Widget

class `openerp.web.Widget`()

This is the base class for all visual components. It corresponds to an MVC view. It provides a number of services to handle a section of a page:

- Rendering with QWeb
- Parenting-child relations
- Life-cycle management (including facilitating children destruction when a parent object is removed)
- DOM insertion, via jQuery-powered insertion methods. Insertion targets can be anything the corresponding jQuery method accepts (generally selectors, DOM objects):

  `appendTo()`
  Renders the widget and inserts it as the last child of the target, uses .appendTo()

  `prependTo()`
  Renders the widget and inserts it as the first child of the target, uses .prependTo()

  `insertAfter()`
  Renders the widget and inserts it as the preceding sibling of the target, uses .insertAfter()

  `insertBefore()`
  Renders the widget and inserts it as the following sibling of the target, uses .insertBefore()

- Backbone-compatible shortcuts

DOM Root

A **`Widget()`** is responsible for a section of the page materialized by the DOM root of the widget. The DOM root is available via the **`el`** and **`$el`** attributes, which are raw DOM Element and the jQuery wrapper around the DOM element.

There are two main ways to define and generate this DOM root:

`openerp.web.Widget.`**`template`**
Should be set to the name of a QWeb template (a **`String()`**). If set, the template will be rendered after the widget has been initialized but before it has been sta element generated by the template will be set as the DOM root of the widget.

`openerp.web.Widget.`**`tagName`**
Used if the widget has no template defined. Defaults to `div`, will be used as the tag name to create the DOM element to set as the widget's DOM root. It is possi customize this generated DOM root with the following attributes:

`openerp.web.Widget.`**`id`**
Used to generate an `id` attribute on the generated DOM root.

`openerp.web.Widget.`**`className`**
Used to generate a `class` attribute on the generated DOM root.

`openerp.web.Widget.`**`attributes`**
Mapping (object literal) of attribute names to attribute values. Each of these k:v pairs will be set as a DOM attribute on the generated DOM root.

None of these is used in case a template is specified on the widget.

The DOM root can also be defined programmatically by overridding

`openerp.web.Widget.`**`renderElement`**()
Renders the widget's DOM root and sets it. The default implementation will render a set template or generate an element as described above, and will call **`setE`** result.

Any override to **`renderElement()`** which does not call its `_super` **must** call **`setElement()`** with whatever it generated or the widget's behavior is undefined.

Note:
The default **`renderElement()`** can be called repeatedly, it will *replace* the previous DOM root (using `replaceWith`). However, this requires that the widget cor unsets its events (and children widgets). Generally, **`renderElement()`** should not be called repeatedly unless the widget advertizes this feature.

Accessing DOM content

Because a widget is only responsible for the content below its DOM root, there is a shortcut for selecting sub-sections of a widget's DOM:

`openerp.web.Widget.`**`$`**(*selector*)
Applies the CSS selector specified as parameter to the widget's DOM root.

```
this.$(selector);
```

is functionally identical to:

```
this.$el.find(selector);
```

Arguments: • **selector** (*String*) – CSS selector
Returns: jQuery object

Note:
this helper method is compatible with `Backbone.View.$`

Resetting the DOM root

`openerp.web.Widget.setElement`(*element*)

Re-sets the widget's DOM root to the provided element, also handles re-setting the various aliases of the DOM root as well as unsetting and re-setting delegate

Arguments: • **element** (*Element*) – a DOM element or jQuery object to set as the widget's DOM root

---

Note:
should be mostly compatible with Backbone's setElement

DOM events handling

A widget will generally need to respond to user action within its section of the page. This entails binding events to DOM elements.

To this end, **`Widget()`** provides an shortcut:

`openerp.web.Widget.events`

Events are a mapping of `event selector` (an event name and a CSS selector separated by a space) to a callback. The callback can be either a method name in function. In either case, the `this` will be set to the widget:

```
events: {
    'click p.oe_some_class a': 'some_method',
    'change input': function (e) {
        e.stopPropagation();
    }
},
```

The selector is used for jQuery's event delegation, the callback will only be triggered for descendants of the DOM root matching the selector [o]. If the selector i event name is specified), the event will be set directly on the widget's DOM root.

`openerp.web.Widget.delegateEvents`()

This method is in charge of binding **`events`** to the DOM. It is automatically called after setting the widget's DOM root.

It can be overridden to set up more complex events than the **`events`** map allows, but the parent should always be called (or **`events`** won't be handled correctly

`openerp.web.Widget.undelegateEvents`()

This method is in charge of unbinding **`events`** from the DOM root when the widget is destroyed or the DOM root is reset, in order to avoid leaving "phantom"

It should be overridden to un-set any event set in an override of **`delegateEvents()`**.

Note:

this behavior should be compatible with Backbone's delegateEvents, apart from not accepting any argument.

Subclassing Widget

`Widget()` is subclassed in the standard manner (via the **`extend()`** method), and provides a number of abstract properties and concrete methods (which you may o override). Creating a subclass looks like this:

```
var MyWidget = openerp.base.Widget.extend({
    // QWeb template to use when rendering the object
    template: "MyQWebTemplate",

    init: function(parent) {
        this._super(parent);
        // insert code to execute before rendering, for object
        // initialization
    },
    start: function() {
        this._super();
        // post-rendering initialization code, at this point
        // ``this.$element`` has been initialized
        this.$element.find(".my_button").click(/* an example of event binding * /);

        // if ``start`` is asynchronous, return a promise object so callers
```

```
        // know when the object is done initializing
        return this.rpc(/* … */)
    }
});
```

The new class can then be used in the following manner:

```
// Create the instance
var my_widget = new MyWidget(this);
// Render and insert into DOM
my_widget.appendTo(".some-div");
```

After these two lines have executed (and any promise returned by `appendTo` has been resolved if needed), the widget is ready to be used.

Note:

the insertion methods will start the widget themselves, and will return the result of **`start()`**.

If for some reason you do not want to call these methods, you will have to first call **`render()`** on the widget, then insert it into your DOM and start it.

If the widget is not needed anymore (because it's transient), simply terminate it:

```
my_widget.destroy();
```

will unbind all DOM events, remove the widget's content from the DOM and destroy all widget data.

[0]	not all DOM events are compatible with events delegation