set_position(*x*, *y*)
Sets the circle position.

Parameters

- **x** (*int*) – the x coordinate
- **y** (*int*) – the y coordinate

set_radius(*radius*)
Sets the circle radius.

Parameters **radius** (*int*) – the circle radius

class remi.gui.**SvgGroup**(*x*, *y*, **args*, ***kwargs*)
Bases: *remi.gui.SvgShape*

svg group widget.

class remi.gui.**SvgLine**(*x1*, *y1*, *x2*, *y2*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

set_coords(*x1*, *y1*, *x2*, *y2*)

set_p1(*x1*, *y1*)

set_p2(*x2*, *y2*)

set_stroke(*width=1*, *color='black'*)

class remi.gui.**SvgPath**(*path_value*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

add_arc(*x*, *y*, *rx*, *ry*, *x_axis_rotation*, *large_arc_flag*, *sweep_flag*)

add_position(*x*, *y*)

set_fill(*color='black'*)
Sets the fill color.

Parameters **color** (*str*) – stroke color

set_stroke(*width=1*, *color='black'*)
Sets the stroke properties.

Parameters

- **width** (*int*) – stroke width
- **color** (*str*) – stroke color

class remi.gui.**SvgPolyline**(*_maxlen=None*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

add_coord(*x*, *y*)

set_stroke(*width=1*, *color='black'*)

class remi.gui.**SvgRectangle**(*x*, *y*, *w*, *h*, **args*, ***kwargs*)
Bases: *remi.gui.SvgShape*

svg rectangle - a rectangle represented filled and with a stroke.

set_size(*w*, *h*)
Sets the rectangle size.

Parameters

- **w** (*int*) – width of the rectangle
- **h** (*int*) – height of the rectangle

class remi.gui.**SvgShape**(*x*, *y*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

svg shape generic widget. Consists of a position, a fill color and a stroke.

set_fill(*color='black'*)
Sets the fill color.

Parameters **color** (*str*) – stroke color

set_position(*x*, *y*)
Sets the shape position.

Parameters

- **x** (*int*) – the x coordinate
- **y** (*int*) – the y coordinate

set_stroke(*width=1*, *color='black'*)
Sets the stroke properties.

Parameters

- **width** (*int*) – stroke width
- **color** (*str*) – stroke color

class remi.gui.**SvgText**(*x*, *y*, *text*, **args*, ***kwargs*)
Bases: *remi.gui.SvgShape*, remi.gui._MixinTextualWidget

class remi.gui.**TabBox**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

add_tab(*widget*, *name*, *tab_cb*)

select_by_index(*index*)
shows a tab identified by its index

select_by_name(*name*)
shows a tab identified by the name

select_by_widget(*widget*)
shows a tab identified by the contained widget

class remi.gui.**Table**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

table widget - it will contains TableRow

append(*value*, *key=''*)
Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

append_from_list(*content*, *fill_title=False*)
Appends rows created from the data contained in the provided list of tuples of strings. The first tuple of the list can be set as table title.

Parameters

- **content** (*list*) – list of tuples of strings. Each tuple is a row.
- **fill_title** (*bool*) – if true, the first tuple in the list will be set as title.

classmethod new_from_list(*content*, *fill_title=True*, ***kwargs*)
Populates the Table with a list of tuples of strings.

Parameters

- **content** (*list*) – list of tuples of strings. Each tuple is a row.
- **fill_title** (*bool*) – if true, the first tuple in the list will be set as title

on_table_row_click(*row*, *item*)

set_on_table_row_click_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

class remi.gui.**TableEditableItem**(*text=''*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

item widget for the TableRow.

onchange(*emitter*, *new_value*)

set_on_change_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**TableItem**(*text=''*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

item widget for the TableRow.

class remi.gui.**TableRow**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

row widget for the Table - it will contains TableItem

append(*value*, *key=''*)
Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

on_row_item_click(*item*)
Event on item click.

Note: This is internally used by the Table widget in order to generate the Table.on_table_row_click event. Use Table.on_table_row_click instead.

Parameters

- **emitter** (TableRow) – The emitter of the event.
- **item** (TableItem) – The clicked TableItem.

set_on_row_item_click_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

class remi.gui.**TableTitle**(*text=''*, **args*, ***kwargs*)
Bases: *remi.gui.TableItem*, remi.gui._MixinTextualWidget

title widget for the table.

class remi.gui.**TableWidget**(*n_rows*, *n_columns*, *use_title=True*, *editable=False*, **args*, ***kwargs*)
Bases: *remi.gui.Table*

Basic table model widget. Each item is addressed by stringified integer key in the children dictionary.

column_count()
Returns table's columns count.

item_at(*row*, *column*)
Returns the TableItem instance at row, column cordinates

Parameters

- **row** (*int*) – zero based index
- **column** (*int*) – zero based index

item_coords(*table_item*)
Returns table_item's (row, column) cordinates. Returns None in case of item not found.

Parameters **table_item** (TableItem) – an item instance

on_item_changed(*item*, *new_value*, *row*, *column*)
Event for the item change.

Parameters

- **emitter** (TableWidget) – The emitter of the event.
- **item** (TableItem) – The TableItem instance.
- **new_value** (*str*) – New text content.
- **row** (*int*) – row index.
- **column** (*int*) – column index.

row_count()
Returns table's rows count (the title is considered as a row).

set_column_count(*count*)
Sets the table column count.

Parameters **count** (*int*) – column of rows

set_on_item_changed_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_row_count(*count*)
Sets the table row count.

Parameters **count** (*int*) – number of rows

set_use_title(*use_title*)
Returns the TableItem instance at row, column cordinates

Parameters **use_title** (*bool*) – enable title bar.

class remi.gui.**Tag**(*attributes=None*, *_type=''*, *_class=None*, ***kwargs*)
Bases: object

Tag is the base class of the framework. It represents an element that can be added to the GUI, but it is not necessarily graphically representable. You can use this class for sending javascript code to the clients.

add_child(*key*, *value*)
Adds a child to the Tag

To retrieve the child call get_child or access to the Tag.children[key] dictionary.

Parameters

- **key** (*str*) – Unique child's identifier, or iterable of keys
- **value** (Tag, *str*) – can be a Tag, an iterable of Tag or a str. In case of iterable of Tag is a dict, each item's key is set as 'key' param

add_class(*cls*)

disable_refresh()

empty()
remove all children from the widget

enable_refresh()

get_child(*key*)
Returns the child identified by 'key'

Parameters **key** (*str*) – Unique identifier of the child.

get_parent()
Returns the parent tag instance or None where not applicable

identifier

innerHTML(*local_changed_widgets*)

remove_child(*child*)
Removes a child instance from the Tag's children.

Parameters **child** (Tag) – The child to be removed.

remove_class(*cls*)

repr(*changed_widgets=None*)
It is used to automatically represent the object to HTML format packs all the attributes, children and so on.

Parameters **changed_widgets** (*dict*) – A dictionary containing a collection of tags that have to be updated. The tag that have to be updated is the key, and the value is its textual repr.

set_identifier(*new_identifier*)
Allows to set a unique id for the Tag.

Parameters **new_identifier** (*str*) – a unique id for the tag

class remi.gui.**TextInput**(*single_line=True*, *hint=''*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

Editable multiline/single_line text area widget. You can set the content by means of the function set_text or retrieve its content with get_text.

get_value()

Returns The text content of the TextInput. You can set the text content with set_text(text).

Return type str

onchange(*new_value*)
Called when the user changes the TextInput content. With single_line=True it fires in case of focus lost and Enter key pressed. With single_line=False it fires at each key released.

Parameters **new_value** (*str*) – the new string content of the TextInput.

onkeydown(*new_value*, *keycode*)
Called when the user types a key into the TextInput.

Note: This event can't be registered together with Widget.onchange.

Parameters

- **new_value** (*str*) – the new string content of the TextInput.
- **keycode** (*str*) – the numeric char code

onkeyup(*new_value*, *keycode*)
Called when user types and releases a key into the TextInput

Note: This event can't be registered together with Widget.onchange.

Parameters

- **new_value** (*str*) – the new string content of the TextInput
- **keycode** (*str*) – the numeric char code

set_on_change_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_key_down_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_key_up_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_value(*text*)
Sets the text content.

Parameters **text** (*str*) – The string content that have to be appended as standard child identified by the key 'text'

class remi.gui.**TreeItem**(*text*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*, remi.gui._MixinTextualWidget

TreeItem widget can contain other TreeItem.

append(*value*, *key=''*)
Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

onclick()
Called when the Widget gets clicked by the user with the left mouse button.

class remi.gui.**TreeView**(**args*, ***kwargs*)
Bases: *remi.gui.Widget*

TreeView widget can contain TreeItem.

class remi.gui.**VBox**(**args*, ***kwargs*)
Bases: *remi.gui.HBox*

The purpose of this widget is to automatically vertically aligning the widgets that are appended to it.

Does not permit children absolute positioning.

In order to add children to this container, use the append(child, key) function. The key have to be numeric and determines the children order in the layout.

Note: If you would absolute positioning, use the Widget container instead.

class remi.gui.**VideoPlayer**(*video*, *poster=None*, *autoplay=False*, *loop=False*, **args*, ***kwargs*)
Bases: *remi.gui.Widget*

onended()
Called when the media has been played and reached the end.

set_autoplay(*autoplay*)

set_loop(*loop*)
Sets the VideoPlayer to restart video when finished.

Note: If set as True the event onended will not fire.

set_on_ended_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, ***userdata)

class remi.gui.**Widget**(*children=None*, *style=None*, **args*, ***kwargs*)
Bases: *remi.gui.Tag*, *remi.gui.EventSource*

Base class for gui widgets.

Widget can be used as generic container. You can add children by the append(value, key) function. Widget can be arranged in absolute positioning (assigning style['top'] and style['left'] attributes to the children or in a simple auto-alignment. You can decide the horizontal or vertical arrangement by the function set_layout_orientation(layout_orientation) passing as parameter Widget.LAYOUT_HORIZONTAL or Widget.LAYOUT_VERTICAL.

Tips: In html, it is a DIV tag The self.type attribute specifies the HTML tag representation The self.attributes[] attribute specifies the HTML attributes like 'style' 'class' 'id' The self.style[] attribute specifies the CSS style content like 'font' 'color'. It will be packed together with 'self.attributes'.

EVENT_ONBLUR = 'onblur'

EVENT_ONCHANGE = 'onchange'

EVENT_ONCLICK = 'onclick'

EVENT_ONCONTEXTMENU = 'oncontextmenu'

EVENT_ONDBLCLICK = 'ondblclick'

EVENT_ONFOCUS = 'onfocus'

EVENT_ONINPUT = 'oninput'

EVENT_ONKEYDOWN = 'onkeydown'

EVENT_ONKEYPRESS = 'onkeypress'

EVENT_ONKEYUP = 'onkeyup'

EVENT_ONMOUSEDOWN = 'onmousedown'

EVENT_ONMOUSELEAVE = 'onmouseleave'

EVENT_ONMOUSEMOVE = 'onmousemove'

EVENT_ONMOUSEOUT = 'onmouseout'

EVENT_ONMOUSEOVER = 'onmouseover'

EVENT_ONMOUSEUP = 'onmouseup'

EVENT_ONTOUCHCANCEL = 'ontouchcancel'

EVENT_ONTOUCHEND = 'ontouchend'

EVENT_ONTOUCHENTER = 'ontouchenter'

EVENT_ONTOUCHLEAVE = 'ontouchleave'

EVENT_ONTOUCHMOVE = 'ontouchmove'

EVENT_ONTOUCHSTART = 'ontouchstart'

EVENT_ONUPDATE = 'onupdate'

LAYOUT_HORIZONTAL = True

LAYOUT_VERTICAL = False

append(*value*, *key=''*)

Adds a child widget, generating and returning a key if not provided

In order to access to the specific child in this way widget.children[key].

Parameters

- **value** (*Widget, or iterable of Widgets*) – The child to be appended. In case of a dictionary, each item's key is used as 'key' param for the single append.
- **key** (*str*) – The unique string identifier for the child. Ignored in case of iterable 'value' param.

Returns

a key used to refer to the child for all future interaction, or a list of keys in case of an iterable 'value' param

Return type str

onblur()

Called when the Widget loses focus

onclick()

Called when the Widget gets clicked by the user with the left mouse button.

oncontextmenu()

Called when the Widget gets clicked by the user with the right mouse button.

ondblclick()

Called when the Widget gets double clicked by the user with the left mouse button.

onfocus()
Called when the Widget gets focus.

onkeydown(*key*, *keycode*, *ctrl*, *shift*, *alt*)
Called when user types and releases a key. The widget should be able to receive the focus in order to emit the event. Assign a 'tabindex' attribute to make it focusable.

Parameters

- **key** (*str*) – the character value
- **keycode** (*str*) – the numeric char code

onkeyup(*key*, *keycode*, *ctrl*, *shift*, *alt*)
Called when user types and releases a key. The widget should be able to receive the focus in order to emit the event. Assign a 'tabindex' attribute to make it focusable.

Parameters

- **key** (*str*) – the character value
- **keycode** (*str*) – the numeric char code

onmousedown(*x*, *y*)
Called when the user presses left or right mouse button over a Widget.

Parameters

- **x** (*float*) – position of the mouse inside the widget
- **y** (*float*) – position of the mouse inside the widget

onmouseleave()
Called when the mouse cursor moves outside a Widget.

Note: This event is often used together with the Widget.onmouseenter event, which occurs when the mouse pointer is moved onto a Widget.

Note: The Widget.onmouseleave event is similar to the Widget.onmouseout event. The only difference is that the onmouseleave event does not bubble (does not propagate up the Widgets tree).

onmousemove(*x*, *y*)
Called when the mouse cursor moves inside the Widget.

Parameters

- **x** (*float*) – position of the mouse inside the widget
- **y** (*float*) – position of the mouse inside the widget

onmouseout()
Called when the mouse cursor moves outside a Widget.

Note: This event is often used together with the Widget.onmouseover event, which occurs when the pointer is moved onto a Widget, or onto one of its children.

onmouseup(*x*, *y*)
Called when the user releases left or right mouse button over a Widget.

Parameters

- **x** (*float*) – position of the mouse inside the widget
- **y** (*float*) – position of the mouse inside the widget

ontouchcancel()
Called when a touch point has been disrupted in an implementation-specific manner (for example, too many touch points are created).

ontouchend(*x*, *y*)
Called when a finger is released from the widget.

Parameters

- **x** (*float*) – position of the finger inside the widget
- **y** (*float*) – position of the finger inside the widget

ontouchenter(*x*, *y*)
Called when a finger touches from outside to inside the widget.

Parameters

- **x** (*float*) – position of the finger inside the widget
- **y** (*float*) – position of the finger inside the widget

ontouchleave()
Called when a finger touches from inside to outside the widget.

ontouchmove(*x*, *y*)
Called continuously while a finger is dragged across the screen, over a Widget.

Parameters

- **x** (*float*) – position of the finger inside the widget
- **y** (*float*) – position of the finger inside the widget

ontouchstart(*x*, *y*)
Called when a finger touches the widget.

Parameters

- **x** (*float*) – position of the finger inside the widget
- **y** (*float*) – position of the finger inside the widget

redraw()
Forces a graphic update of the widget

repr(*changed_widgets=None*)
Represents the widget as HTML format, packs all the attributes, children and so on.

Parameters

- **client** (App) – Client instance.
- **changed_widgets** (*dict*) – A dictionary containing a collection of widgets that have to be updated. The Widget that have to be updated is the key, and the value is its textual repr.

set_enabled(*enabled*)

set_layout_orientation(*layout_orientation*)
For the generic Widget, this function allows to setup the children arrangement.

Parameters **layout_orientation** (*Widget.LAYOUT_HORIZONTAL or Widget.LAYOUT_VERTICAL*) –

set_on_blur_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_click_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_contextmenu_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_dblclick_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_focus_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_key_down_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_key_up_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_mousedown_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_mouseleave_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_mousemove_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_mouseout_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_mouseup_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_touchcancel_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_touchend_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_touchenter_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *user-data)

set_on_touchleave_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_touchmove_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_on_touchstart_listener(*callback*, **userdata*)
Registers the listener For backward compatibility Suggested new dialect event.connect(callback, *userdata)

set_size(*width*, *height*)
Set the widget size.

Parameters

- **width** (*int or str*) – An optional width for the widget (es. width=10 or width='10px' or width='10%').
- **height** (*int or str*) – An optional height for the widget (es. height=10 or height='10px' or height='10%').

set_style(*style*)
Allows to set style properties for the widget. :param style: The style property dictionary or json string. :type style: str or dict

remi.gui.**decorate_constructor_parameter_types**(*type_list*)
Private decorator for use in the editor. Allows Editor to instantiate widgets.

Parameters **params** (*str*) – The list of types for the widget constructor method (i.e. "(int, int, str)")

remi.gui.**decorate_event**(*method*)
setup a method as an event

remi.gui.**decorate_event_js**(*js_code*)
setup a method as an event, adding also javascript code to generate

Parameters **js_code** (*str*) – javascript code to generate the event client-side. js_code is added to the widget html as widget.attributes['onclick'] = js_code%{'emitter_identifier':widget.identifier, 'event_name':'onclick'}

remi.gui.**decorate_explicit_alias_for_listener_registration**(*method*)

remi.gui.**decorate_set_on_listener**(*prototype*)
Private decorator for use in the editor. Allows the Editor to create listener methods.

Parameters **params** (*str*) – The list of parameters for the listener method (es. "(self, new_value)")

remi.gui.**from_pix**(*x*)

remi.gui.**jsonize**(*d*)

remi.gui.**to_pix**(*x*)

1.3 remi.server module

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class remi.server.**App**(*request*, *client_address*, *server*, ***app_args*)
Bases: BaseHTTPServer.BaseHTTPRequestHandler, object

This class will handles any incoming request from the browser The main application class can subclass this In the do_POST and do_GET methods it is expected to receive requests such as:

- function calls with parameters
- file requests

close()
Command to initiate an App to close

do_AUTHHEAD()

do_GET()

do_HEAD()

do_POST()

do_gui_update()
This method gets called also by Timer, a new thread, and so needs to lock the update

execute_javascript(*code*)

idle()
Idle function called every UPDATE_INTERVAL before the gui update. Useful to schedule tasks.

main(*_)
Subclasses of App class *must* declare a main function that will be the entry point of the application. Inside the main function you have to declare the GUI structure and return the root widget.

notification_message(*title*, *content*, *icon=''*)
This function sends "javascript" message to the client, that executes its content. In this particular code, a notification message is shown

on_close()
Called by the server when the App have to be terminated

onerror(*emitter*, *message*, *source*, *lineno*, *colno*)
WebPage Event that occurs on webpage errors

onload(*emitter*)
WebPage Event that occurs on webpage loaded

ononline(*emitter*)
WebPage Event that occurs on webpage goes online after a disconnection

onpagehide(*emitter*)
WebPage Event that occurs on webpage when the user navigates away

onpageshow(*emitter*)
WebPage Event that occurs on webpage gets shown

onresize(*emitter*, *width*, *height*)
WebPage Event that occurs on webpage gets resized

re_attr_call = <_sre.SRE_Pattern object at 0x22f1260>

re_static_file = <_sre.SRE_Pattern object>

set_root_widget(*widget*)

websocket_handshake_done(*ws_instance_to_update*)

class remi.server.**Server**(*gui_class*, *title=''*, *start=True*, *address='127.0.0.1'*, *port=8081*, *username=None*, *password=None*, *multiple_instance=False*, *enable_file_cache=True*, *update_interval=0.1*, *start_browser=True*, *websocket_timeout_timer_ms=1000*, *pending_messages_queue_length=1000*, *certfile=None*, *keyfile=None*, *ssl_version=None*, *userdata=()*)

Bases: object

address

serve_forever()

start()

stop()

title

class remi.server.**StandaloneServer**(*gui_class*, *title=''*, *width=800*, *height=600*, *resizable=True*, *fullscreen=False*, *start=True*, *userdata=()*)

Bases: *remi.server.Server*

serve_forever()

class remi.server.**ThreadedHTTPServer**(*server_address*, *RequestHandlerClass*, *auth*, *multiple_instance*, *enable_file_cache*, *update_interval*, *websocket_timeout_timer_ms*, *pending_messages_queue_length*, *title*, *server_starter_instance*, *certfile*, *keyfile*, *ssl_version*, **userdata*)

Bases: SocketServer.ThreadingMixIn, BaseHTTPServer.HTTPServer

daemon_threads = False

class remi.server.**WebSocketsHandler**(*headers*, **args*, ***kwargs*)

Bases: SocketServer.StreamRequestHandler

static bytetonum(*b*)

close()

handle()

handshake()

magic = '258EAFA5-E914-47DA-95CA-C5AB0DC85B11'

on_message(*message*)

read_next_message()

send_message(*message*)

setup()

remi.server.**encode_text**(*data*)

remi.server.**from_websocket**(*data*)

remi.server.**get_method_by_id**(*_id*)

remi.server.**get_method_by_name**(*root_node*, *name*)

remi.server.**parse_parametrs**(*p*)
Parses the parameters given from POST or websocket reqs expecting the parameters as: "11|par1='asd'|6|par2=1" returns a dict like {par1:'asd',par2:1}

remi.server.**parse_session_cookie**(*cookie_to_cook*)
cookie_to_cook = http_header['cookie']

remi.server.**start**(*main_gui_class*, ***kwargs*)
This method starts the webserver with a specific App subclass.

remi.server.**to_websocket**(*data*)

1.4 Module contents

CHAPTER 2

remi

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

V

W