Widgets v2 Instance API

The functions in this API can be used for the instance of a widget that you created using the **Widgets v2 Class API** either at runtime (execution of the code) or in your custom widget's **widget.js** file.

All the functions can be used either at runtime or in your custom widget's code as shown below:

At Runtime:

```
$$('customWidget1').addClass('mycssclass');
```

Code for your widget:

```
CustomWidget.prototype.init = function () {
    this.addClass('mycssclass');
};
```

Observable

These functions allow you to define or execute an event as well as subscribe to and unsubscribe from an event.

fire( )

void **fire**(String *event* [,String *target*] [,Object *data*] [,Object *options*])

Parameter	Type	Description
event	String	Event name
target	String	Target expressed in RegEx
data	Object	Optional data
options	Object	Event options

Description

fire( ) allows you to declare or fire an event.

options

The *options* parameter contains the following attributes:

Attribute	Description
deferred	timeout to launch the callback
animationFrame	launch the callback during an animation frame
once	discard previous pending events if true
onlyRealEvent	launch callback only when the fired event is the subscribed event

Example

In the following example, we create a custom event attached to the standard DOM "click" event. In the widget.js file, we write the following:

```
CustomWidget.prototype.init = function() {
    this.fire('myEvent');
};
CustomWidget.mapDomEvents( { 'click': 'myEvent' } );
```

You can customize how the event appears in the GUI Designer even though it's not necessary, by writing:

```
CustomWidget.addEvent({
    'name':'myEvent',
    'description':'On My Event',
    'category':'My Custom Events'
});
```

When you click on your custom widget, the "myEvent" event will be launched.

removeSubscriber( )

void **removeSubscriber**(Object *subscriber*)

Parameter	Type	Description
subscriber	Object	Subscriber object

Description

With **removeSubscriber( )**, you can remove a subscriber that was created with **subscribe( )**.

Example

If you subscribe to an event, as shown below:

```
var mySubcriber = $$('customWidget1').subscribe('click', function() {
    //do something
});
```

You can remove the subscriber from it in the following manner:

```
$$('customWidget1').removeSubscriber({mySubcriber);
```

subscribe( )

Object **subscribe**(String *event*[, String *target*], Function *callback*[, String *observer*][, Object *userData*])

Parameter	Type	Description
event	String	Event to which to subscribe
target	String	Target filtering for the event
callback	Function	Callback after the event is fired
observer	String	Object subscribing to the event
userData	Object	Data passed to the callback
Returns	Object	Subscriber object

Description

subscribe( ) allows you to subscribe to an event and execute a ***callback*** asynchronously. The event must have already been defined for the custom widget by using

mapDomEvents( ).

Example

If you subscribe to an event, as shown below:

```
var mySubcriber = $$('customWidget1').subscribe('click', function() {
    //do something
});
```

You can unsubscribe from it in the following manner:

```
$$('customWidget1').unsubscribe({
    event: 'click'
});
```

unsubscribe( )

void **unsubscribe**(Object *subscriber*)

Parameter	Type	Description
subscriber	Object	Object defining the event, target, callback, and observer

Description

unsubscribe( ) allows you to unsubscribe a subscriber from an event.

Example

If you subscribe to an event, as shown below:

```
var mySubcriber = $$('customWidget1').subscribe('click', function() {
    //do something
});
```

You can unsubscribe from it in the following manner:

```
$$('customWidget1').unsubscribe({
    event: 'click'
});
```

Position

To retrieve the positions of the top, left, bottom, and right coordinates after having setting them using the functions below, you can do so by writing:

```
var rightCoord = $$('customWidget1').getNode().style.right;
```

rightCoord is equal to "33px" if we wrote the following code beforehand:

```
$$('customWidget1').right(33);
```

Each coordinate is a property in the "style" object: top, left, bottom, and right.

absolutePosition( )

Object **absolutePosition**

Returns	Object	Absolute position of the widget

Description

absolutePosition( ) returns the widget's absolute position. In the object that it returns, you can retrieve the top and left coordinates:

```
var position =$$('customWidget1').absolutePosition();
```

The two properties below define the left and top coordinates of the widget's absolute position:

Property	Description
left	Left offset
top	Top offset

bottom( )

void **bottom** (offset)

Parameter	Type	Description
offset	Number	Widget's bottom offset in pixels

Description

bottom( ) allows you to set the widget's bottom offset in pixels.

fitToBottom( )

void **fitToBottom**( )

Description

fitToBottom( ) allows you to fit the widget to the bottom of the Page or Container in which it is located. Consequently, the width and height of the widget will be modified.

fitToLeft( )

void **fitToLeft**( )

Description

fitToLeft( ) allows you to fit the widget to the left of the Page or Container in which it is located. Consequently, the width and height of the widget will be modified.

fitToRight( )

void **fitToRight**( )

Description

fitToRight( ) allows you to fit the widget to the right of the Page or Container in which it is located. Consequently, the width and height of the widget will be modified.

fitToTop( )

void **fitToTop**( )

Description

fitToTop( ) allows you to fit the widget to the top of the Page or Container in which it is located. Consequently, the width and height of the widget will be modified.

left( )

void **left** (position)

Parameter	Type	Description

position	Number	Widget's left offset in pixels

Description

left( ) allows you to set the widget's left offset in pixels.

position( )

Object **position**

Returns	Object	Position of the widget

Description

position( ) returns the widget's position depending on its location (if it's in a Container). In the object that it returns, you can retrieve the top and left coordinates:

```
var position =$$('customWidget1').position();
```

The two properties below define the left and top coordinates of the widget's position:

Property	Description
left	Left offset
top	Top offset

right( )

void **right** (position)

Parameter	Type	Description
position	Number	Widget's right offset in pixels

Description

right( ) allows you to set the widget's right offset in pixels

top( )

void **top** (position)

Parameter	Type	Description
position	Number	Widget's top offset in pixels

Description

top( ) allows you to set the widget's top offset in pixels.

Properties

The functions in the Properties category allow you to set or get the value of your widget's property, define a callback when the property changes or bind a datasource attribute to it.

{propertyName}.onChange( )

void **{propertyName}.onChange**(Function *callback*)

Parameter	Type	Description
callback	Function	Callback function to call

Description

{propertyName}.onChange( ) allows you to set a callback when the value of the property changes.

event object for onChange event

In the event object are the following properties:

Name	Description
data.oldValue	Previous value for the property
data.value	Actual value for the property
kind	Event type
parentEvent	Event that triggered this event
target	Property name

Example

In the following example, we set a callback function to occur when the value in the *test* property changes:

```
documentEvent.onLoad = function documentEvent_onLoad (event)
{
    $$('customWidget1').test.onChange(function(newValue) {
        //do something here
    });
};
```

The *newValue* parameter contains the new value of the property.

{propertyName}( )

void **{propertyName}**(String | Number | Boolean *value*)

Parameter	Type	Description
value	String, Number, Boolean	Value to set for the property

Description

{propertyName}( ) allows you to get or set the value in the property whose name is the function's name.

Getting the property's value

To retrieve the value in a property you defined as "titleProp", you write the following:

```
var propValue = this.titleProp(); //propValue contains the current value of the titleProp property
```

Setting the property's value

If you want to set the value of the "titleProp" property, you write the following:

```
this.titleProp("New value");
```

bindDatasourceAttribute( )

void **bindDatasourceAttribute**(DataSource *datasource* , String *attribute* [, String *property*])

Parameter	Type	Description
datasource	DataSource	Datasource in which the attribute is located (e.g., sources.company)
attribute	String	Attribute name as defined in the datastore class
property	String	Custom widget's property name to bind to the datasource attribute

Description

With **bindDatasourceAttribute( )**, you can bind a datasource attribute to a property.

Example

In our example below, we modify the datasource attribute bound to our custom widget's "test" property so that it's now bound to the "name" attribute in the "company" datasource:

```
$$('customWidget1').bindDatasourceAttribute(sources.company,"name","test");
```

bindDatastourceAttributeWithCallback( ) **NOT PUBLIC**

void **bindDatastourceAttributeWithCallback **NOT PUBLIC****

Description

bindDatastourceElement( ) **NOT PUBLIC**

void **bindDatastourceElement **NOT PUBLIC****

Description

Properties Datasource

All the functions in this category can be used on a property of type "datasource". For example, if the syntax is {*propertyName*}.attributes( ), you'd write the following if your property's name of type "datasource" is "testProperty":

```
var myAttributes = $$('customWidget1').testProperty.attributes( )
```

or, when in the "widget.js" file:

```
var myAttributes = this.testProperty.testProperty.attributes( )
```

{propertyName}.attributeFor( )

String **{propertyName}.attributeFor**(String *attribute*)

Parameter	Type	Description
attribute	String	Datasource defined for a property in the "attributes" property
Returns	String	Datasource attribute bound to attribute

Description

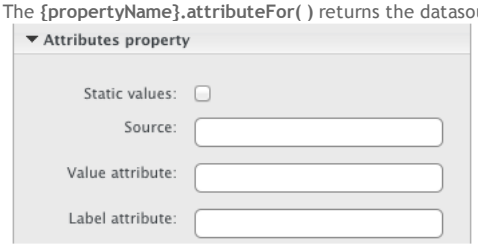
{propertyName}.attributeFor( ) allows you to retrieve the datasource bound to a property's attribute.
For example, if you have created a property of type "datasource":

```
CustomWidget.addProperty('attributes', {
    type: 'datasource',
    attributes: [{
        name: 'value'
    }, {
        name: 'label'
    }]
});
```

If you write:

```
var myds = $$('customWidget1').myAtts.attributeFor('value');
```

The **{propertyName}.attributeFor( )** returns the datasource affected in the "Attribute value" property:



{propertyName}.attributes( )

Array **{propertyName}.attributes**( )

Returns	Array	Attributes defined for the property

Description

{propertyName}.attributes( ) allows you to retrieve the property's attributes.
If you have created your property of type "datasource":

```
CustomWidget.addProperty('myAtts', {
    type: 'datasource',
    attributes: [{
        name: 'value'
    }, {
        name: 'label'
    }]
});
```

In our example above, **{propertyName}.attributes( )** returns an array in which each object is an attribute:

```
[ { name = "value" },  { name = "label" } ]
```

{propertyName}.getCollection( )

void **{propertyName}.getCollection**(Function *callback* [, Function *errorCallback*])

Parameter	Type	Description
callback	Function	Callback function
errorCallback	Function	Error callback

Description

{propertyName}.getCollection( ) allows you to retrieve the current collection of the datasource mapped to the property.

```
this.source.getCollection(function(elements) {
    // do something...
});
```

In the callback function's parameter *elements*, we retrieve an array of objects containing the current values for the attributes:

```
[ { company="Apple", url="http://www.apple.com"}, { company="4D", url="http://www.4d.com"} ]
```

{propertyName}.mapElement( )

Object **{propertyName}.mapElement**(Object *map*)

Parameter	Type	Description
map	Object	Map the datasource to the value for each attribute
Returns	Object	Object containing a property for each attribute and its value

Description

{propertyName}.mapElement( ) allows you to map an element in the datasource.
In the *map* object, you pass the datasource bound to the property with its value:

```
{ datasource: "value" }
```

Example

If you have the following property:

```
CustomWidget.addProperty('myAtts', {
    type: "datasource",
    attributes: [{
        name: 'coName'
    }, {
        name: 'coUrl'
    }]
});
```

The "coName" property's datasource is "name" and the "coUrl" property's datasource is "url":

```
{ coName:"name", coUrl:"url" }
```

If you call the following code:

```
var myMap = $$('customWidget1').myAtts.mapElement({ name: "4D", url: "http://www.4D.com/" })
```

myMap will return the following:

```
{ coName:"4D", coUrl:"http://www.4D.com/" }
```

{propertyName}.onChange( )

void **{propertyName}.onChange**(Function *callback*)

Parameter	Type	Description
callback	Function	Callback function

Description

{propertyName}.onChange( ) allows you to define a callback that will be executed when the property changes.

```
this.myAtts.onChange(function(event) {
    // do something...
});
```

{propertyName}.onCollectionChange( )

void **{propertyName}.onCollectionChange**(Function *callback* [, Function *errorCallback*])

Parameter	Type	Description
callback	Function	Callback function
errorCallback	Function	Error callback

Description

{propertyName}.onCollectionChange( ) allows you to install a callback to get the mapped collection if the datasource changes.

```
this.myAtts.onCollectionChange(function(elements) {
    // do something...
});
```

In the callback function's parameter *elements*, we retrieve an array of objects containing the values for the attributes:

```
[ { company="Apple", url="http://www.apple.com"}, { company="4D", url="http://www.4d.com"} ]
```

{propertyName}.setMapping( )

void **{propertyName}.setMapping**(Object *map*)

Parameter	Type	Description
map	Object	Map for the property's attributes

Description

{propertyName}.setMapping( ) allows you to map the attributes with datasources.
The object you send contains the property and its new datasource attribute.
For example, if we have two attributes in the myAtts property: "title" and "link", we assign two datasource attributes using this function:

```
$$('customWidget1').myAtts.setMapping({ title:"fullName", link:"email"})
```

{propertyName}( )

DataSource **{propertyName}**( )

Returns	DataSource	Datasource bound to this property

Description

{propertyName}( ) allows you to retrieve the datasource bound to this property.
A property for each attribute that you define is included in this datasource object.
For example, if you have defined the "name" attribute from your Company datastore class, the value of the current entity will be in the *mydsName* variable:

```
var mydsName = $$('customWidget1').mysource().name;
```

Properties List

All the functions in this category can be used on a property of type "list". For example, if the syntax is {*propertyName*}.count( ), you'd write the following if your property's name of type "list" is "listProperty":

```
var myCount = $$('customWidget1').listProperty.count( )
```

or, when in the "widget.js" file:

```
var myCount = this.listProperty.count( )
```

You define **attributes** for the property of type "list". An **element** is the set of values you specify for each set of attributes.

Events fired by function

The following events are fired after each function is called:

{propertyName}.concat( )

- onInsert event for each added attribute
- onChange event

{propertyName}.insert( )

- onInsert event
- onChange event

{propertyName}.move( )

- onRemove event
- onInsert event
- onMove event
- onChange event

{propertyName}.pop( )

- onRemove event
- onChange event

{propertyName}.push( )

- onChange event

{propertyName}.remove( )

- onRemove event
- onChange event

{propertyName}.removeAll( )

- onRemove event for each removed attribute
- onChange event

{propertyName}.shift( )

- onRemove event
- onChange event

{propertyName}( )

Array **{propertyName}**(Number *element*, Object *listElement*)

Parameter	Type	Description
element	Number	Element to insert or modify
listElement	Object	An object defining an element
Returns	Array	Array of objects defining the elements in the list property

Description

{propertyName}( ) allows you to return an array of objects in which each object is an element in the list property.

You can also update an existing element or add a new one by passing the ***element*** and ***listElement*** parameters to **{propertyName}( )**. Afterwards, the array of objects is returned as well. In this case, the following events are fired: *onRemove* (if an element is removed), *onInsert*, *onModify*, and *onChange*.

Example

The following example modifies the first element in the list property:

```
var listElements = $$("customWidget1").listproperty(0, {value: "modifiedValue", label:"Modified Value" })
```

listElements contains an array defining the elements in the list property with the first one modified as specified.

{propertyName}.concat( )

void **{propertyName}.concat**(Array *listElements*)

Parameter	Type	Description
listElements	Array	An array containing multiple objects that define an element

Description

{propertyName}.concat( ) allows you to add an array of elements, ***listElements***, in the property.
The *onInsert* event is fired for each attribute added and then the *onChange* event is fired.

Example

The following example adds two elements in the listProp property:

```
$$("customWidget1").listProp.concat( [ { value: "value1", label: "label1" }, { value: "value2", label: "label2" } ] );
```

{propertyName}.count( )

Number **{propertyName}.count**( )

Returns	Number	Number of elements defined in the list property

Description

{propertyName}.count( ) returns the number of elements in the list property.

{propertyName}.first( )

void **{propertyName}.first**( )

Description

{propertyName}.first( ) allows you to select the first element in the property of type "list".

{propertyName}.insert( )

Number **{propertyName}.insert** (element , listElement)

Parameter	Type	Description
element	Number	Element where listElement is inserted
listElement	Object	An object defining an element
Returns	Number	Index of listElement inserted in the property

Description

{propertyName}.insert( ) allows you to insert a ***listElement*** in the property at ***element***. This function returns the added element's index in the property.
The *onInsert* event is fired and then afterwards the *onChange* event.

Example

The following example inserts ***listElement*** at the first position:

```
var index = $$("customWidget1").listProp.insert(0, { value: "value1", label: "label1" }); //returns 0
```

{propertyName}.last( )

void **{propertyName}.last**( )

Description

{propertyName}.last( ) allows you to select the last element in the property of type "list".

{propertyName}.move( )

void **{propertyName}.move**(Number *element*, Number *newPosition*)

Parameter	Type	Description
element	Number	Element to move
newPosition	Number	New position for element

Description

{propertyName}.move( ) allows you to move ***element*** to ***newPosition***.
The *onRemove*, *onInsert*, and *onMove* events are fired after calling {propertyName}.move( ). As usual, the *onChange* event is fired afterwards.

Example

This example moves the 7th element to the 1st position:

```
$$('customWidget1').listProp.move(6,0);
```

{propertyName}.onChange( )

void **{propertyName}.onChange**(Function *callback*)

Parameter	Type	Description
callback	Function	Callback function

Description

{propertyName}.onChange( ) allows you to define a *callback* that will be executed when the property changes.

```
this.listproperty.onChange(function(event) {
    // do something...
});
```

This *callback* is executed after other events occur, like an object being inserted, removed, and moved.

event object

The *event* object returns two properties:

Property	Type	Description
index	Number	Element number
value	Array	An array of objects (defining the elements) in the "list" property that were affected

{propertyName}.onInsert( )

void **{propertyName}.onInsert** (callback)

Parameter	Type	Description
callback	Function	Callback function

Description

{propertyName}.onInsert( ) allows you to define a callback function when an element is inserted in the property of type "list".

event object

The *event* object returns two properties:

Property	Type	Description
index	Number	Element number
value	Array	An array of objects (defining the elements) in the "list" property that were affected

{propertyName}.onModify( )

void **{propertyName}.onModify** (callback)

Parameter	Type	Description
callback	Function	Callback function

Description

{propertyName}.onModify( ) allows you to define a callback function when the value of an element is modified in the property of type "list".
You can modify an element by passing the element number and its new values to the **{propertyName}( )** function.

event object

The *event* object returns two properties:

Property	Type	Description
index	Number	Element number
value	Array	An array of objects (defining the elements) in the "list" property that were affected

{propertyName}.onMove( )

void **{propertyName}.onMove** (callback)

Parameter	Type	Description
callback	Function	Callback function

Description

{propertyName}.onMove( ) allows you to define a callback function when an element is moved in the property of type "list".

event object

The *event* object returns two properties:

Property	Type	Description
index	Number	Element number
value	Array	An array of objects (defining the elements) in the "list" property that were affected

{propertyName}.onRemove( )

void **{propertyName}.onRemove**(Function *callback*)

Parameter	Type	Description
callback	Function	Callback function

Description

{propertyName}.onRemove( ) allows you to define a callback function when an element is removed from the property of type "list".
If, for example, you call **{propertyName}.removeAll( )**, the callback defined for {propertyName}.onRemove( ) is called for each object removed.

event object

The *event* object returns two properties:

Property	Type	Description
index	Number	Element number
value	Array	An array of objects (defining the elements) in the "list" property that were affected

{propertyName}.pop( )

Object **{propertyName}.pop**( )

Returns	Object	Removed element

Description

{propertyName}.pop( ) allows you to remove the last element in the property of type "list". The removed element is returned by this function.
The *onRemove* event is fired after calling {propertyName}.pop( ). As usual, the *onChange* event is fired afterwards.

{propertyName}.push( )

Number **{propertyName}.push**(Object *listElement*)

Parameter	Type	Description
listElement	Object	An object defining an element
Returns	Number	Index of the listElement added

Description

{propertyName}.push( ) allows you to append ***listElement*** to the property. This function returns the added element's index in the property.
The *onChange* event is fired after calling this function.

Example

In the following example, we add ***listElement*** to the end of the elements in the listProp property:

```
var index = $$("customWidget1").listProp.push({ value: "value1", label: "label1" });
```

{propertyName}.remove( )

Object **{propertyName}.remove**(Number *element*)

Parameter	Type	Description
element	Number	Define which object to remove (first element is 0)
Returns	Object	Removed element

Description

{propertyName}.remove( ) allows you to remove a specific object defined by ***element*** in the property. The removed object is returned by this function.
The *onRemove* and *onChange* events are fired after calling this function.

{propertyName}.removeAll( )

void **{propertyName}.removeAll**( )

Description

{propertyName}.removeAll( ) allows you to remove all the elements in the property of type "list".
The *onRemove* is fired after the removal of each element. Afterwards, the *onChange* events is fired.

{propertyName}.shift( )

Object **{propertyName}.shift**( )

Returns	Object	Removed element

Description

{propertyName}.shift( ) allows you to remove the first element defined for the property. The removed element is returned by this function. The *onRemove* and *onChange* events are fired after calling this function.

Size

These functions allow you to set the size of the instance of your custom widget.

autoHeight( )

void **autoHeight**( )

Description

autoHeight( ) allows you to set the height to "auto" in the "style" property of the widget.
For example, after calling this function, the "style" property becomes:

```
style="height: auto;"
```

autoWidth( )

void **autoWidth**( )

Description

autoWidth( ) allows you to set the width to "auto" in the "style" property of the widget.
For example, after calling this function, the "style" property becomes:

```
style="width: auto;"
```

height( )

Number **height**( )

Returns	Number	Widget's height

Description

height( ) returns the widget's height in pixels.

size( )

Object **size**( )

Returns	Object	Object containing widget's width and height

Description

size( ) returns the widget's size (height and width) in pixels.
Here's a way to retrieve the widget's size:

```
var widgetSize = $$('customWidget1').size();
        //height = widgetSize.height
        //width = widgetSize.width
```

width( )

Number **width**( )

Returns	Number	Widget's width

Description

width( ) returns the widget's width in pixels.

Style

The functions in the Style category allow you to define the CSS style(s) for the instance of your custom widget.
For information about Wakanda's generic CSS classes, refer to **Using Wakanda's generic CSS classes**.

addClass( )

void **addClass**(String *cssClass*)

Parameter	Type	Description
cssClass	String	CSS class to add to the widget

Description

addClass( ) allows you to define your widget's CSS class by passing it to ***cssClass***.
The *cssClass* is added to the widget's DOM node's *class* property:

```
<div id="customWidget1" data-type="CustomWidget" data-lib="WAF" data-package="CustomWidget"
class="waf-widget waf-customwidget customClass1 customClass2" data-constraint-left="true" data-constraint-top="true">
```

For information about Wakanda's generic CSS classes, refer to **Using Wakanda's generic CSS classes**.

bindDatasourceAttributeCSS( )**NOT PUBLIC**

void **bindDatasourceAttributeCSS**NOT PUBLIC****(DataSource *datasource*, String *attribute*, String *cssProperty*)

Parameter	Type	Description
datasource	DataSource	Datasource to which to bind
attribute	String	Attribute name
cssProperty	String	CSS property to which to bind

Description

bindDatasourceAttributeCSS( )NOT PUBLIC**** allows you to bind a datasource attribute to a CSS property.

hasClass( )

Boolean **hasClass**(String *cssClass*)

Parameter	Type	Description
cssClass	String	CSS class
Returns	Boolean	True/False = custom widget has cssClass

Description

hasClass( ) allows you to check if ***cssClass*** exists for the custom widget.

hide( )

void **hide**( )

Description

Call the **hide( )** function to hide the widget on the Page.

removeClass( )

void **removeClass**(String *cssClass*)

Parameter	Type	Description
cssClass	String	CSS class to remove

Description

removeClass( ) allows you to remove a CSS class from the custom widget.

show( )

void **show**( )

Description

With the **show( )** function, you can show the widget on the Page.

style( )

void **style**(String *cssProperty*, String *value*)

Parameter	Type	Description
cssProperty	String	CSS property, e.g., "background", "font-size", "border"

value	String	Value for the cssProperty

Description

style( ) allows you to define a value for a CSS property.
For example, you can write the following to set the background of the widget to grey:

```
$$('customWidget1').style('background','#ccc');
```

toggleClass( )

void **toggleClass**(String *cssClass*)

Parameter	Type	Description
cssClass	String	CSS class to toggle

Description

toggleClass( ) allows you to toggle a CSS class for the custom widget. If it exists in the custom widget's "class" property, it will be removed and if it does not exist, it will be added.

Subscriber

The functions in this category allow you to interact with a subscriber that you defined for a specific event by using the functions in the **Observable** category.

isPaused( )

Boolean **isPaused**( )

Returns	Boolean	True/False = event is paused

Description

isPaused( ) allows you to discover if the event is paused.

pause( )

void **pause**( )

Description

pause( ) allows you to temporarily pause the event.

resume( )

void **resume**( )

Description

resume( ) allows you to resume a paused event.

unsubscribe( )

void **unsubscribe**( )

Description

unsubscribe( ) allows you to unsubscribe a subscriber.

Example

If you subscribe to an event, as shown below:

```
var mySubcriber = $$('customWidget1').subscribe('click', function() {
    //do something
});
```

You can also unsubscribe from it in the following manner:

```
$$('customWidget1').unsubscribe({
    event: 'click'
});
```

or

```
mySubscriber.unsubscribe(); //the subscriber object was created with the subscribe() function
```

Widget

These functions allow you to interact with your custom widget's instance.

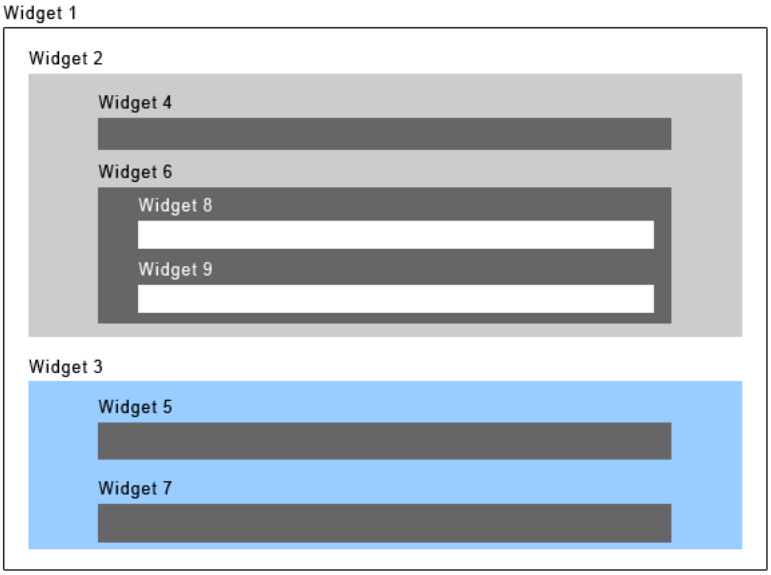
allChildren( )

Array **allChildren**( )

Returns	Array	Children widgets (along with children's children widgets)

Description

With **allChildren( )**, you can retrieve all the "children" widgets. The "children" widgets are all those that are included in the "parent" widget.
If you call the **allChildren()** function on "Widget1" (see screenshot below), an array containing all the children widgets's IDs (widgets 2 to 9) is returned.
If you call the **children()** function on "Widget1", an array containing the children widgets's IDs (widgets 2 and 3) is returned.



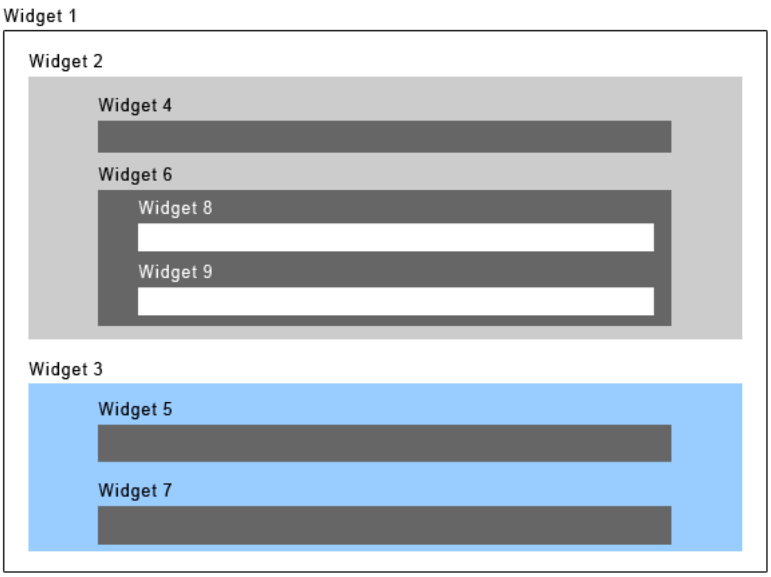
children( )

Array **children**( )

Returns	Array	Children widgets (only first level)

Description

With **children( )**, you can retrieve the "children" widgets for the "parent" widget. The "children" widgets are all those included in the "parent" widget.
If you call the **allChildren()** function on "Widget1" (see screenshot below), an array containing all the children widgets's IDs (widgets 2 to 9) is returned.
If you call the **children()** function on "Widget1", an array containing the children widgets's IDs (widgets 2 and 3) is returned.



disable( )

void **disable**( )

Description

disable( ) allows you to disable the widget.
The "waf-state-disabled" CSS class is added to the widget's "class" property:

```
<div id="customWidget1" class="waf-widget waf-customwidget waf-state-disabled"... >
```

disabled( )

Boolean **disabled**( )

Returns	Boolean	True/False = widget is disabled

Description

With **disabled( )**, you can test if the widget has been disabled or not.

enable( )

void **enable**( )

Description

enable( ) allows you to enable the widget if it had previously been disabled.

getNode( )

String **getNode**( )

Returns	String	Widget's DOM node

Description

With **getNode( )**, you can retrieve the widget's DOM node.

```
var myNode = $$('customWidget1').getNode();
```

The node returned looks something like this:

```
<div id="customWidget2" data-type="customWidget"
data-constraint-left="true" data-constraint-top="true" data-label="customWidget2"
data-label-position="top" data-lib="WAF" data-package="customWidget"
class="waf-widget waf-customwidget" >
```