Widgets

With this Widget API, you can access and retrieve data, get and set properties as well as modify the interface for all of Wakanda's widgets.

The **Widget** class contains all the properties and methods that are common to all Wakanda widgets. However, not all the properties and methods are available to all widgets: some widgets do not make use of certain methods or properties and therefore do or return nothing.

The following widgets have their own specific functions:

- **Auto Form**
- **Calendar**
- **Canvas**
- **Checkbox**
- **Combo Box**
- **Component**
- **Container**
- **Dialog**
- **File Upload**
- **Frame**
- **Grid**
- **Grid Column**
- **Image Button**
- **Login Dialog**
- **Matrix**
- **Menu Bar**
- **Navigation View**
- **Progress Bar**
- **Query Form**
- **Slider**
- **Switch**
- **Tab View**
- **WYSIWYG Editor**

The **Grid Column** class can only be applied to the Grid widget. To use any of these functions, you must first use the **column( )** function to retrieve the *column* object needed to define the Grid's column.

Syntax

For these methods, you must use the following two notations for a widget whose ID is "widgetID":

```
var myValue = $$('widgetID').getValue();
        //or you could also write
var myValue = WAF.widgets.widgetID.getValue();
```

For a property, you can write the following:

```
var myObj = $$('widgetID').format;
```

If you are in the context of the widget, you can write:

```
var myValue = this.getValue();
var myObj = this.format;
```

Accessing a Widget's Label

If you have specified a value for a widget's **Label** property, a Label widget is added to it. To apply a function from the Widget class API to the widget's label, you must apply the function to the widget's **label** ID that you retrieve by using the **getLabel( )** function.

For example, you can write the following:

```
var labelWidget = $$('combobox1').getLabel();
labelWidget.setTextColor("red");
```

You can also write:

```
$$('combobox1').getLabel().setTextColor("red");
```

To access the DOM of the label, you can write the following:

```
var labelDOM = $$('combobox1').label;
```

Auto Form

Besides the methods in the **Widget** class that you can apply to an **Auto Form** widget, you can also use these methods that are specific to the Auto Form widget:

For the Auto Form widget, you can perform all the actions in the footer buttons:

- **addEntity( )**: Add a new entity.
- **dropEntity( )**: Drop the current entity.
- **findEntity( )**: Query the datasource with the data entered in the widgets.
- **nextEntity( )**: Go to the next entity.
- **prevEntity( )**: Go to the previous entity.
- **saveEntity( )**: Save the current entity.

addEntity( )

void **addEntity**( )

Description

addEntity( ) allows you to add a new entity to the **Auto Form** widget.

dropEntity( )

void **dropEntity**( )

Description

dropEntity( ) allows you to drop the current entity in the **Auto Form** widget.

findEntity( )

void **findEntity**( )

Description

The findEntity( ) function allows you to find the entities whose values were entered in the widgets in the Auto Form widget. Before starting the query, you can use the **clear( )** function to clear the values in the Auto Form.

nextEntity( )

void **nextEntity**( )

Description

With nextEntity( ), you can go to the next entity in the **Auto Form** widget.

prevEntity( )

void **prevEntity**( )

Description

With prevEntity( ), you can go to the previous entity in the **Auto Form** widget.

saveEntity( )

void **saveEntity**( )

Description

saveEntity( ) allows you to save the data entered in the **Auto Form** widget. Remember to create an entity beforehand in the Auto Form by calling the **addEntity( )** method.

Button

The **Button** widget inherits from the Widget class; however, not all the properties and functions are available to it.
To set the title of a button, use the **setLabelText( )** method.

Calendar

The **Calendar** widget inherits from the Widget class; however, not all the properties and functions are available to it. To retrieve the value(s) of the Calendar widget, use the **getValue( )** function and to set the value, use the **setValue( )** function.

The **hide( )** function launches the Calendar widget's On Hide event and the **show( )** function launches its On Show event.

This widget also has the following function:

- **getSelectionMode( )**: Returns the selection mode (single, multiple, or range)

getSelectionMode( )

String **getSelectionMode**( )

Returns	String	Selection mode for the Calendar (single, multiple, or range)

Description

getSelectionMode( ) returns the value chosen for the **Selection mode** property for the Calendar widget.

Canvas

The **Canvas** widget inherits from the Widget class; however, not all the properties and functions are available to it. This widget has two functions:

- **get2DContext( )**: Return the 2D context of the Canvas widget.
- **getCanvas( )**: Returns the Canvas widget's DOM object.

For more information about the functions available to the Canvas widget, please refer to the W3C reference.

get2DContext( )

Object **get2DContext**( )

Returns	Object	2D Context of the Canvas widget

Description

get2DContext( ) returns the 2D context of the Canvas widget so that you can draw in it.

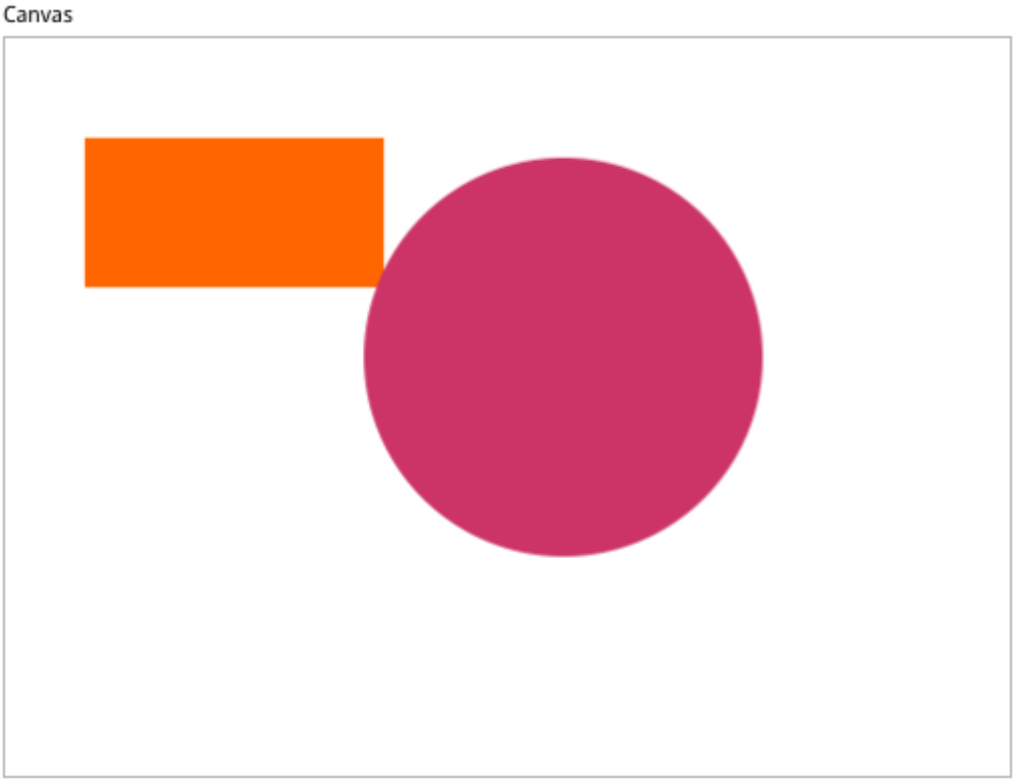
Example

In our example below, we draw a circle and rectangle in the Canvas widget:

```
var context = $$('canvas1').get2DContext();
context.fillStyle="#F60";
context.fillRect(40,50,150,75); //draw rectangle

context.fillStyle="#C36";
context.arc(280,160,100,90,Math.PI*2,true); //draw circle
context.closePath();
context.fill();
```

The Canvas widget now appears as shown below:



getCanvas( )

Object **getCanvas**( )

Returns	Object	Canvas DOM object

Description

getCanvas( ) returns the Canvas widget's DOM object.

Example

In the following example, we clear out the contents of the Canvas widget:

```
var canvas  = $$('canvas1').getCanvas();
var context = $$('canvas1').get2DContext();
context.clearRect(0, 0, canvas.width, canvas.height);
```

Chart

The **Chart** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Checkbox

Besides inheriting most of the properties and functions from the Widget class, the **Checkbox** widget also has its own functions, which are:

- **check( )**: checks the checkbox and sets its value to true
- **toggleCheckbox( )**: either checks the checkbox and sets its value to true or unchecks it and sets it to false
- **uncheck( )**: unchecks the checkbox and sets its value to false

Note: These functions are only available in the development branch for Wakanda.

check( )

void **check**( )

Description

check( ) checks the checkbox and sets its value to true.

toggleCheckbox( )

void **toggleCheckbox**( )

Description

toggleCheckbox( ) checks the checkbox and sets its value to true or unchecks it and sets its value to false depending on the Checkbox widget's current state.

uncheck( )

void **uncheck**( )

Description

uncheck( ) unchecks the checkbox and sets its value to false.

Combo Box

Besides the properties and functions that the **Combo Box** widget inherits from the Widget class and also has two functions of its own:

- **addOption( )**: add an option to the Combo Box
- **removeOption( )**: remove an option from the Combo Box

addOption( )

void **addOption**(String *value* , String *label* [, Boolean *selected*])

Parameter	Type	Description
value	String	Value for the option
label	String	Label for the option
selected	Boolean	True = define option as "selected"

Description

With **addOption( )**, you can add an option to the Combo Box widget by defining its value and label. You can also specify if the value is the selected option by default by passing *True* to the *selected* parameter.

Example

The following example adds an additional option to the Combo Box and selects it by default:

```
$$('cityComboBox').addOption("dallas","Dallas",true);
```

removeOption( )

void **removeOption**(Number *option*)

Parameter	Type	Description
option	Number	Option in the Combo Box to remove (starting at 1)

Description

The **removeOption( )** function allows you to remove an option from the Combo Box widget.

Example

The following example removes the third option in the Combo Box widget:

```
$$('cityComboBox').removeOption(3);
```

Component

Besides the methods in the **Widget** class that you can apply to a **Component** widget, you can also use these methods that are specific to the Component widget:

- **loadComponent( )**: Load a new **Web Component** into the Component widget.
- **removeComponent( )**: Remove the current **Web Component** from the Component widget.
- **widgets**: Property containing an object that defines all the widgets in the **Web Component**.

widgets

Description

The widgets property returns an object containing an object for each widget in the Web Component.

For example, if you have a Web Component containing multiple widgets, this property returns an object that has one object for each widget in it:

```
{
    "button1": { id="component1_button1", kind="button", divID="component1_button1", ...}
    "button2": { id="component1_button2", kind="button", divID="component1_button2", ...}
    "container1": { id="component1_container1", kind="container", divID="component1_conta
    "label1": { id="component1_label1", kind="label", divID="component1_label1", ...}
    "label2":  { id="component1_label2", kind="label", divID="component1_label2", ...}
    "label3": { id="component1_label3", kind="label", divID="component1_label3", ...}
    "richText1": { id="component1_richText1", kind="richText", divID="component1_richText
    "textField1": { id="component1_textField1", kind="textField", divID="component1_textF
    "textField2":     { id="component1_textField2", kind="textField", divID="component1_t
    "textField3":     { id="component1_textField3", kind="textField", divID="component1_t
}
```

Example

In the example below, we fill an array with all the widget IDs of the widgets inside the Web Component displayed by the Component widget:

```
var componentIDArray=[],
compWidgets=$$('component1').widgets;
for (name in compWidgets)
{
    componentIDArray.push(compWidgets[name].id);
}
```

loadComponent( )

void **loadComponent**([String *webComponent* [, Object *componentObject*]])

Parameter	Type	Description
webComponent	String	Path of the Web Component to load
componentObject	Object	Object defining the id, path, and userData

Description

loadComponent( ) allows you to load a Web Component already present in your project into the Component widget on your Interface page. loadComponent( ) has two different syntaxes:

1. Pass the Web Component's path to *webComponent* or
2. Pass an object containing the following three parameters: *id* (Component widget ID the same one or a different one), *path* (path to the Web Component), and *userData* (another object containing data you'd like to pass and retrieve when the Component widget is published). The *id* parameter is not mandatory.

If you have defined a Web Component in the **Path** property for your Component widget and have deselected the **Load by default** property, you can call **loadComponent( )** without passing the Web Component's path as the parameter as long as you have not already removed the Web Component by calling **removeComponent( )**.

Example

If, for example, you have a Component widget whose ID is "myDialog" and have selected the **Modal** property and deselected the **Load by default** property, you can show the Web Component as a modal dialog by simply writing the

following line of code:

```
$$('myDialog').loadComponent("inputEmployeeDialog.waComponent");
```

To close the modal dialog, you can write the following line of code:

```
$$('myDialog').removeComponent();
```

Example

Using the second syntax for loadComponent( ), you can load a specific Web Component and pass your own data:

```
$$('component1').loadComponent({ path: "dialog.waComponent", userData: { myParameter: "Sp
```

In the **this.load** section of your Web Component's JavaScript file, you can retrieve your data in the userData object:

```
var myTextField;
myTextField = getHtmlId('richText1');  //get the ID to a Text widget in my Web Compon
$$(myTextField).setValue(data.userData.myParameter);  //affect the value from the use
```

removeComponent( )

void **removeComponent**

Description

This method allows you to remove the Web Component currently loaded in the Component widget.

Example

See the example for the **loadComponent( )** method.

Container

Besides the methods in the **Widget** class for the **Container** widget, you can also use the methods below:

- **getSplitPosition( )**: Get the splitter position of the Container widget.
- **mobileSplitView( )**: Hide or Show the left splitter Container in a vertically split Container widget.
- **setSplitPosition( )**: Set the splitter position for the Container widget.

collapseSplitter( )

void **collapseSplitter**( )

Description

The collapseSplitter( ) function collapses the first container in a previously split **Container** widget.

expandSplitter( )

void **expandSplitter**( )

Description

The expandSplitter( ) function expands the first container in a previously split **Container** widget.

getSplitPosition( )

Number **getSplitPosition**( )

Returns	Number	Position of the container's splitter

Description

This method allows you to retrieve the current position of the Container widget's splitter. The position is expressed in pixels. Remember that when you split a **Container** widget, each split area is a Container.

Example

Retrieve the Container's splitter's current position when it is moved manually:

```
var myposition = $$('container1').getSplitPosition();
```

mobileSplitView( )

void **mobileSplitView**(Boolean *popupDisplay*)

Parameter	Type	Description
popupDisplay	Boolean	True = collapse left container in split Container widget and show Left Splitter Button; False = expand left container in split Container widget

Description

The mobileSplitView( ) function either collapses the left container in a vertically split **Container** widget and shows the **Left Splitter Button**. Otherwise, the left container in a vertically split Container widget is expanded and the **Left Splitter Button** is hidden.

Note: For more information regarding the Left Splitter Button property, refer to the Showing/Hiding Left Split Container section for the Container widget.

setSplitPosition( )

void **setSplitPosition**(Number *number*)

Parameter	Type	Description

number	Number	Splitter position (in pixels) for the Container to set

Description

Set the position of the splitter in the Container widget. The position is expressed in pixels. Remember that when you split a **Container** widget, each split area is also a Container.

If the Container widget does not have an existing splitter, this function does nothing.

Example

Set the position of the splitter in the Container to 120 pixels:

```
$$('container1').setSplitPosition(120);
```

toggleSplitter( )

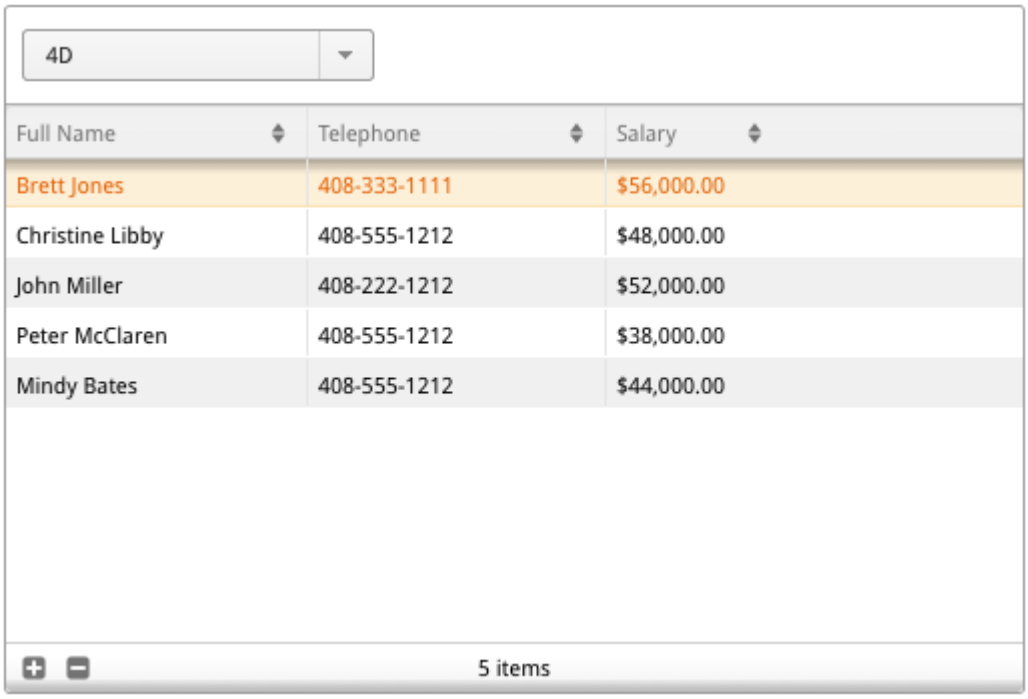
void **toggleSplitter**( )

Description

The toggleSplitter( ) function toggles the first container in a previously split **Container** widget. If the first container in the split Container widget is hidden, it will be displayed. If it is displayed, it will be hidden.

Example

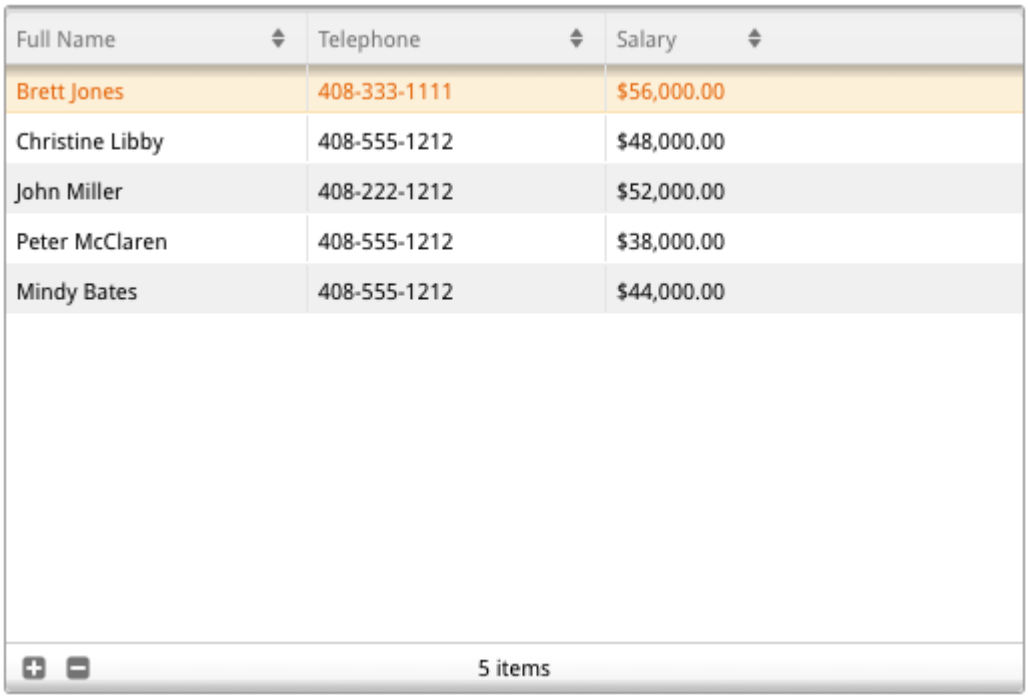
For the following Container widget was split horizontally:



We call the toggleSplitter( ) function to show and/or hide the top container of the Container widget:

```
$$('staffContainer').toggleSplitter();
```

The Container widget then appears as shown below:



By calling the above line of code a second time, the top container reappears.

Note: You can also use the collapseSplitter( ) and expandSplitter( ) functions.

Dialog

Besides the methods in the **Widget** class for the **Dialog** widget, the functions below are specific to this widget:

- **closeDialog( )**: Closes the Dialog widget.
- **displayDialog( )**: Displays the Dialog widget.
- **maximizeDialog( )**: Maximizes the Dialog widget.
- **minimizeDialog( )**: Minimizes the Dialog widget.

closeDialog( )

void **closeDialog**( )

Description

The **closeDialog( )** function allows you to close the Dialog widget.

displayDialog( )

void **displayDialog**( )

Description

The **displayDialog( )** function allows you to display the Dialog widget. A Dialog widget can be defined as **Closed by default** in the **Dialog Properties** and therefore you must use this function to display it.

maximizeDialog( )

void **maximizeDialog**( )

Description

maximizeDialog( ) allows you to maximize the Dialog widget to the window in which it is displayed.

minimizeDialog( )

void **minimizeDialog**( )

Description

minimizeDialog( ) allows you to minimize the Dialog widget to just its title bar in either the Interface page or the Container widget in which it is located. Below is the title bar of a Dialog widget when it is minimized:

To restore the Dialog widget back to its original size, you must click on its title bar.

Display Error

The **Display Error** widget inherits from the Widget class; however, not all the properties and functions are available to it.

The Display Error widget is generally referenced by a widget and you can display (by using the **setErrorMessage( )** function) or clear (by using the **clearErrorMessage( )** function) the error message in this widget (by using these two functions on the widget that has set the Display Error widget's ID in its properties).

You can also define the Display Error widget ID for a particular widget by using the **setErrorDiv( )** function and retrieve the ID by using the **getErrorDiv( )** function.

File Upload

Besides the methods in the **Widget** class for the **File Upload** widget, the functions below are specific to this widget:

- **appendFiles( )**: Appends one or more files to the selected list of files for the widget.
- **getFiles( )**: Gets all the files that the user selected.
- **getFileUploadText( )**: Gets the text defined in the widget's **Text** property or set by **setFileUploadText( )**.
- **getFolderName( )**: Gets the folder defined by the **Temporary folder** or **Folder** property or set by **setFolderName( )**.
- **getMaximumFiles( )**: Gets the number of maximum files defined in the **Maximum files** property.
- **getMaximumFileSize( )**: Gets the maximum size of files defined in the **Maximum size** property.
- **removeAll( )**: Removes all the files that the user selected.
- **removeFile( )**: Removes a specific file that the user selected.
- **setFiles( )**: Sets one or more files to be the one selected for the widget and displayed in the list.
- **setFileUploadText( )**: Sets the text defined in the widget's **Text** property.
- **setFolderName( )**: Sets the folder for the **Temporary folder** or **Folder** property.
- **setMaximumFiles( )**: Sets the number of maximum files for the **Maximum files** property.
- **setMaximumFileSize( )**: Sets the maximum size of files for the **Maximum size** property.
- **setNotificationStatus( )**: Sets the Display notification property.

appendFiles( )

void **appendFiles**(Object | Array *fileList*)

Parameter	Type	Description
fileList	Object, Array	Either a FileList object or an array of files

Description

The **appendFiles( )** function allows you to append one or more files to the list of selected files for the File Upload widget.

You can only have more than one selected file if you have not defined a source for the File Upload widget in which case the files will be uploaded to the server.

You can either pass an object, which is a FileList object, or an array. Each one defines the files to append to the File Upload widget.

Example

In the following example, you can allow users to drag and drop a file onto a Container widget to append them to the File Upload widget.

This code is inserted in the Interface page's **On Load** event:

```
documentEvent.onLoad = function documentEvent_onLoad (event) {
    jQuery.event.props.push("dataTransfer"); // add the dataTransfer property to use with
         // so that you can get information about the files dropped onto the widget

    function handleFileSelect(evt) {
        evt.stopPropagation(); //jQuery event to stop propagating up the DOM tree
        evt.preventDefault(); //jQuery event to stop default event from occurring

        var files = evt.dataTransfer.files; // FileList object containing the file(s) drc
        $$('fileUpload1').appendFiles(files);
    }

    function handleDragOver(evt) {
        evt.stopPropagation();
        evt.preventDefault();
    }

    $$("container1").addListener("dragover", handleDragOver);
    $$("container1").addListener("drop", handleFileSelect);
};
```

Example

An example of passing an array of files to this function would be to retrieve the list of files using the **getFiles( )** function on one File Upload widget and appending that list to another File Upload widget:

```
$$('fileUpload2').setFiles($$('fileUpload1').getFiles());
```

getFiles( )

Array **getFiles**( )

Returns Array An array of objects defining the selected files

Description

getFiles( ) allows you to retrieve an array of objects defining each selected file.
The object for each file has the following properties:

Property	Description
mozFullPath	Local path to the selected file (e.g., "/Users/john/Desktop /myPhoto.jpg")
name	File name (e.g., "myPhoto.jpg")
size	Size of file in bytes (e.g., 7720)
type	MIME file type (e.g., "image/jpeg")

Example

In our example, we put all the names of the selected files for the File Upload widget in an array:

```
var myArray = $$('fileUpload1').getFiles(),
    //get all the selected files
    fileNamesArray = [],
    //create an array that will contain the file names
    i = 0,
    arrLength = myArray.length;
for (i = 0; i < arrLength; i++) {
    fileNamesArray.push(myArray[i].name); //for each object in the array, get the file na
}
```

In our array, we receive the following result based on the order in which the files were added to the File Upload widget:

```
["photo1.jpg", "photo4.jpg", "photo2.jpg", "photo3.jpg"]
```

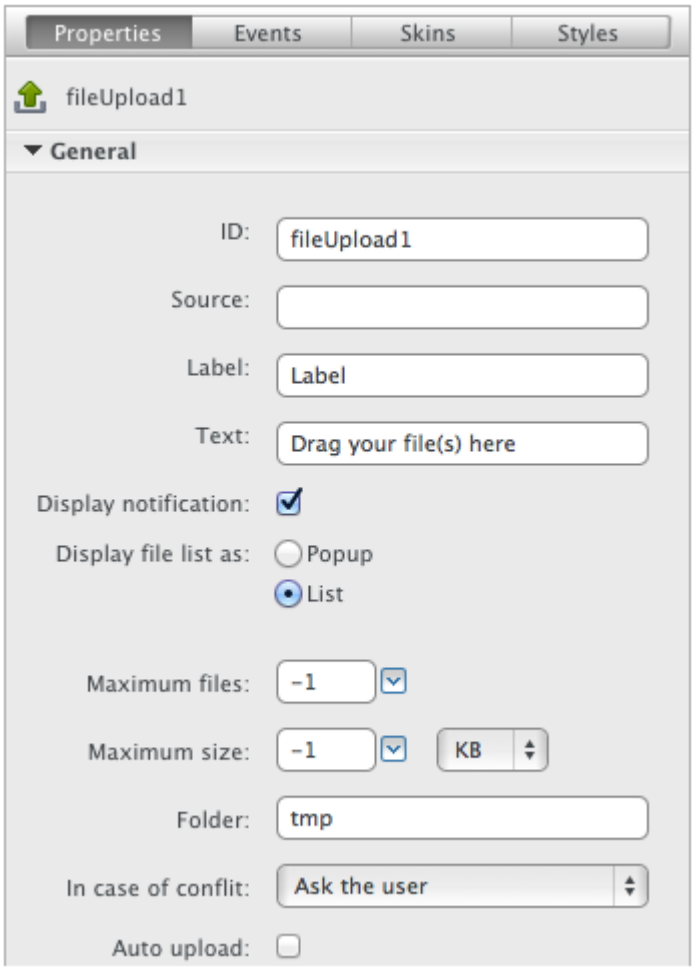
getFileUploadText( )

String **getFileUploadText**( )

Returns String Text displayed inside the File Upload widget

Description

The **getFileUploadText( )** function returns the text defined in the File Upload widget's **Text** property or set by the **setFileUploadText( )** function as shown below:



At runtime, the text appears in the File Upload widget:

getFolderName( )

String **getFolderName**( )

Returns	String	Text entered for the Folder/Temporary folder property

Description

getFolderName( ) returns the folder defined in the **Folder** or **Temporary folder** property or set by **setFolderName( )**.

getMaximumFiles( )

Number **getMaximumFiles**( )

Returns	Number	Maximum number of files defined

Description

getMaximumFiles( ) allows you to retrieve the value set for the **Maximum files** property in the File Upload widget's properties.

getMaximumFileSize( )

String **getMaximumFileSize**( )

Returns	String	Maximum file size

Description

The **getMaximumFileSize( )** function returns the value defined for the widget's **Maximum size** property along with the type (bytes, kb, or mb).

The numeric value can be set by using **getMaximumFileSize( )**. The type used is the one already defined in the widget's properties.

Example

In the following example, maxFileSize contains either "unlimited" if you have left -1 in the **Maximum size** property or the value entered in the **Maximum size** property plus the type selected from the popup menu ("bytes", "kb", or "mb"):

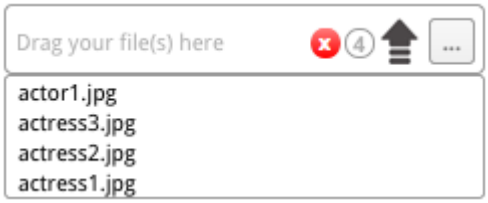
```
var maxFileSize = $$('fileUpload1').getMaximumFileSize();  // returns either "200 kb" or
```

removeAll( )

void **removeAll**( )

Description

The **removeAll( )** function removes all the files the user selected for the File Upload widget.These selected files show up below the File Upload widget if you have selected List as the **Display file list as** property.



If you call the following line of code:

```
$$('fileUpload1').removeAll();
```

The selected files are removed:

removeFile( )

void **removeFile**(Number *fileNumber*)

Parameter	Type	Description
fileNumber	Number	Number of file in the list of selected files

Description

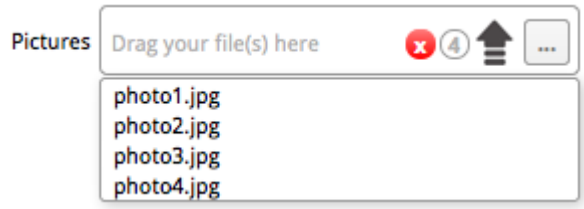
The **removeFile( )** function allow you to remove a specific file in the list of selected files for the File Upload widget.

Example

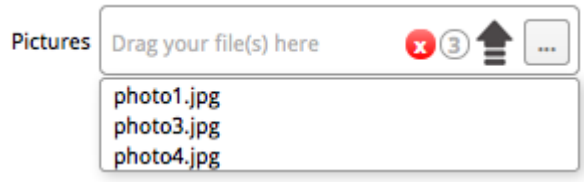
In the following example, we remove the second file in the list of selected files by calling this line of code:

```
$$('fileUpload1').removeFile(2);
```

Before executing that line of code, we have:



Afterwards, we have only three remaining files:



setFiles( )

void **setFiles**(Object | Array *fileList*)

Parameter	Type	Description
fileList	Object, Array	Either a FileList object or an array of files

Description

The **setFiles( )** function allows you to set one or more files to be the ones selected for the File Upload widget and displayed in the list of files. If any other files were previously defined for the File Upload widget, they will be removed and replaced by those set by this function.

You can only have more than one selected file if you have not defined a source for the File Upload widget in which case the files will be uploaded to the server.

You can either pass an object, which is a FileList object, or an array. Each one defines the files to set to the File Upload widget.

Example

Please refer to the example for the **appendFiles( )** function.

setFileUploadText( )

void **setFileUploadText**(String *text*)

Parameter	Type	Description
text	String	Text to insert in the File Upload widget

Description

The **setFileUploadText( )** function allows you to set the text in the File Upload widget. This text can be defined in the File Upload's **Text** property.

By default, the text is the following:

Example

In our example, we modify the text:

```
$$('fileUpload1').setFileUploadText("Select one file please");
```

The File Upload widget appears as shown below:

setFolderName( )

void **setFolderName**(String *folderName*)

Parameter	Type	Description
folderName	String	Text to define the Folder/Temporary folder

Description

The **setFolderName( )** function allows you to define the folder for the **Folder/Temporary folder** property.

Example

In the example below, you can define the Folder property so that you can place the uploaded file(s) into different directories/folders:

```
var uniqueFolder = sources.employee.lastName + sources.employee.ID; // last name is "Smit
$$('fileUpload1').setFolderName("Photos/" + uniqueFolder); // the folder will be "Photos/
```

If the folder/directory does not exist, it will be created in your project's **data** folder.

setMaximumFiles( )

void **setMaximumFiles**(Number *maxFiles*)

Parameter	Type	Description
maxFiles	Number	Maximum number of files to allow to upload

Description

setMaximumFiles( ) allows you to set the maximum number of files to upload with the File Upload widget. By default, this value can be defined in the **Maximum files** property.

If you set this value to -1, the number of files will be unlimited.

setMaximumFileSize( )

void **setMaximumFileSize**(Number *maxFileSize*)

Parameter	Type	Description
maxFileSize	Number	Maximum file size (expressed in bytes)

Description

The **setMaximumFileSize( )** function allows you to set the maximum file size in bytes, KB, or MB as defined in the **Properties** tab.

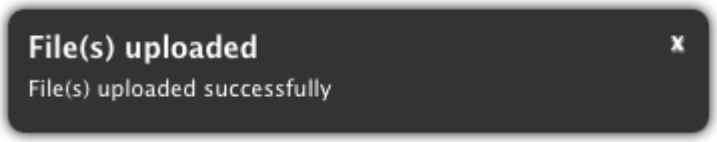
setNotificationStatus( )

void **setNotificationStatus**(Boolean *status*)

Parameter	Type	Description
status	Boolean	True to display notification message when files are successfully uploaded; False to hide it

Description

The **setNotificationStatus( )** function allows you to set the **Display notification** property of the File Upload widget so that you can show or hide the message that appears once one or more files are uploaded successfully:



Example

This example ensures that the **Display notification** property is set to true before uploading a file:

```
$$('fileUpload1').setNotificationStatus(true);
```

Frame

The **Frame** widget inherits from the Widget class; however, not all the properties and functions are available to it. This widget has one function:

- **getFrame( )**: Return the iFrame tag for the Frame widget.

To retrieve the value set for the Frame widget's **Source Page** property, use the **getValue( )** method.

To set the Frame widget's **Source page** property, use the **setValue( )** method. Remember that if you want to pass a URL, you must write it in its entirety, i.e., "http://www.wakanda.org". For a PDF or HTML file, it must already be in the WebFolder and you must use its relative path, e.g., "/pageToInclude.html" or "/pdf/MyDocument.pdf".

Grid

Besides the methods in the **Widget** class that you an apply to the **Grid** widget, the following methods allow you to set or retrieve properties for a specific column or the actual Grid:

- **column( )**: returns the *column* object to then call any of the functions below:
  - **getFormattedValue( )**: retrieve the formatted value in the cell defined by *column* and the currently selected row.
  - **getValueForInput( )**: get the value in the cell defined by *column* and the currently selected row.
  - **setAlignment( )**: set the alignment of the *column*.
  - **setBackgroundColor( )**: set the background color for the entire *column* (including the header).
  - **setTextColor( )**: set the text color for the entire *column* (including the header).
  - **setFormat( )**: set the format for the data in the *column*.
  - **setRenderer( )**: define a function to be applied to each cell in the *column*.
  - **setTextSize( )**: set the text size for the entire *column* (including the header).
  - **setWidth( )**: set the width of the *column*.
- **centerRow( )**: center a specific row in the Grid without changing the current entity.
- **countSelected( )**: count the number of selected rows in the Grid.
- **getRowHeight( )**: get the height of the rows in the Grid.
- **getSelectedRows( )**: get the Grid's selected row numbers.
- **getSelectionMode( )**: get the Grid's selection mode: either "single" or "mulitple".
- **getSortIndicator( )**: get the Grid's current sort indicator for a column.
- **reduceToSelected( )**: reduce the Grid's selection to the currently selected rows.
- **resetSortIndicator( )**: reset the sort indicators for one column or all the columns in the Grid.
- **setReadOnly( )**: set the Grid to read-only or read/write mode.
- **setRowHeight( )**: set the height of the rows in the Grid.
- **setSelectedRows( )**: select specific rows in the Grid.
- **setSelectionMode( )**: set the selection mode to either "single" or "mulitple".
- **setSortIndicator( )**: set the column's sort indicator to show either ascending (up arrow) or descending (down arrow) order.

centerRow( )

void **centerRow** (rowNumber)

Parameter	Type	Description
rowNumber	Number	Row number to display at the center of the Grid

Description

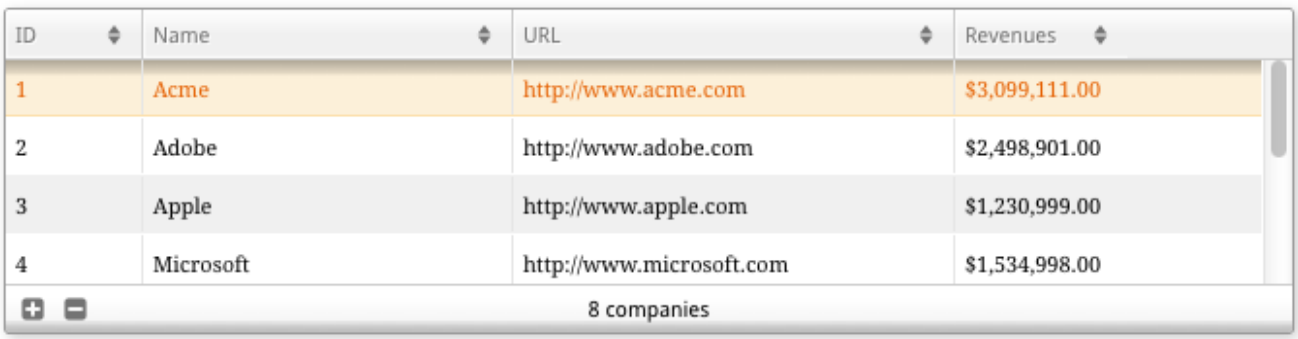
centerRow( ) allows you to center the Grid with the *rowNumber* without changing the current entity. Row numbers begin at 0.

Example

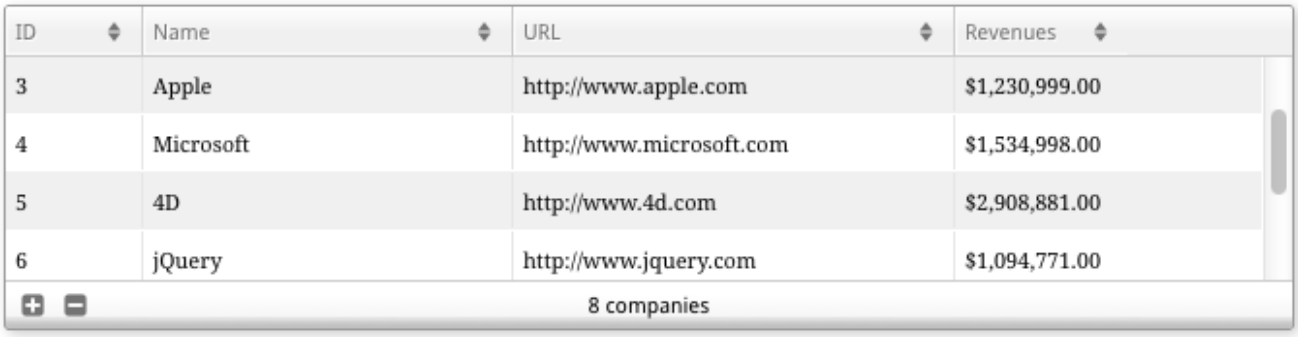
In our example, we center the Grid so that the fourth row is shown at the top.

```
$$('dataGrid1').centerRow(4);
```

Before calling the line above, the Grid appeared as follows:



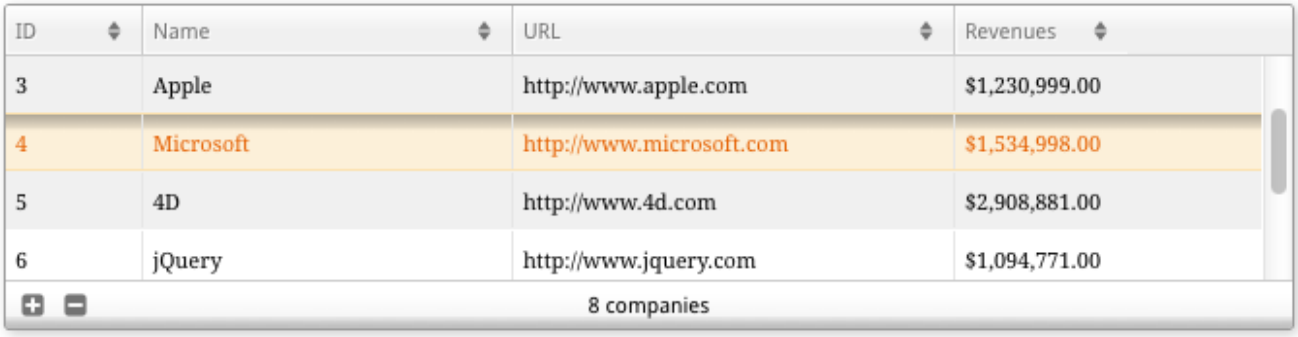
Afterwards, it is shown below:



If we wanted to select the fourth row, we'd have to write the following code:

```
sources.company.select(3);
```

The Grid will appear as follows:



column( )

Column **column**(Number | String *columnRef*)

Parameter	Type	Description
columnRef	Number, String	Column name (e.g., "lastName") or column number (e.g., 3) in the Grid
Returns	Column	Column of the grid

Description

column( ) returns an object of type *Column* to which you can apply other functions in the **Grid Column** class. Numbering of the columns starts at 1.

For example, you can use this function in one of two ways:

```
$$("employeeGrid").column(2).getFormattedValue();
        // or
$$("employeeGrid").column("lastName").getFormattedValue();
```

countSelected( )

Number **countSelected**( )

Returns	Number	Number of selected rows

Description

countSelected( ) returns the number of selected rows when a Grid's **Selection mode** is "multiple". If the Selection mode is "single," this function returns 1.

getRowHeight( )

Number **getRowHeight**

Returns	Number	Height (in pixels) of the rows in the Grid

Description

getRowHeight( ) allows you to retrieve the height (in pixels) of the rows in the Grid widget. By default, the row height is 28.

getSelectedRows( )

Array **getSelectedRows**( )

Returns	Array	An array containing the selected row numbers

Description

getSelectedRows( ) allows you to retrieve the selected rows in a Grid when the **Selection mode** is "multiple". Row numbers start at 0.

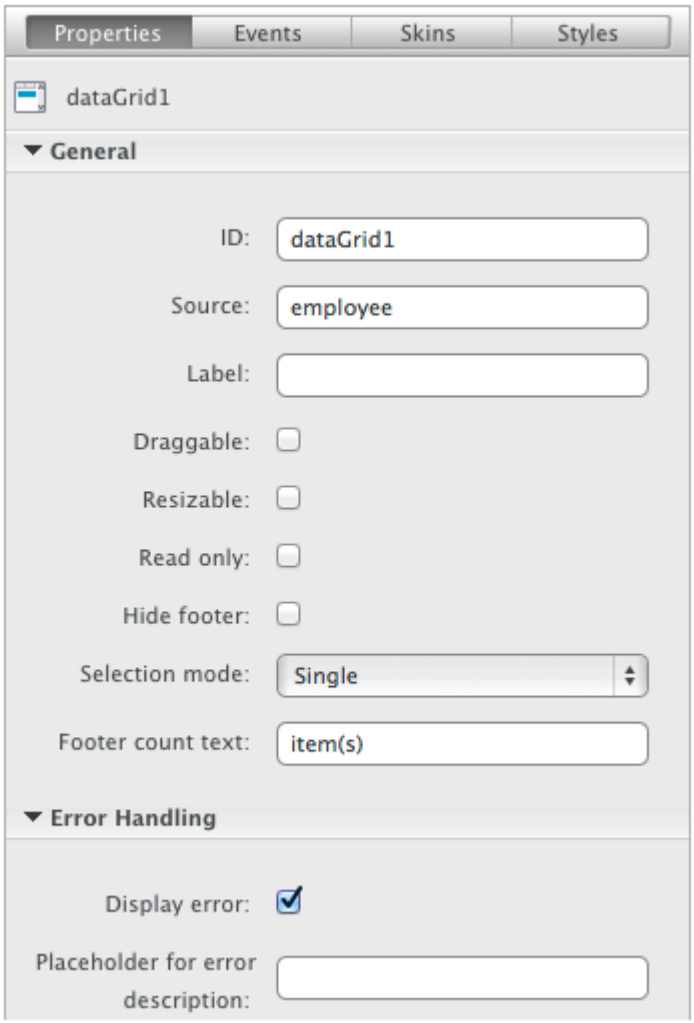
getSelectionMode( )

String **getSelectionMode**( )

Returns	String	Returns either "single" or "multiple"

Description

The **getSelectionMode( )** function returns the Selection mode defined in the **Grid**'s Properties tab. The possible values can be either "single" or "multiple".



getSortIndicator( )

Object **getSortIndicator**( )

Returns	Object	An object defining the column and its sort order ("asc" or "desc")

Description

The **getSortIndicator( )** function returns an object with two properties, *colNb* and *order*, that define how the Grid is currently being sorted.

colNb defines the column number (starting at 0) and *order* defines the sort order (either "asc" or "desc").

If none of the columns had the sort indicator set with **setSortIndicator( )**, this function returns *null* to both properties.

reduceToSelected( )

void **reduceToSelected**([Object *options* [, Object *userData*]])

Parameter	Type	Description
options	Object	Block of options for asynchronous execution
userData	Object	Block of user data passed as-is

Description

reduceToSelected( ) reduces the current selection of rows in the Grid to the ones that are selected.

resetSortIndicator( )

void **resetSortIndicator**( )

Description

resetSortIndicator( ) resets the column's sort indicators (the two sort arrows in the header) for the columns in the Grid.

setReadOnly( )

void **setReadOnly**(Boolean *mode*)

Parameter	Type	Description
mode	Boolean	True = Read only mode; False = Read/write mode

Description

With **setReadOnly( )**, you can set the Grid to be in read only or read/write mode, which is a property in the Properties tab. If you pass *true* to *mode*, the Grid will be in read only mode (i.e., the Grid's columns will not be enterable) and the  and  buttons in the Grid's footer will be removed. If you pass *false* to *mode*, the Grid will be in read/write mode.

setRowHeight( )

void **setRowHeight** (rowHeight)

Parameter	Type	Description
rowHeight	Number	Height (in pixels) of the rows in the Grid

Description

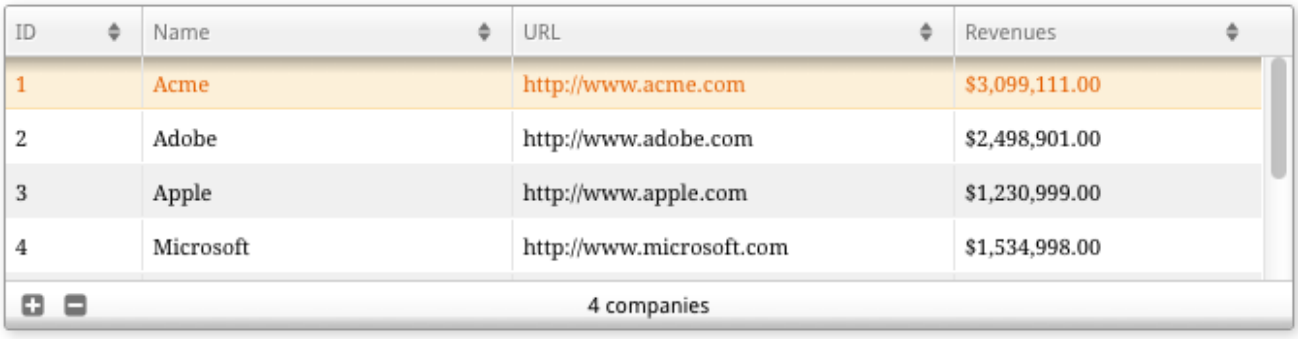
setRowHeight( ) allows you to set the height (in pixels) of the rows in the Grid widget. By default, the height of the row is 28.

Example

The following example allows you to set the row height to 40 pixels:

```
$$('dataGrid1').setRowHeight(40);
```

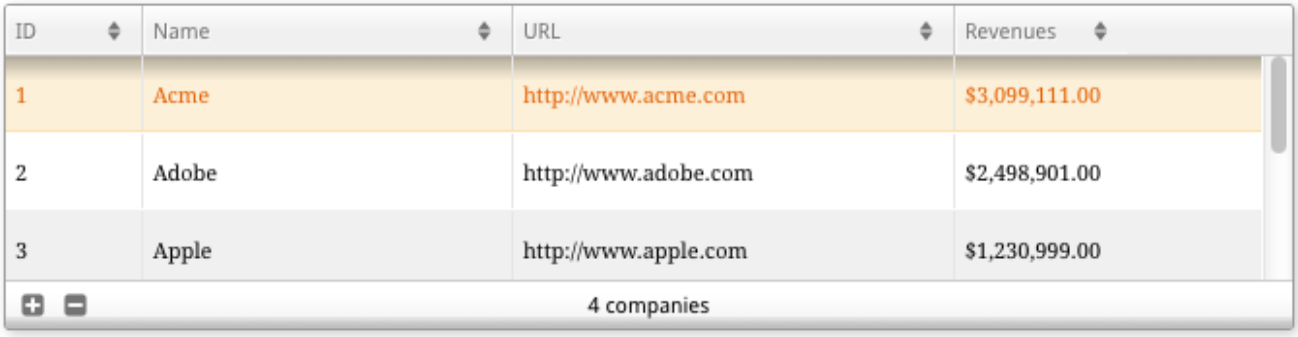
If originally, our Grid widget was as follows:



ID	Name	URL	Revenues
1	Acme	http://www.acme.com	$3,099,111.00
2	Adobe	http://www.adobe.com	$2,498,901.00
3	Apple	http://www.apple.com	$1,230,999.00
4	Microsoft	http://www.microsoft.com	$1,534,998.00

4 companies

The row height is increased to 40 as shown below:



ID	Name	URL	Revenues
1	Acme	http://www.acme.com	$3,099,111.00
2	Adobe	http://www.adobe.com	$2,498,901.00
3	Apple	http://www.apple.com	$1,230,999.00

4 companies

setSelectedRows( )

void **setSelectedRows**(Array *rows*)

Parameter	Type	Description
rows	Array	Rows to select in the Grid

Description

setSelectedRows( ) allows you to select a selection of rows in the Grid when the **Selection mode** is "multiple".
Row numbers start at 0.
Note: This widget is only available in the development branch for Wakanda.

Example

In the following example, we select the first three rows in the Grid:

```
$$('countryGrid').setSelectedRows([0,1,2]);
```

setSelectionMode( )

void **setSelectionMode**(String *mode*)

Parameter	Type	Description
mode	String	Pass "single" for single selection mode and "multiple" for multiple selection mode

Description

setSelectionMode( ) allows you to set the selection mode of the **Grid** widget to either "single" or "multiple". For more information, you can refer to the **Grid Properties** section.

Example

Set the Grid to multiple selection mode:

```
$$('employeeGrid').setSelectionMode("multiple");
```

To set it back to single selection mode:

```
$$('employeeGrid').setSelectionMode("single");
```

setSortIndicator( )

void **setSortIndicator** (column , sortOrder)

Parameter	Type	Description
column	Number	Column number (starting at 0 for the first column)
sortOrder	String	"asc" = ascending order; "desc" = descending order

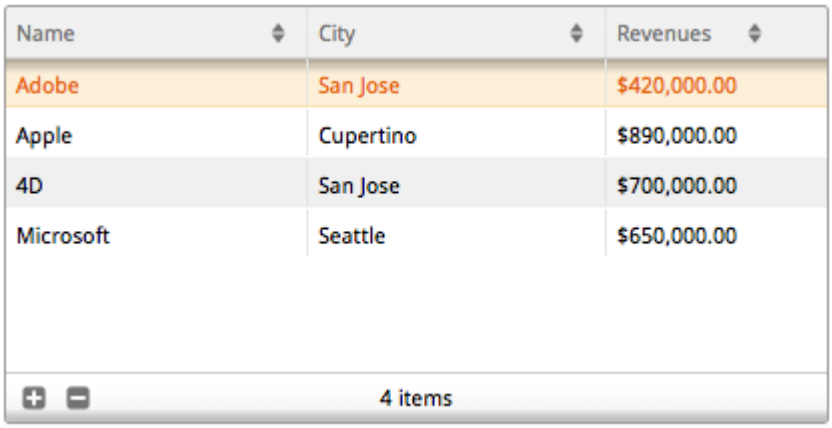
Description

setSortIndicator( ) allows you to define the sorting arrow in the header of the column in the Grid widget. This method does not actually sort the column.

The column numbers start at 0.

Example

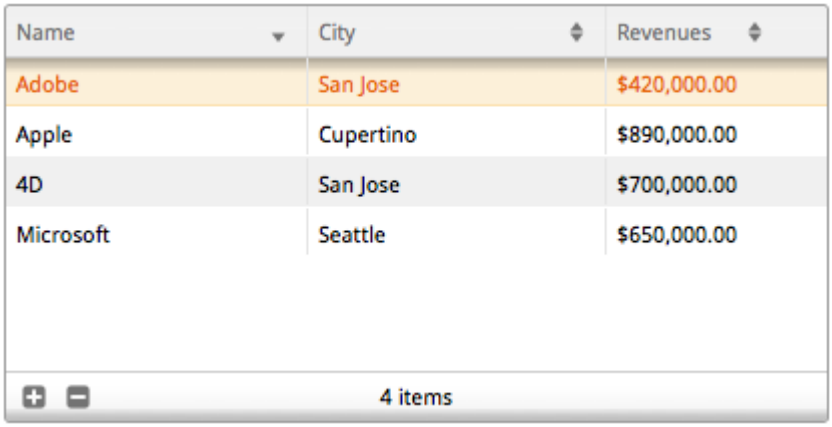
For example, if we have the following Grid displayed:



Once we make the following call:

```
$$('dataGrid1').setSortIndicator(0,"desc");
```

The Grid will appear as follows:



Note: The data in the column is not sorted.

Grid Column

A *column* object, which is a part of the **Grid** widget, is returned by the **column( )** function of the **Grid** class. Once you have a *column* object, you can call any of the functions in this class:

You can retrieve the *column* object in one of two ways:

```
var colObject;
colObject = $$("employeeGrid").column(2); //numbered starting from 1 on left
        // or
colObject = $$("employeeGrid").column("lastName"); //by the attribute it contains
```

Then, you can use any of the functions in this class as shown below:

```
var myValue = colObject.getFormattedValue();
```

getFormattedValue( )

String **getFormattedValue**

Returns String Formatted value in the column defined by column() and the selected row

Description

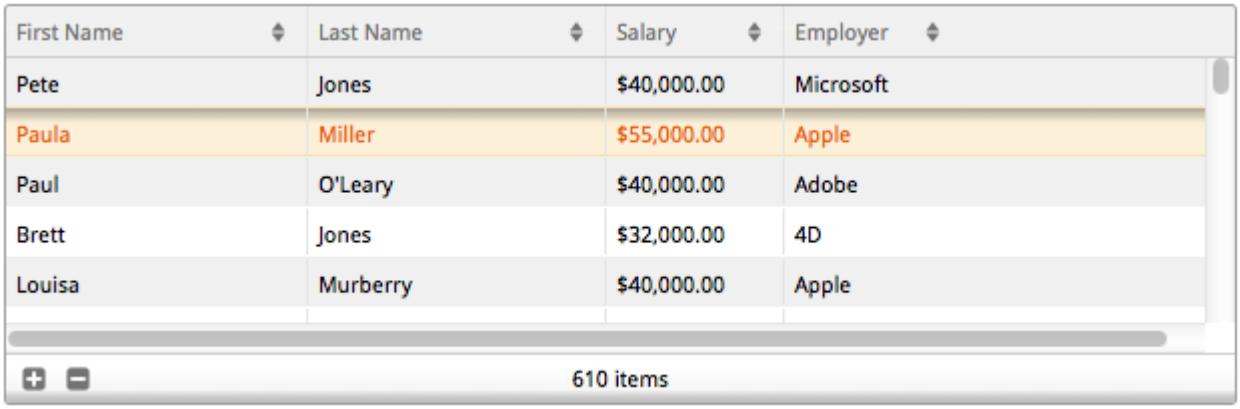
getFormattedValue( ) allows you to retrieve the formatted value in a cell defined by *column* and the currently selected row.

Example

In the Grid below, if we execute the following line of code:

```
$$('employeeGrid').column(3).getFormattedValue();
```

The value returned would be:
$55,000.00



If we execute the following line of code:

```
$$('employeeGrid').column(3).getValueForInput();
```

The value returned would be:
55000

getValueForInput( )

void **getValueForInput**

Description

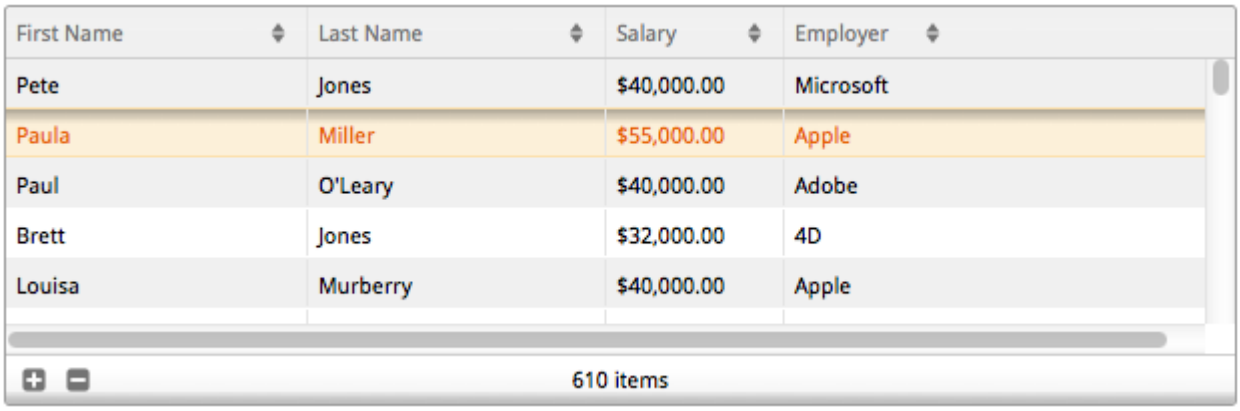
getValueForInput( ) allows you to retrieve the value in a cell defined by *column* and the currently selected row.

Example

In the Grid below, if we execute the following line of code:

```
$$('employeeGrid').column(3).getValueForInput();
```

The value returned would be:
55000



If we execute the following line of code:

```
$$('employeeGrid').column(3).getFormattedValue();
```

The value returned would be:
$55,000.00

setAlignment( )

void **setAlignment** (alignment)

Parameter	Type	Description
alignment	String	Alignment of the column: "left", "center", or "right"

Description

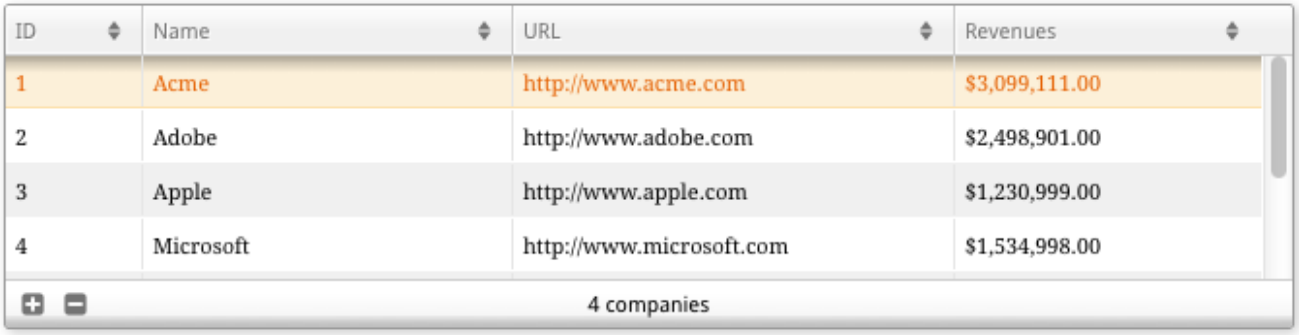
setAlignment( ) allows you to set the alignment of the column in a Grid widget. You can set it to either left, center, or right. The header text will also be centered along with the data in the column.

Example

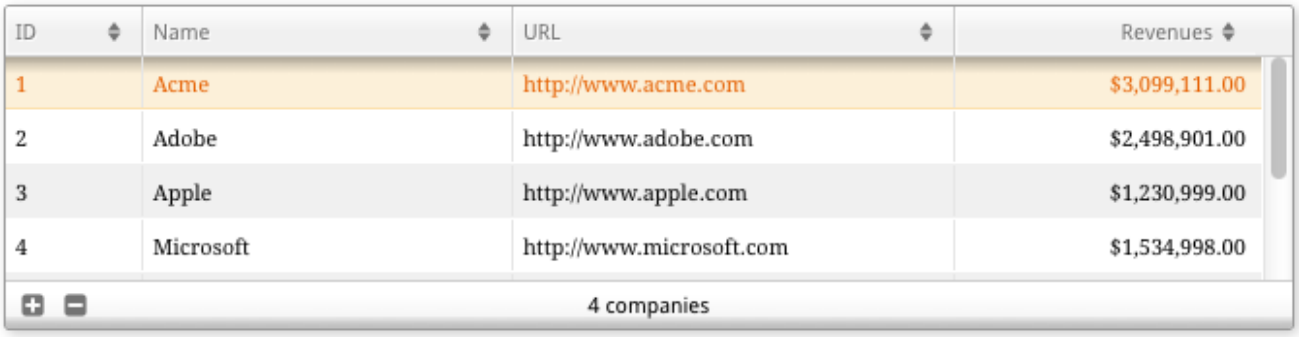
The following example sets the data in the "revenues" column to the right:

```
var col = $$('dataGrid1').column("revenues");
col.setAlignment("right");
```

When the Grid widget was originally as shown below:



After running the code above, the "revenues" column is aligned to the right:



setBackgroundColor( )

void **setBackgroundColor**([String *backgroundColor*])

Parameter	Type	Description
backgroundColor	String	Background color

Description

setBackgroundColor( ) allows you to set *column*'s background color *to backgroundColor*. This modification applies to the entire column along with its header.

backgroundColor can be expressed as:

- an HTML color value, i.e., "blue",
- a HEX value, i.e., "#CCC" or "#39C488", or
- a RGB value, i.e., #ff00ff.

Example

The following line sets the background color for the second column:

```
$$('employeeGrid').column(2).setBackgroundColor("#C30");
```

To reset it back to the background color it was defined with:

```
$$('employeeGrid').column(2).setBackgroundColor("");
```

setFormat( )

void **setFormat**(String *format*)

Parameter	Type	Description
format	String	Display format for the data in the column defined by column()

Description

setFormat( ) allows you to set the display format to *format* for the data in *column*.

Example

This example formats the Salary column in the Grid:

```
$$('employeeGrid').column("salary").setFormat("$###,###,##0.00");
```

The column will be formatted as shown below:

setRenderer( )

void **setRenderer**(Function *function*)

Parameter	Type	Description
function	Function	Function to call for each cell in the column

Description

setRenderer( ) allows you to call a function for each cell in a *column*. The parameter that you define in your function receives as its first parameter an object that contains the following information:

Property	Description
cellDiv	Cell's DIV tag
rowNumber	Number of the row in the column (starting at 0)
value	Value in the cell

For example, if you named your function's parameter *myCell*, *myCell.value* will contain the value in the cell. If you want to change it, you just need your function to return it.

You can apply either **addClass()** or **removeClass()** to the cellDiv property to add or remove a CSS class to the specific cell. See the third example.

Example

The following example modifies the data in the "lastName" column so that the text is in uppercase:

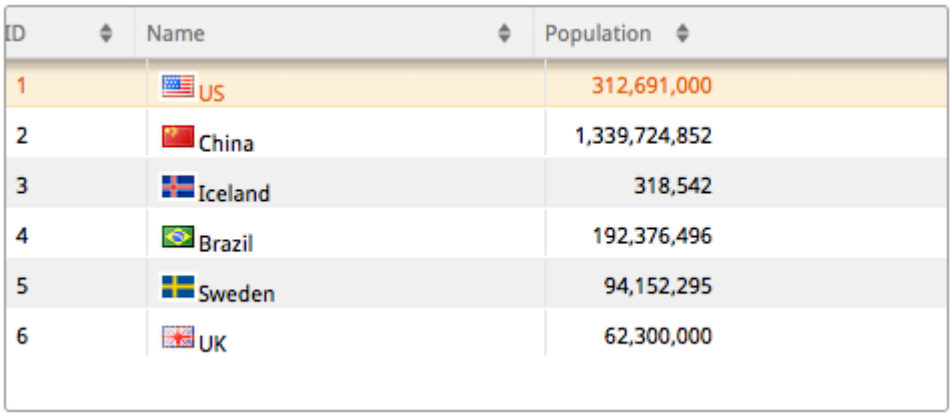
```
$$('employeeGrid').column('lastName').setRenderer(
    function(myCell) {
        return myCell.value.toUpperCase();
    }
);
```

Example

This example places a flag icon (whose filename is the same as the country) in the column next to the country name:

```
$$('countryGrid').column(2).setRenderer(
    function(myCell) {
      var myimage="images/country/"+myCell.value.toLowerCase()+".png";
      return "<img src="+myimage+" alt=\"\">"+ myCell.value;
    }
);
```

The result is the following:



Example

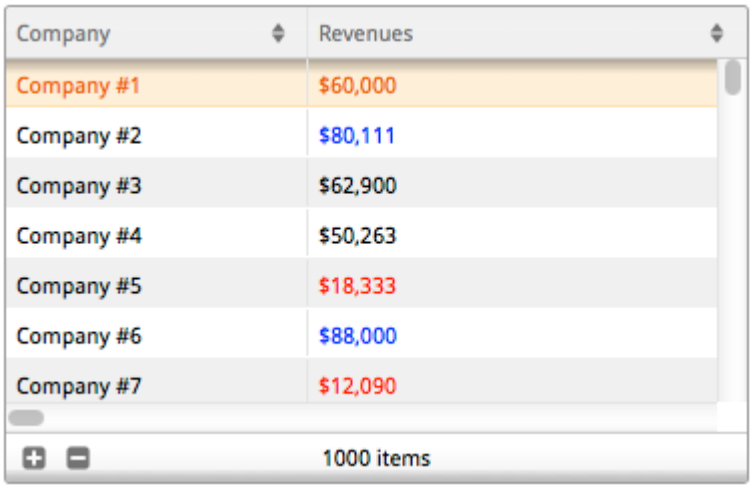
In this example, you can apply a CSS class to a specific cell. First, we create the function that will apply the "highRevenues" CSS class to the cell whose value is greater than or equal to 75000 and the "lowRevenues" class to the cell whose value is less than or equal to 25000:

```
function showLowAndHighRevenues(myCell) {
    var theRev = myCell.value;
    if(theRev >= 75000) {
        myCell.cellDiv.addClass('highRevenues'); //both of these CSS classes were defined
    } else if(theRev <= 25000) {
        myCell.cellDiv.addClass('lowRevenues');
    } else {
        myCell.cellDiv.removeClass('highRevenues');
        myCell.cellDiv.removeClass('lowRevenues');
    }
}
```

Afterwards, we call the following line to apply our function to the Grid:

```
$$('companyInfoGrid').column('revenues').setRenderer( showLowAndHighRevenues );
```

The result is the following:



setTextColor( )

void **setTextColor**([String *textColor*])

Parameter	Type	Description
textColor	String	Text color

Description

setTextColor( ) allows you to set *column*'s text color *to textColor*. This modification applies to the entire column along with its header.

textColor can be expressed as:

- an HTML color value, i.e., "blue",
- a HEX value, i.e., "#CCC" or "#39C488", or
- a RGB value, i.e., #ff00ff.

Example

The following example sets the color of the text in the second column of the Grid:

```
$$('employeeGrid').column(2).setTextColor("#C30");
```

To reset the color back to the one defined for the Grid:

```
$$('employeeGrid').column(2).setTextColor("");
```

setTextSize( )

void **setTextSize**(Number *textSize*)

Parameter	Type	Description
textSize	Number	Size of the text in pixels (e.g., 12 for 12px)

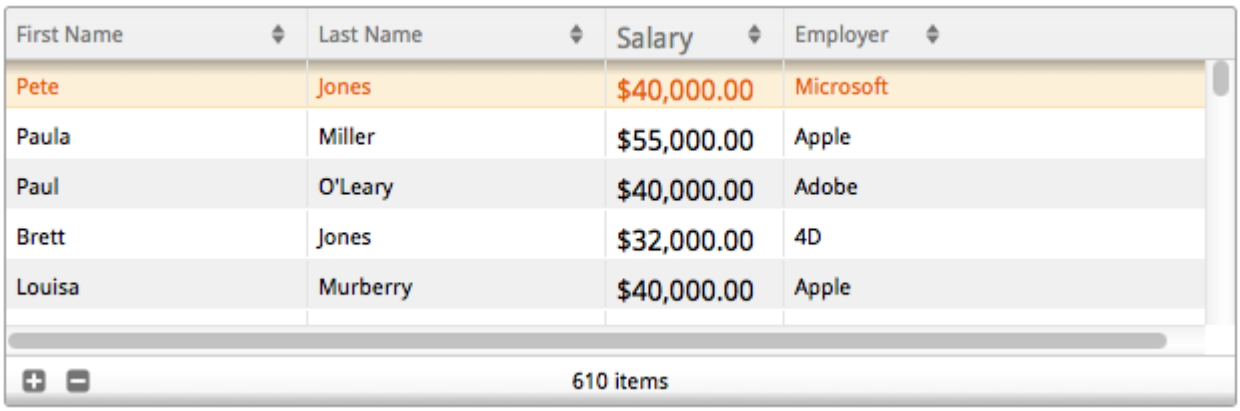
Description

setTextSize( ) allows you to set the text size of the column to *textSize* in your Grid widget. When you set the text size of the column, the header is also modified.

Example

The following example sets the text size for the third column to 14 pixels:

```
$$('employeeGrid').column(3).setTextSize(14);
```



setWidth( )

void **setWidth**(Number *width*)

Parameter	Type	Description
width	Number	Width of the column in pixels

Description

setWidth( ) sets the column width to *width* (expressed in pixels) for *column*.

Example

This example sets the width of the third column to 150 pixels:

```
$$('employeeGrid').column(3).setWidth(150);
```

Icon

The **Icon** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Image

The **Image** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Image Button

The **Image Button** widget inherits from the Widget class; however, not all the properties and functions are available to it.

This widget has one function, **getIcon( )**, which returns the Icon widget.

getIcon( )

Object **getIcon**( )

Returns	Object	Icon widget included in the Image Button widget

Description

getIcon( ) returns the Icon widget as an object for the Image Button widget. Once you retrieve the Icon widget, you can apply other functions in the Widget class to it.

Example

The following example sets the Image Button's text color from black to dark red:

```
$$('imageButton1').getIcon().setLabelTextColor('#9C0');
//the text in an Image Button widget is the label of the Icon widget
```

Login Dialog

Besides the methods in the **Widget** class for the **Login Dialog** widget, the methods below are specific to this widget:

- **login( )**: Login a user into the application by passing the *username* and *password*.
- **showLoginDialog( )**: Display the login dialog for the Login Dialog widget.
- **logout( )**: Log the current user out of the application.
- **refresh( )**: Refresh the display of the current user in the Login Dialog widget.

login( )

void **login**(String *username*, String *password*)

Parameter	Type	Description
username	String	Username of the user to login
password	String	Password for the user to login

Description

With this method, you can pass the *username* and *password* to login a user in the **Login Dialog** widget.

Example

The following example allows you to login using the Login Dialog widget without displaying the dialog to enter the username and password:

```
$$('login1').login("jsmith","3b97c1a5");
```

Once the user has successfully logged in, the Login Dialog widget displays the information defined in the Login Status section on the Properties tab:

logout( )

void **logout**( )

Description

logout( ) allows you to log the current user out of the application through the **Login Dialog** widget displayed on the Interface page.

Example

Log the current user out of the application:

```
$$('login1').logout();
```

Once the user has been logged out, the information in the Login Dialog section of the Properties tab appears.

refresh( )

void **refresh**( )

Description

refresh( ) allows you to refresh the **Login Dialog** widget to display the current user.

Example

Refresh the Login Dialog widget:

```
$$('login1').refresh();
```

showLoginDialog( )

void **showLoginDialog**( )

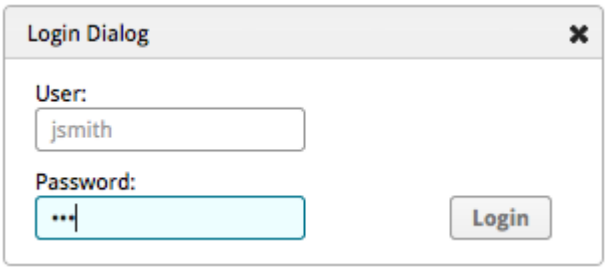
Description

This method allows you to display the login dialog so that the user can enter his/her username and password for the **Login Dialog** widget.

If you make the following call:

```
$$('login1').showLoginDialog();
```

The following dialog appears:



Matrix

Besides the methods in the **Widget** class for the **Matrix** widget, the methods below are specific to this widget:

- **getCurrentPage( )**: Retrieves the current page number displayed in the Matrix widget.
- **getDisplayedRow( )**: Get the displayed row in Matrix widget.
- **getTotalPages( )**: Get the total number of pages in the Matrix widget.
- **goTo( )**: Display the *nth* element in the Matrix widget.
- **goToFirst( )**: Display the first element in the Matrix widget.
- **goToLast( )**: Display the last element in the Matrix widget.
- **goToNextPage( )**: Display the next page in the Matrix widget.
- **goToPreviousPage( )**: Display the previous page in the Matrix widget.

getCurrentPage( )

Number **getCurrentPage**( )

Returns	Number	Current page displayed in the Matrix

Description

getCurrentPage( ) returns the current page number in the **Matrix** widget. The first page is numbered as 0.

Example

This example returns the current page displayed for Matrix.

```
var currentPage=$$('matrix1').getCurrentPage();
```

getDisplayedRow( )

Number **getDisplayedRow**( )

Returns	Number	Currently displayed row in the Matrix widget

Description

This method returns the currently displayed row in the **Matrix** widget.

getTotalPages( )

Number **getTotalPages**( )

Returns	Number	Number of pages in the Matrix widget

Description

This method returns the total number of pages in the **Matrix** widget.

goTo( )

void **goTo**(Number *element*)

Parameter	**Type**	**Description**
element	Number	Go to a specific element in the Matrix

Description

goTo( ) allows you to go to a specific *element* in the **Matrix** widget.

Example

If you have 100 elements in the Matrix widget, you can display the 30th element by doing the following:

```
$$('matrix1').goTo(30);
```

goToFirst( )

void **goToFirst**( )

Description

goToFirst( ) allows you to display the first element in the **Matrix** widget.

goToLast( )

void **goToLast**( )

Description

goToLast( ) allows you to display the last element in the **Matrix** widget.

goToNextPage( )

void **goToNextPage**( )

Description

goToNextPage( ) allows you to display the next page in the **Matrix** widget.

goToPreviousPage( )

void **goToPreviousPage**( )

Description

goToPreviousPage( ) allows you to display the previous page in the **Matrix** widget.

Menu Bar

The **Menu Bar** widget inherits from the Widget class; however, not all the properties and functions are available to it.

It does, however, have two functions specific to it:

- **renameMenuItem( )**: Modify the text in a specific Menu Item.
- **getSelectedMenuItem( )**: Returns the Menu Item widget selected for a Menu Bar widget when "On Mouse Click" is selected for the **Show Submenus** property.

getSelectedMenuItem( )

Object **getSelectedMenuItem**( )

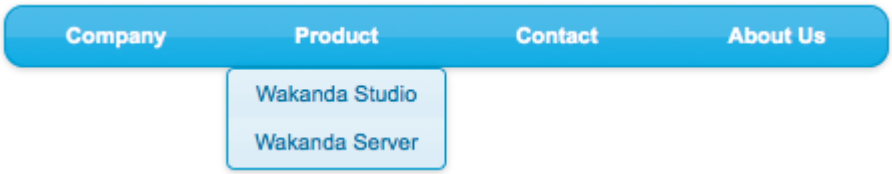
Returns	Object	Object for the selected menu item in the Menu Bar widget

Description

The **getSelectedMenuItem( )** function allows you to retrieve the currently selected Menu Item widget in the **Menu Bar** widget. This function only works for a Menu Bar widget that has "On Mouse Click" selected for the **Show Submenus** property.

Example

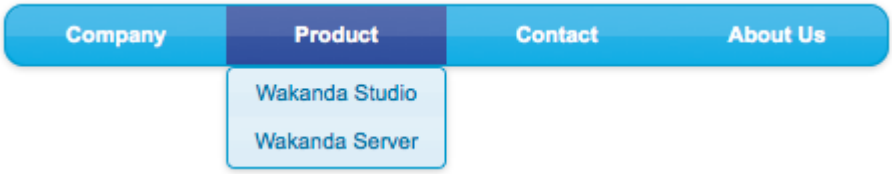
In the following Menu Bar widget and you select the second menu item:



When you call the following line of code:

```
$$('menuBar1').getSelectedMenuItem().setBackgroundColor("#349");
```

The same Menu Bar appear as follows:



To set the background color back to the one defined by default, you can write:

```
$$('menuBar1').getSelectedMenuItem().setBackgroundColor("");
```

renameMenuItem( )

void **renameMenuItem** (menuItemText)

Parameter	Type	Description
menuItemText	String	Text to display for the Menu Item

Description

The **renameMenuItem( )** function allows you to modify the title of a Menu Item, which is a part of the **Menu Bar** widget.

The **Tab View** widget is also made up of a Menu Bar widget and each tab is a Menu Item.

Example

In the following example, we retrieve the IDs that are a part of the tab and modify them:

```
var tabObject = $$('tabView1').getSelectedTab();
var tabNumber = tabObject.index //returns the tab number
$$(tabObject.menuItem.id).renameMenuItem("My Tab"); //Menu bar item ID
```

```
$$(tabObject.container.id).setBackgroundColor("#9c0"); //Container ID
```

Navigation View

The **Navigation View** widget inherits from the Widget class; however, not all the properties and functions are available to it.

This widget also has its own set of functions:

- **goToView( )**: Go to a specific view in the Navigation View widget.
- **goToPreviousView( )**: Go to the previous view in the Navigation View widget.

goToPreviousView( )

void **goToPreviousView**( )

Description

goToPreviousView( ) allows you to go to the previous view in the Navigation View widget.

Example

To go to the previous view, you write the following:

```
$$('navigationView1').goToPreviousView();
```

goToView( )

void **goToView** (view)

Parameter	Type	Description
view	Number	View index to go to in the Navigation View

Description

goToView( ) allows you to go to a specific view in the Navigation View widget defined by the **Index** property in the Views section.

Example

To go to the third view (whose Index property is 3), you write the following:

```
$$('navigationView1').goToView(3);
```

Progress Bar

These methods can be applied to the **Progress Bar** widget:

- **startListening( )**: Start sending requests to the server in order to display the progress of the session.
- **stopListening( )**: Stops sending requests to the server.
- **userBreak( )**: Sends a request to the server to interrupt the session.

startListening( )

void **startListening**( )

Description

startListening( ) begins sending requests to the server in order to display the progress of the session associated with the **Progress Bar** widget.

Example

The following example shows how to start and stop listening to the requests sent to the server:

```
$$("progressBar1").startListening();
ds.Company.callMethod({ method:"searchCompanies", onSuccess: function()
    {
        $$("progressBar1").stopListening();
    }
 }, searchCriteria, "progressBarRef");
```

For more information about how to set up a Progress Indicator, refer to the **ProgressIndicator( )** method.

stopListening( )

void **stopListening**( )

Description

stopListening( ) stops sending requests to the server in order to display the progress of the session associated with the widget.

userBreak( )

void **userBreak**( )

Description

userBreak( ) sends a request to the server to interrupt the session.

Example

The following example interrupts the session that progressBar1 is listening to:

```
$$("progressBar1").userBreak();
```

Query Form

Besides the methods in the **Widget** class that you can apply to a **Query Form** widget, you can also use the following method, which is specific to the Query Form widget:

- **findEntity( )**: Query the datasource with the data entered in the fields of the Query Form.

To clear the values in the Query Form, use the **clear( )** function.

findEntity( )

void **findEntity**( )

Description

The findEntity( ) function allows you to find the entities whose values were entered in the fields of the Query Form widget. You can also use the **clear( )** function to clear the values in the Query Form.

Radio Button Group

The **Radio Button Group** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Select

The **Select** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Slider

Besides the methods in the **Widget** class for the **Slider** widget, the following functions are available to you:

- **addHandle( )**: add a handle to the Slider
- **getMin( )**: get Minimum Value property set for the Slider
- **getMax( )**: get Maximum Value property set for the Slider
- **getOrientation( )**: get Orientation property set for the Slider
- **getRange( )**: get Range property set for the Slider
- **getStep( )**: get Step property set for the Slider
- **setMin( )**: set Minimum Value property set for the Slider
- **setMax( )**: set Maximum Value property set for the Slider
- **setOrientation( )**: set Orientation property set for the Slider ("Horizontal" or "Vertical")
- **getRange( )**: set Range property set for the Slider
- **setStep( )**: set Step property set for the Slider
- **setValues( )**: set the values for the Slider

Note: These functions are only available in the Development Branch of Wakanda.

addHandle( )

void **addHandle**(Number *number*)

Parameter	Type	Description
number	Number	Position of the handle

Description

Note: This function is only available in the Development Branch of Wakanda.

With **addHandle( )**, you can add an additional handle to the **Slider** widget on your Interface page. The value passed to *number* must be between the minimum and maximum values defined for the Slider widget. You can only add one additional handle to a Slider widget.

When you call the **getValue( )** method for the Slider widget, it returns an array with the value of each handle. If you only have one handle, only one value will be returned.

Example

If our Slider widget has a handle at 50000, we can add a second handle so that we can define a range instead of just one value:

```
$$('salarySlider').addHandle(20000);
```

For our example below, the **getValue( )** method returns: [20000, 50000].

getMax( )

Number **getMax**( )

Returns	Number	Maximum Value defined in the Slider properties

Description

getMax( ) returns the **Maximum Value** for the Slider widget either as defined in the **Slider Properties** or specified using the **setMax( )** function.

getMin( )

Number **getMin**( )

Returns	Number	Minimum Value defined in the Slider properties

Description

getMin( ) returns the **Minimum Value** for the Slider widget either as defined in the **Slider Properties** or specified using the **setMin( )** function.

getOrientation( )

String **getOrientation**( )

Returns	String	Orientation of the Slider widget: "horizontal" or "vertical"

Description

getOrientation( ) returns the Orientation of the Slider widget (either set in the **Slider Properties** or in your code).

getRange( )

String **getRange**( )

Returns	String	Range of the slider: "min", "max", or "none"

Description

getRange( ) returns the **Range** of the Slider widget:"min", "max", or "none". Either as it is defined in the **Slider Properties** or specified by using the **setRange( )** function.

getStep( )

Number **getStep**( )

Returns	Number	Step as defined in the Slider properties

Description

getStep( ) returns the **Step** for the Slider widget either as defined in the **Slider Properties** or as specified by using the **setStep( )** function.

setMax( )

void **setMax**(Number *value*)

Parameter	**Type**	**Description**
value	Number	Maximum value

Description

setMax( ) allows you to set the **Maximum Value** for the Slider and overrides the one defined in the **Slider Properties**.

setMin( )

void **setMin**(Number *value*)

Parameter	**Type**	**Description**
value	Number	Minimum value

Description

setMin( ) allows you to set the **Minimum Value** for the Slider and overrides the one defined in the **Slider Properties**.

setOrientation( )

void **setOrientation** (orientation)

Parameter	Type	Description
orientation	String	Orientation of the Slider widget: "horizontal" or "vertical"

Description

With **setOrientation( )**, you can set the Orientation of the Slider widget.

setRange( )

void **setRange** (range)

Parameter	Type	Description
range	String	Range of the slider: "min", "max", or "none"

Description

setRange( ) allows you to set the **Range** of the Slider widget. Pass "min", "max", or "none".

setStep( )

void **setStep**(Number *step*)

Parameter	Type	Description
step	Number	Step for the Slider widget

Description

setStep( ) allows you to set the **Step** for the Slider widget.

setValues( )

void **setValues**(Array *values*)

Parameter	Type	Description
values	Array	Values to set both handles of a Slider widget

Description

setValues( ) allows you to set the two values for a Slider that has two handles.

When you use the **addHandle( )** function to add a second handle to the Slider widget, you can set both values by using this function.

Example

The following example sets the two values for the Slider widget (having two handles):

```
$$('slider1').setValues([20,300])
```

Split View

The **Split View** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Switch

The **Switch** widget inherits from the Widget class; however, not all the properties and functions are available to it. To retrieve the value of the Switch widget, use the **getValue( )** function and to set the value, use the **setValue( )** function.

This widget also has its own set of functions:

- **slide( )**: Slide the switch to "on" or "off".
- **toggleSlide( )**: Toggle the switch to either "on" or "off".

slide( )

void **slide**(Boolean *state*)

Parameter	Type	Description
state	Boolean	True = slide switch to "on"; False = slide switch to "off"

Description

slide( ) allows you to set the Switch widget to either "on" or "off" by passing either *true* or *false*.

toggleSlide( )

void **toggleSlide**( )

Description

toggleSlide( ) allows you to toggle the Switch widget to either "on" or "off" depending on its current state. If the Switch widget is set to "on" (true), it will be set to "off" (false).

Tab View

The **Tab View** widget inherits from the Widget class; however, not all the properties and functions are available to it.

This widget also has its own set of functions:

- **addTab( )**: Add a new tab the Tab View widget.
- **countTabs( )**: Retrieve the number of tabs in the Tab View widget.
- **getSelectedTab( )**: Get the currently selected tab's number, Menu Item widget ID and Container widget ID.
- **getTabContainer( )**: Get the tab's Container widget ID.
- **removeTab( )**: Remove a tab from the Tab View widget.
- **selectTab( )**: Select a specific tab.

Note: This widget is only available in the development branch for Wakanda.

addTab( )

void **addTab** (title , closable)

Parameter	Type	Description
title	String	Title of the tab to add
closable	Boolean	True = Closable tab; False = Not closable tab

Description

The **addTab( )** function allows you to add a tab to the **Tab View** widget by passing its *title* as parameter. You can also define if the tab has a close box on it by passing True to *closable*.

countTabs( )

Number **countTabs**( )

Returns	Number	Number of tabs currently in the Tab View widget

Description

The **countTabs( )** function returns the number of tabs currently displayed in the **Tab View** widget.

getSelectedTab( )

Object **getSelectedTab**( )

Returns	Object	Object defining the currently selected tab in the Tab View widget

Description

The getSelectedTab( ) function returns an object that defines the currently selected tab in the **Tab View** widget.

The object that is returned contains three properties:

- **index**: a number representing the currently selected tab
- **menuItem**: an object that defines Menu Item widget for the tab
- **container**: an object that defines the container for the tab

The **id** property in the tab and container objects is the main one that can be useful to you.

Example

In the following example, we retrieve the IDs that are a part of the tab and modify them:

```
var tabObject = $$('tabView1').getSelectedTab();
var tabNumber = tabObject.index //returns the tab number
$$(tabObject.menuItem.id).renameMenuItem("My Tab"); //Menu bar item ID
$$(tabObject.container.id).setBackgroundColor("#9c0"); //Container ID
```

getTabContainer( )

Object **getTabContainer**(Number *tabNumber*)

Parameter	Type	Description
tabNumber	Number	Tab number in the Tab View widget
Returns	Object	Container object for the tab in the Tab View widget

Description

The **getTabContainer( )** function returns the Container object for the tab specified by *tabNumber* in the **Tab View** widget.

The **id** property can then be passed to other functions in the Widgets API.

The following example shows you how to retrieve the ID for the Container widget that makes up the Tab View widget.

```
var containerID = $$('tabView1').getTabContainer(2).id;
```

removeTab( )

void **removeTab**(Number *tab*)

Parameter	Type	Description
tab	Number	Number of the tab to remove

Description

The **removeTab( )** function allows you to remove a tab displayed in the **Tab View** widget defined by the *tab* parameter. The tab number is defined by the tabs that are currently displayed, and are numbered from left to right or from top to bottom.

selectTab( )

void **selectTab**(Number *tab*)

Parameter	Type	Description
tab	Number	Number of the tab

Description

The **selectTab( )** function allows you to select a specific tab and display its contents in the **Tab View** widget. The tab number is defined by the tabs that are currently displayed, and are numbered from left to right or from top to bottom.

Text

The **Text** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Text Input

The **Text Input** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Video

The **Video** widget inherits from the Widget class; however, not all the properties and functions are available to it.

Widget

All of Wakanda's widgets inherit the properties and methods in the **Widget** class. However, not all the properties and methods are available to all widgets: some widgets do not make use of certain methods or properties and therefore do or return nothing.

There are also other instances in which a property does not exist for a particular widget in the GUI Designer, yet you can still set it through the API. For example, the **Combo Box** widget does not have the Draggable property; however, you can still make it draggable by calling **draggable( )**.

config

Description

This property returns an object that defines all the widget's properties. If you have created your own widget, all the properties with the "data" prefix are also included.

Each widget will return different properties. The following properties are common to most widgets (if they are available in the widget's Properties tab):

Property	Description	Example
class	CSS classes	"waf-widget waf-textField default inherited "
data-binding	Source/Source In property	"employee.salary" or "employee"
data-draggable	Draggable property	null
data-errorDiv	Display Error property	myErrorDiv
data-format	Format property	"$###,###,#00.00"
data-label	Label property	"Salary"
data-label-position	Position of label (in Styles tab)	"left"
data-lib	Data Library	"WAF"
data-resizable	Resizable property	null
data-type	Widget type	"textField"
id	ID property	"textField1"
tabindex	Tabindex property	null
type	Type of data	"text"
value	Value	null

Note: All Boolean values return either "true" or null. By default, any string value left blank returns null.

Below are the properties per widget when you make a call to **config:**

```
var myObject = $$('widgetID').config;
```

Auto Form

Besides the *class*, *id*, *data-binding*, *data-errorDiv, data-draggable*, *data-resizable*, *data-type,* and *data-lib* properties, the Auto Form returns the following properties:

Property	Description	Example
data-column	Columns	"[{'title':'First Name','sourceAttID':'firstName'}, {'title':'Last Name','sourceAttID':'lastName'}, {'title':'Gender','sourceAttID':'gender'}]"
data-column-attribute	Attributes	"firstName,lastName,fullName,gender"
data-column-name	Attribute Titles	"First Name,Last Name,Gender"

data-columns	Public Attributes in Datasource	"ID,firstName,lastName,fullName,gender,telephone, birthday,salary,photo,employer,employerName"
data-display-error	Display Error	"true"
data-resize-each-widget	Allow to resize each widget property	"true"
data-withoutTable	With Included widgets property	"true"

The object returned for a column in an Auto Form has the following properties:

Property	Description
title	Label for the attribute
sourceAttID	Attribute property

Button widget

Besides the *class*, *id*, *data-binding*, *data-type,data-lib*,and *tabindex* properties, the Button widget returns the following properties:

Property	Description	Example
data-action	Action property	"simple"
data-link	Link property	null
data-state-1	Default state icon (Styles tab)	"/images/002_47.png"
data-state-2	Hover state icon (Styles tab)	"/images/002_48.png"
data-state-3	Active state icon (Styles tab)	"/images/002_49.png"
data-state-4	Disabled state icon (Styles tab)	"/images/002_50.png"
data-target	Target property	"_blank"
data-text	Text property	null

Checkbox widget

Besides the *class*, *id*, *data-binding*, *data-errorDiv, data-type, data-label, data-label-position, data-lib*,and *tabindex* properties, the Checkbox widget returns the following properties:

Property	Description	Example
data-checked	Checked property	true
data-link	Link property	null
data-icon-active	Active state icon (Styles tab)	null
data-icon-default	Default state icon (Styles tab)	"/images/002_48.png"
data-icon-hover	Hover state icon (Styles tab)	null
data-icon-selected	Selected state icon (Styles tab)	null

Combo Box widget

Besides the *class*, *id*, *data-binding*, *data-type, data-label, data-label-position, data-lib*,and *tabindex* properties, the Combo Box widget returns the following properties:

Property	Description	Example
data-autoDispatch	Auto Dispatch property	"true"
data-binding-key	Key property	"name"
data-binding-options	Attributes properties	"[name] "
data-binding-out	Source Out property	"country.name"
data-editable	Autocomplete property	"true"

Component widget

Besides the *class*, *id*, *data-draggable*, *data-resizable*, *data-type*, and *data-lib* properties, the Component widget returns the following properties:

Property	Description	Example
data-modal	Modal property	"false"
data-start-load	Load by default property	null

Grid widget

Besides the *class*, *id*, *data-binding*, *data-errorDiv, data-type, data-label, data-label-position, data-lib*, *data-resizable*, and *data-draggable* properties, the Grid widget returns the following properties:

Property	Description	Example
data-column	Columns	"[{'sourceAttID':'fullName','colID':'fullName','width':'150', {'sourceAttID':'fromTime','colID':'fromTime','width':'100', {'sourceAttID':'toTime','colID':'toTime','width':'100','title':
data-display-error	Display Error property	"true"
data-footer-hide	Hide footer property	null
data-readOnly	Read only property	null
data-selection-mode	Selection mode property	"single"

The object returned for a column in a Grid has the following properties:

Property	Description
sourceAttID	Attribute property
colID	ID for the column (based on Attribute property)
width	Width of the column
title	Label for the column
format	Format of the data (only appears if it was defined)
readOnly	Read Only property for the column (only appears if it was checked)

Image widget

Beisdes the *class*, *id*, *data-binding*, *data-label*, *data-label-position*, *data-type*, and *data-lib* properties, the Image widget returns the following properties:

Property	Description	Example
data-fit	Fit property (0=to container, 1=to width, 2=to height, 3=container to image, 4=to container (proportionately))	"4"
data-link	Link property	null

data-src	Src property	null
data-target	Target property	"_blank"

Matrix widget

Besides the *class*, *id*, *data-draggable*, *data-resizable*, *data-lib*, *data-scrollbar*, and *data-type* properties, the Matrix widget returns the following properties:

Property	Description	Example
data-fit	Auto Fit property	"false"
data-margin	Margin property	null
data-scrollbar	Maximum Value property	"true"
data-scrolling	Scrolling property	"vertical"

Menu Bar widget

Besides the *class*, *id*, *data-lib,* and *data-type* properties, the Menu Bar widget returns the following properties:

Property	Description	Example
data-display	Display property	"horizontal"
data-subMenuShow	Show Submenus property	"hover"
data-tab-margin	Margin property	null

Query Form

Beisdes the *class*, *id*, *data-binding*, *data-draggable*, *data-resizable*, *data-type*, and *data-lib* properties, the Query Form widget returns the following properties:

Property	Description	Example
data-column	Attributes	"[{'title':'ID','sourceAttID':'ID'}, {'title':'lastName','sourceAttID':'lastName'}]"
data-column-attribute	Attribute property	"ID,lastName"
data-column-name	Attribute title property	"ID,lastName"
data-columns	Public attributes in datasource	"ID,firstName,lastName,fullName,gender,telephone, birthday,salary,photo,employer,employerName"
data-withoper	Show Operators property	null

Radio Group widget

Beisdes the *class*, *id*, *data-binding*, *data-label*, *data-label-position*, *data-draggable*, *data-resizable*, *data-type*, *tabindex*, and *data-lib* properties, the Radio Group widget returns the following properties:

Property	Description	Example
data-autoDispatch	Auto Dispatch property	"true"
data-binding-key	Key property	"name"
data-binding-options	Atttriutes Display property	"[name] "
data-binding-out	Source Out property	null

data-display	Display property	"vertical"
data-icon-active	Active state icon (Styles tab)	null
data-icon-default	Default state icon (Styles tab)	"/images/002_48.png"
data-icon-hover	Hover state icon (Styles tab)	null
data-icon-selected	Selected state icon (Styles tab)	null

Slider widget

Besides the *id*, *data-binding*, *data-errorDiv*, *data-type*, *data-label*, and *data-label-position*, the Slider widget returns the following properties:

Property	Description	Example
data-maxValue	Maximum Value property	"200000"
data-minValue	Minimum Value property	"10000"
data-orientation	Orientation property	"horizontal"
data-range	Range property	"min"
data-step	Step property	"10000"

Tab View widget

Besides the *class*, *id*, *data-type, data-draggable*, *data-resizable*, and *data-lib* properties, the Tab View widget returns the following properties:

Property	Description	Example
data-menu-position	Tab Position property	"false"
data-padding	Padding property	null

Text widget

Besides the *class*, *id*, *data-binding*, *data-format, data-label*, *data-label-position*, *data-type*, and *data-lib* properties, the Text widget returns the following properties:

Property	Description	Example
data-autoWidth	Auto resize property	"true"
data-link	Link property	null
data-overflow	Scrollbar property	"Horizontal"
data-target	Target property	"_self"
data-text	Text property (when no datasource is defined)	null

Text Input widget

Besides the *class*, *id*, *data-binding*, *data-format, data-errorDiv, data-label*, *data-label-position*, *data-type*, *tabindex*, and *data-lib* properties, the Text Input widget returns the following properties:

Property	Description	Example
data-autocomplete	Auto-Complete property	null
data-datapicker-icon-only	Display calendar when icon is clicked property	"false"
data-datapicker-on	Allow calendar for dates	"true"
data-multiline	Multi-Line property	"false"
data-password	Password field property	"false"

domNode

Description

The DOM node is the actual HTML tag created for the widget.

Example

If we try to find out the following proeprty of our widget:

```
var myDomNode = $$('textField5').domNode;
```

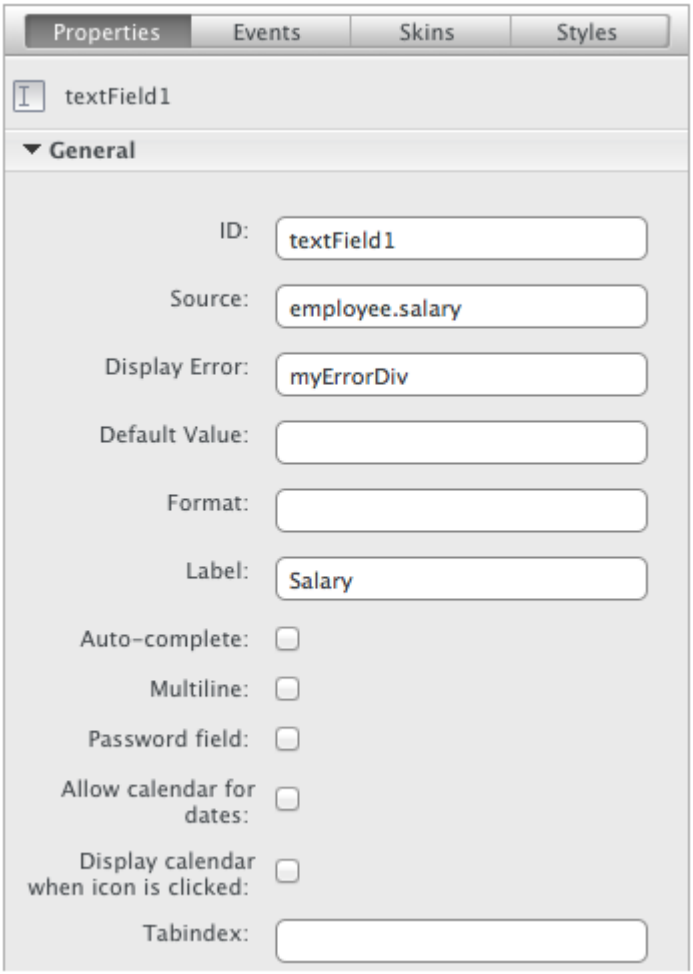
The following text will be returned:

```
<input id="textField5" class="waf-widget waf-textField default inherited AF_InputOK" type
data-binding="employee.salary" data-constraint-right="false" data-constraint-bottom="fals
data-constraint-left="true" data-datapicker-icon-only="false" data-datapicker-on="true" d
data-label-position="left" data-label="salary" data-lib="WAF" data-type="textField" name=
```

errorDiv

Description

You can define the Display Error ID, which corresponds to the **Display Error** widget, in the widget's Properties tab like the one shown below for the Text Input widget:



Example

If we want to find out the value in the **Display Error** property, we write:

```
var myErrorDivID = $$('textField1').errorDiv;  // returns "myErrorDiv"
```

format

Description

This property returns an object containing the format entered in the widget's **Format** property in the Properties tab.

Example

If we want to find out the format for our Text Input widget, we can write:

```
var myFormatObj = $$('textField1').format;
```

myFormatObj will contain one property:

```
{ format="$###,###,##0.00"}
```

id

Description

This property contains the widget's ID as defined in the Properties tab.

kind

Description

The kind property returns the type of widget. Below is a table with the values returned and the actual widget name displayed in Wakanda:

Widget Name	Kind
Auto Form	autoForm
Chart	chart
Checkbox	checkbox
Combo Box	combobox
Component	component
Container	container
Grid	dataGrid
Button	button
Display Error	errorDiv
Image	image
Login Dialog	login
Matrix	matrix
Menu Item	menuItem
Menu Bar	menuBar
Query Form	queryForm
Progress Bar	progressBar
Radio Button Group	radioGroup
Slider	slider
Text	richText
Tab View	tabView
Text Input	textField

Below is the list of Experimental widgets:

Widget Name	Kind
File Upload	fileUpload
Google Chart	googleChart
Google Maps	googleMap
Yahoo! Weather	yahooWeather

label

Description

When you add text in the **Label** property for a widget, a Label widget is created. This property contains the HTML tag for the Label widget.

If you make the following call for a Text Input widget:

```
var myLabel = $$('textField1').label;
```

myLabel returns the HTML tag for the Label widget:

```
<label id="label2" class="waf-widget waf-label default inherited " data-constraint-top="t
data-constraint-left="true" data-valign="middle" for="textField1" data-lib="WAF" data-typ
style="width: auto;">
```

Here are the properties for this **Label** widget:

Property	Description
id	Label ID
class	CSS class for Label widget
data-constraint-top	Top constraint defined in the Styles tab
data-constraint-left	Left constraint defined in the Styles tab
data-valign	vAlign defined in the Styles tab
for	Label widget defined for the widget
data-lib	Data library
data-type	Widget type (label)
style	Other styles defined in the Styles tab

renderId

Description

This property is the same as **id**.

source

Description

source provides you with all the information regarding the datasource bound to the widget. All the properties and methods for the datasource are contained in this object as defined in the **Server Datasources**.

If the datasource defined for the widget is a datastore class, all the public attributes are added as properties. An additional property, **length**, defines the total number of entites in the datastore class.

If the datasource is of type Array, there will be a property for each of the attributes defined for the array and the **length** property, which defines number of elements in the array.

If the datasource is of type Variable, the ID of the JavaScript Variable will be indicated as another property.

If no datasource is defined for the widget, this property returns *null*.

If we have a Text Input widget whose Source is "employee.salary", we can obtain the values in the other attributes in the Employee datastore class through the **source** property:

```
var myFirstName, myID, myTotalEntities;
myFirstName = $$('textField1').source.firstName; // returns "Pete"
myID = $$('textField1').source.ID; // returns 3 (entity's ID)
myTotalEntities = $$('textField1').source.length; // returns 544 (total number of entitie
```

sourceAtt

Description

sourceAtt provides you with all the information regarding the datasource bound to the widget. All the properties and methods for the datasource are contained in this object.

All the methods are defined in the **Server Datasources (Attribute)**.

This object has a few properties that you can access:

Property	Description
name	Name of the attribute
kind	Attribute kind (see **Attribute Categories**)
type	Attribute type (see **Storage Attribute Types** for storage attribute types)

If no datasource is defined for the widget, this property returns *null*.

To retrieve the value defined in the widget's datasource, you can write **sourceAtt.getValue()**.

Example

If we have a Text widget whose source is employee.lastName, the following data is returned for the properties in the **sourceAtt** object:

```
var attName,attKind,attType,attValue;
attName=$$('richText2').sourceAtt.name;    // returns lastName
attKind=$$('richText2').sourceAtt.kind;     // returns storage
attType=$$('richText2').sourceAtt.type;     // returns string
attValue=$$('richText2').sourceAtt.getValue();     // returns "Smith"
```

sources

Description

The **sources** property returns an object with an object for each datasource used in the Component widget. If, for example, you are using two datasources "employee" and "company," this property contains an object with two objects of type server datasource: "employee" and "company".

For more information about server datasources, refer to the **Server Datasources** chapter.

addChild( )

void **addChild**(Widget *widget*)

Parameter	Type	Description
widget	Widget	Widget to add as a child

Description

addChild( ) allows you to add a widget as a "child," which means that it will be contained in the "parent" widget.

This function is useful when you want to include a widget (already available on your Interface page) inside of a **Container** widget. Because each tab in a **Tab View** widget is made up of a Container widget and a Menu Item, you can also add a widget to a tab's Container widget.

To position each widget you include into the Container widget, you must use the **move( )** function.

If you want to remove the "child" from the "parent" widget, you can use the **destroy( )** function.

Example

The following example adds a Button widget to the Container widget:

```
$$('container1').addChild($$('button1')); //add this widget (already on your page) to the
$$('button1').move(10,10);
```

addChildren( )

void **addChildren**(Widget *widget* [,..., Widget *widget*N])

Parameter	Type	Description
widget	Widget	Widgets to add as children

Description

addChildren( ) allows you to add widgets as "children," which means that they will be contained in the "parent" widget.

This function is useful when you want to include widgets (already available on your Interface page) inside of a **Container** widget. Because each tab in a **Tab View** widget is made up of a Container widget and a Menu Item, you can

also add one or more widgets to the tab's Container widget.

To position each widget you include into the Container widget, you must use the **move( )** function.

Example

The following example allows you to add a Button widget and a Component widget into a new tab in the Tab View widget:

```
$$('tabView1').addTab("New Tab"); //add a new tab to our Tab View
var newlyAddedTab=$$('tabView1').countTabs(); //newly created tab is the last one; theref
var myContainer=$$('tabView1').getContainer(newlyAddedTab); //get the newly created tab's
$$(myContainer.id).addChildren($$('button2'),$$('component2')); //add the following widge

$$('button2').show(); //the button was hidden previously offscreen
$$('button2').move(20,20); //position the button inside the Container

$$('component2').loadComponent();//load component (by default it was not loaded)
$$('component2').move(20,50); //position the component inside the Container
```

addClass( )

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

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

Description

With this method, you can add a CSS class to your widget by passing it in *cssClass*. If you want to remove it, call the **removeClass( )** method.

Example

The following example adds the "myCustomClass" CSS class to a widget:

```
$$('firstName').addClass('myCustomClass');
```

addListener( )

void **addListener**(String *event* , String *callback* [, Object *options*])

Parameter	Type	Description
event	String	Event to add the listener to (e.g., "click")
callback	String	Callback associated to the event
options	Object	Options to add to the callback

Description

Use this method to add a listener to the widget for a specific *event*. You can add as many listeners to a widget for an *event*. To remove a listener, call **removeListener( )**.

event is a jQuery event, like "click", "dblcick", and "focus". Refer to the jQuery API, to see which form, mouse, and keyboard events are supported.

Example

In the following example, we add a listener to a button. Each time it is clicked, an alert is displayed:

```
$$('button1').addListener('click',function() { alert("Button was clicked."); });
```

In this example, you can add a listener to the same button with *options* defined:

```
$$('button1').addListener('click',function(event) { alert(event.data.title + event.data.m
{msg: "Button was clicked", title: "MESSAGE: "});
```

clear( )

void **clear**( )

Description

This method clears the value displayed in the widget.

When called for the **Text Input**, **Slider**, **Checkbox**, **Image**, and **Text** widgets, the value displayed is cleared.

When used with the **Auto Form** widget, all the values in the **Text Input**, **Slider**, **Checkbox**, **Image**, and **Text** widgets are cleared.

Example

The example below clears the value defined in a **Slider**:

```
$$('slider0').clear()
```

clearErrorMessage( )

void **clearErrorMessage**( )

Description

Clears the error message for the widget displayed in the associated **Display Error** widget.

You define the **Display Error** widget in the Properties tab for widgets like **Text Input**, **Checkbox**, **Slider**, and **Grid**.

Example

If you want to clear the error message that appears for a Text Input widget, you call **clearErrorMessage( )** as shown below:

```
$$('textField3').clearErrorMessage()
```

If there is any text displayed in the **Display Error** widget whose ID was specified in the Text Input widget's **Display Error** property, it will be cleared.

destroy( )

void **destroy**( )

Description

This method deletes the widget from the Interface page and removes all the listeners associated to it.

disable( )

void **disable**([Number *tabNumber*])

Parameter	Type	Description
tabNumber	Number	For the Tab View widget, you can specify the tab number

Description

disable( ) allows you to disable data entry for a widget or all the widgets in a Container or Tab View. In the *tabNumber* parameter, you can define on which tab you want to disable all the widgets. Otherwise, all the widgets on all the tabs will be disabled.

You can use this function on the following widgets:

Widget	Description
Text Input	Disable the widget
Slider	Disable the Slider and its handle(s)

Grid	Cells become read-only and if the footer is displayed, the + and the - buttons are removed
Button	Disable the Button

Note: Other widgets will be added to this list.

The **Button** widget will be disabled, but visually it will appear unchanged. You can add your styles to the **.waf-widget.waf-state-disabled** class in your Interface page's custom CSS file to modify how the Button widget appears when disabled. In our example below, we change the background and the text color:

```
.waf-widget.waf-state-disabled {
    background: #666;
    color:  #333;
}
```

draggable( )

void **draggable**(Boolean *boolean*)

Parameter	Type	Description
boolean	Boolean	True = activate the draggable option for a widget; False = disactivate it

Description

This method allows you to activate or disactivate the draggable option for the widget.

Pass *True* to make the widget draggable and *False* to not allow it to be draggable.

Example

To make a widget draggable:

```
$$('dataGrid1').draggable(true);
```

To make it no longer draggable:

```
$$('dataGrid1').draggable(false);
```

enable( )

void **enable** (tabNumber)

Parameter	Type	Description
tabNumber	Number	For the Tab View widget, you can specify the tab number

Description

enable( ) allows you to enable data entry for a widget or all the widgets in a Container or Tab View. In the *tabNumber* parameter, you can define on which tab you want to enable all the widgets. Otherwise, all the widgets on all the tabs will be enabled.

You can use this function on the following widgets:

Widget	Description
Text Input	Enable the widget
Slider	Enable the Slider and its handle(s)
Grid	Cells become enabled and if the footer is displayed, the + and the - buttons are displayed
Button	Enable the Button

Note: Other widgets will be added to this list.

focus( )

void **focus**( )

Description

This method allows you to put the focus on a widget on the Interface page. For the Text Input widget, the cursor is also placed in it as well.

To check if the Text Input widget already has the focus, call **hasFocus( )**.

In this example, we put the focus on a Text Input widget and add a message to the Display Error widget (defined for the "firstName" Text Input widget):

```
$$('firstName').focus();
$$('firstName').setErrorMessage("The First Name field is mandatory.");
```

getChildren( )

Array **getChildren**( )

Returns Array Returns an array of objects where each object defines a widget included in the main widget

Description

This method returns an array of objects where each object defines a widget included in the parent widget. The parent widget contains "children" widgets.

getChildren( ) is useful when you want to know which widgets are inside of a widget, like a **Container**.

Example

If you call this method on a Container widget that contains other widgets:

```
$$('container1').getChildren();
```

It will return an array of objects:

```
[Object { id="combobox1", kind="combobox", divID="combobox1", ...},
Object { id="label2", kind="label", divID="label2", ...},
Object { id="textField1", kind="textField", divID="textField1", ...},
Object { id="label3", kind="label", divID="label3", ...},
Object { id="textField2", kind="textField", divID="textField2", ...},
Object { id="label4", kind="label", divID="label4", ...},
Object { id="button1", kind="button", divID="button1", ...}]
```

getErrorDiv( )

String **getErrorDiv**( )

Returns String jQuery reference to the Display Error widget defined for the widget

Description

This method returns the jQuery reference to the Display Error widget ID defined in the properties for the widget. You can also dynamically set the **Display Error** widget ID for a widget by calling **setErrorDiv( )**.

This example allows you to retrieve the Display Error widget ID for the Text Input widget:

```
var myErrorDiv;
myErrorDiv=$$('textField3').getErrorDiv();
 //myErrorDiv returns the following: [div#myErrorDivWidget.waf-widget]
 //where myErrorDivWidget is the Display Error widget ID defined for the Text Input widge
```

getHeight( )

Number **getHeight**( )

Returns	Number	Height of the widget

Description

getHeight( ) returns the height of the widget.

getLabel( )

Object **getLabel**( )

Returns	Object	Label widget for a widget

Description

The **getLabel( )** function returns the Label widget as an object.

Once you retrieve the Label widget object, you can then apply other Widget class APIs to it, like **setTextColor( )** and **setValue( )**.

Example

In our example below, we change the color of the Text input widget's label:

```
var labelWidget=$$('firstName').getLabel();
labelWidget.setTextColor("red");
```

You can also write the following:

```
$$('firstName').getLabel().setTextColor("red");
```

getLinks( )

Array **getLinks**( )

Returns	Array	Returns an array of objects where each object defines a widget linked to the main widget

Description

This method returns an array of objects where each object defines a widget linked to the main widget. A widget can contain "linked" widgets.

getLinks( ) is useful when you want to know which widgets are linked to a widget, like for a **Tab View** that is made up of multiple "linked" widgets (i.e., a **Menu Bar** and one or more **Container** widgets).

Example

If you call this method on a Tab View widget:

```
$$('tabView1').getLinks();
```

It will return an array of objects:

```
[Object { id="menuBar1", kind="menuBar", divID="menuBar1", ...},
Object { id="container3", kind="container", divID="container3", ...},
Object { id="container4", kind="container", divID="container4", ...}]
```

getParent( )

Widget **getParent**( )

Returns	Widget	Parent widget

Description

getParent( ) returns the "parent" widget for the widget to which it is applied. The parent can be another widget, like **Container** or **Tab View**, although generally you will use a Container as the widget's parent.

getPosition( )

Object **getPosition**

Returns Object Object with four properties (bottom, left, top, and right) defining the widget's position

Description

getPosition( ) returns the widget's position in an object containing four properties: **bottom**, **left**, **top**, and **right**.

Example

The following example retrieves the position of a Grid:

```
gridPositions = $$('dataGrid1').getPosition();
    //top position is gridPositions.top
    //left position is gridPositions.left
    //bottom position is gridPositions.bottom
    //right position is gridPositions.right
$$('dataGrid1').move(gridPositions.left, gridPositions.top + 20); //move the grid 20 pixe
    //or
$$('dataGrid1').setTop(gridPositions.top + 20); //move the grid 20 pixels to the left
```

getState( )

String **getState**( )

Returns String State of the widget (e.g., default, active, focus, hover, or disabled)

Description

getState( ) allows you to get the state of the widget.

getTheme( )

String **getTheme**( )

Returns String Theme(s) defined for the widget

Description

This method returns the theme(s) defined for the widget. **getTheme( )** returns either the selected theme for the widget defined on the **Widget Skins** tab. If "Inherited" is selected (which is the value by default), the Interface Page's theme will be returned after "Inherited."

Example

In our example, we retrieve the theme for a widget:

```
var myTheme;
myTheme = $$('firstName').getTheme();
```

If a Theme was selected for Text Input widget (even if another Theme was selected for the Interface Page), it will return the selected theme for the Text Input widget. Otherwise, **getTheme( )** returns the widget's theme and then the Interface page's theme, i.e., "inherited metal".

getValue( )

String **getValue**( )

Returns	String	Value displayed in the widget

Description

This method returns the value displayed in the widget.

Below is a tab that defines the values returned for the different widgets:

Widget	**Value**
Button	Title for the button
Checkbox	If checked, returns true, otherwise it returns false
Combo Box	Selected value
Image	Either the URL to get to the image if defined by the **Src** property or the REST request to view it if the image is in the datastore
Radio Button Group	Selected value
Slider	Currently displayed value (based on the **Step** property in the **Properties** tab)
Switch	If set to "on", returns true, otherwise it returns false
Text	Currently displayed value
Text Input	Currently displayed value

You can format values for widgets of type **Text** and **Text Input** in its **Properties** tab in the **GUI Designer** or the actual attribute bound to a widget in the **Datastore Model Designer**.

To retrieve the actual value from the datasource, you must use the sourceAtt property as shown below:

```
$$('myWidget').sourceAtt.getValue();
```

Example

To retrieve the formatted value displayed in a widget:

```
$$('salary').getValue();  //returns $44,000.00
```

To get the actual value in the widget (without formatting), you can write the following:

```
$$('salary').sourceAtt.getValue();  //returns 44000
```

getWidth( )

Number **getWidth**( )

Returns	Number	Width of the widget in pixels

Description

getWidth( ) returns the width of the widget.

hasFocus( )

Boolean **hasFocus**( )

Returns	Boolean	True = widget has the focus; False = widget does not have the focus

Description

The **hasFocus( )** method returns *True* if the widget has the focus and *False* if it does not. To put the focus on a particular widget, call **focus( )**.

hide( )

void **hide**([String *mode*])

Parameter	Type	Description
mode	String	Pass "visibility" to hide the widget (visibility:"hidden"). Otherwise, it will be removed from the Interface page (display:"none").

Description

This method hides the widget on the Interface page.

If you have layered widgets, pass "visibility" to this method to make sure that the other widgets are resized correctly. If you pass "visibility" to hide the widget, the CSS generated is visibility:"hidden". Otherwise, the CSS generated is display:"none".

You can show a hidden widget by using the **show( )** method or toggle (if it's visible, hide it and if it's hidden, show it) it by calling **toggle( )**.

isDisabled( )

Boolean **isDisabled**( )

Returns	Boolean	True = widget is disabled; False = widget is not disabled

Description

The **isDisabled( )** method returns *True* if the widget is disabled and *False* if it is not. To disable a particular widget, call **disable( )**.

isVisible( )

Boolean **isVisible**( )

Returns	Boolean	True = widget is visible; False = widget is not visible

Description

The **isVisible( )** method returns *True* if the widget is visible and *False* if it is not. To make a particular widget visible, call **show( )** or **toggle( )** if it is hidden.

link( )

void **link**(Widget *widget*)

Parameter	Type	Description
widget	Widget	Widget to link

Description

link( ) allows you to link a widget to another widget.

This function is useful when you create your own widget and want to link multiple widgets together. We have linked multiple widgets (a **Menu Bar** and one or more **Container** widgets) for a **Tab View** widget.

To find out which widgets are linked to a specific widget, you can use the **getLinks( )** function.

Example

The following example links a Button widget with a Text Input widget:

```
$$('saveDialogButton').link($$('nameField'));
```

move( )

void **move**(String | Number *left*, String | Number *top*)

Parameter	Type	Description
left	String, Number	Left coordinate of the new position for the widget
top	String, Number	Top coordinate of the new position for the widget

Description

This method allows you to move the widget to its new *left* and *top* coordinates. The *left* and *top* coordinates can be expressed as numbers or strings.

Example

The following example shows two ways to move a widget on your Interface page:

```
$$('title').move(10,10);

$$('title').move("10px","10px");
```

redraw( )

void **redraw**( )

Description

This method allows you to redraw the widget on the Interface page.

removeClass( )

void **removeClass**([String *cssClass*])

Parameter	Type	Description
cssClass	String	CSS class to remove from the widget

Description

This method removes the *cssClass* from the widget. If you do not pass a *cssClass*, all the classes defined for the widget will be removed. If you want to add a CSS class to a widget, call the **addClass( )** method.

Example

Remove the "myCustomClass" from the widget's CSS class property:

```
$$('firstName').removeClass('myCustomClass');
```

removeListener( )

void **removeListener**(String *event* [, String *callback*])

Parameter	Type	Description
event	String	Event for which to remove the listener
callback	String	Callback for the event

Description

With this method, you can remove a particular listener or all listeners previously added by the **addListener( )** method. To remove a particular listener, you must pass not only the *event*, but also the *callback*. To remove all listeners for an event, pass only the *event*.

Example

The following example removes all listeners added to the widget for the "click" *event*:

```
$$('button1').removeListener('click');
```

removeState( )

void **removeState**(String *state*)

Parameter	Type	Description
state	String	State of the widget (e.g., default, active, focus, hover, or disabled)

Description

With **removeState( )**, you can remove the state of the widget.

resizable( )

void **resizable**(Boolean *boolean*)

Parameter	Type	Description
boolean	Boolean	True = activate the resizable option for the widget; False = disactivate it

Description

This method activates or disactivates the ability to resize the widget. **resizable( )** can only be applied to widgets that have the **Resizable** option available in the Properties tab.

Pass *True* to activate the ability to resize the widget and *False* to disactivate it.

Example

To make the Grid resizable:

```
$$('dataGrid1').resizable(true);
```

resize( )

void **resize**(String | Number *height*, String | Number *width*)

Parameter	Type	Description
height	String, Number	Height for the widget
width	String, Number	Width for the widget

Description

This method resizes the widget to its new *height* and *width*. The *height* and *width* can be expressed as a number or string.

Example

The following example shows two ways to resize a widget on your Interface page:

```
$$('lastName').resize(20,200);

$$('lastName').resize("20px","200px");
```

setBackgroundColor( )

void **setBackgroundColor**(String *backgroundColor*)

Parameter	Type	Description
backgroundColor	String	Background color

Description

This method sets the widget's background color to *backgroundColor*. *backgroundColor* can be expressed as:

- an HTML color value, i.e., "blue",
- a HEX value, i.e., "#CCC" or "#39C488", or
- a RGB value, i.e., #ff00ff.

Example

Set the background color for a container:

```
$$('formContainer').setBackgroundColor("#650092");
```

setBottom( )

void **setBottom**(Number *bottom*)

Parameter	Type	Description
bottom	Number	Number of pixels for the widget's bottom position

Description

setBottom( ) sets the bottom position of the widget. Passing no value to this function clears the value for the widget's bottom position.

For more information about the positioning of a widget, refer to the **Size & Position** section in the **GUI Designer** manual.

setErrorDiv( )

void **setErrorDiv**(String *errorDiv*)

Parameter	Type	Description
errorDiv	String	Display Error widget ID for the widget

Description

With this method, you can set the Display Error for the widget if it has the Display Error property in its Properties tab. To get the existing **Display Error** widget ID, use the method.

setErrorMessage( )

void **setErrorMessage**(Object *messageObject*)

Parameter	Type	Description
messageObject	Object	message: error message to display tooltip: true=display tooltip

Description

Sets the error message for the widget that will be displayed in the associated **Display Error** widget. If the widget (e.g., Text Input, Slider, or Checkbox widgets) has the **Display Error** property and nothing is entered, this function displays the error message in an alert.

In this example, we set the error message for the **Display Error** widget that is associated to a **Text Input** widget:

```
$$('email').setErrorMessage( { message: "Please enter a valid email address.", tooltip: t
```

The error message will be displayed in the Display Error widget whose ID was defined for our Text Input and when you hover over the widget, the tooltip will be displayed:

setHeight( )

void **setHeight**(Number *height*)

Parameter	Type	Description
height	Number	Height of the widget

Description

The setHeight( ) function allows you to set the height of the widget by passing a new value to the *height* parameter.

setLeft( )

void **setLeft**(Number *left*)

Parameter	Type	Description
left	Number	Number of pixels for the widget's left position

Description

setLeft( ) sets the left position of the widget. Passing no value to this function clears the value for the widget's left position.

For more information about the positioning of a widget, refer to the **Size & Position** section in the **GUI Designer** manual.

setParent( )

void **setParent** (widget)

Parameter	Type	Description
widget	Widget	Widget to define as parent

Description

With **setParent( )**, you can set a **Container** widget as the parent of a widget.

Once you have set the Container widget as the parent, you must position the widget inside of the Container widget by using the **move( )** function.

Example

The following example allows you to add a Button widget and a Component widget into a new tab in the Tab View widget:

```
$$('tabView1').addTab("New Tab"); //add a new tab to our Tab View
var newlyAddedTab=$$('tabView1').countTabs(); //newly created tab is the last one; theref
var myContainer=$$('tabView1').getContainer(newlyAddedTab); //get the newly created tab's

$$('button2').show();
$$('button2').setParent($$(myContainer.id)); //the button was hidden previously offscreen
$$('button2').move(20,20); //move the button inside the Container

$$('component2').setParent($$(myContainer.id)); //the button was hidden previously offscr
$$('component2').loadComponent();//load component (by default it was not loaded)
$$('component2').move(20,50); //move the component inside the Container
```

setRight( )

void **setRight**(Number *right*)

Parameter	Type	Description
right	Number	Number of pixels for the widget's right position

Description

setRight( ) sets the right position of the widget. Passing no value to this function clears the value for the widget's right position.

For more information about the positioning of a widget, refer to the **Size & Position** section in the **GUI Designer** manual.

setState( )

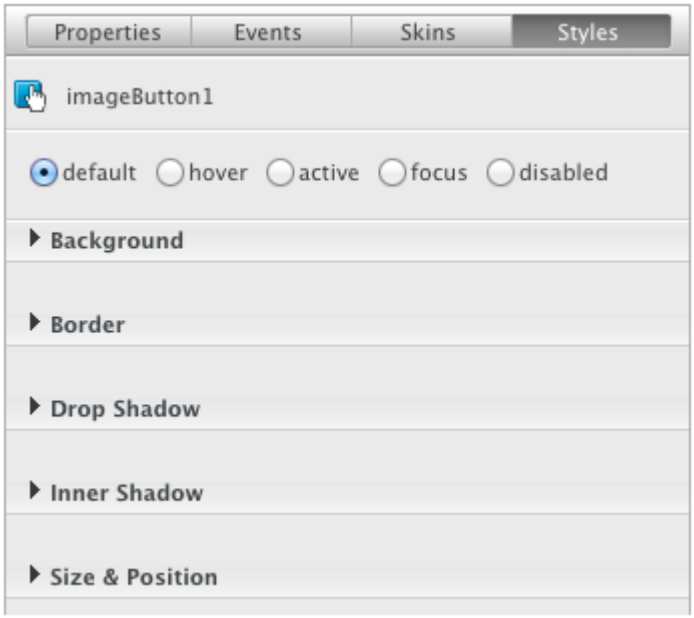
void **setState**(String *state*)

Parameter	Type	Description
state	String	State of the widget (e.g., default, active, focus, hover, or disabled)

Description

setState( ) allows you to set the widget's state. Many widgets have states, like default, hover, active, focus, selected, and disabled. This function allows you to set it to one of these states even if it's not available in the widget's Styles tab.

The widget's states are displayed in the Styles tab:



Example

The following example sets the Image Button widget's state to disabled, thus displaying the "disabled" state of the widget:

```
$$('imageButton1').setState('disabled');
```

setTabIndex( )

void **setTabIndex**(Number *tabIndex*)

Parameter	Type	Description
tabIndex	Number	Tab index for the widget

Description

With this method, you can set the widget's tab index defined in the **Tabindex** property on the widget's Properties tab.

setTextColor( )

void **setTextColor**(String *textColor*)

Parameter	Type	Description
textColor	String	Text color

Description

This method sets the widget's text color to *textColor*. *textColor* can be expressed as:

- an HTML color value, i.e., "blue",
- a HEX value, i.e., "#CCC" or "#39C488", or
- a RGB value, i.e., #ff00ff.

Example

Set the color of the text typed in a Text Input widget:

```
$$('firstName').setTextColor("#650092");
```

setTop( )

void **setTop**(Number *top*)

Parameter	Type	Description
top	Number	Number of pixels for the widget's top position

Description

setTop( ) sets the top position of the widget. Passing no value to this function clears the value for the widget's top position.

For more information about the positioning of a widget, refer to the **Size & Position** section in the **GUI Designer** manual.

setValue( )

void **setValue**(String *value*)

Parameter	Type	Description
value	String	Value to set in the widget

Description

With this method, you can set the value for a widget. The value entered is not saved until you save the entity in the datasource.

Widget	Value
Button	Title for the button
Checkbox	Pass true of false
Combo Box	A value defined for the widget
Image	A path to the image file (URL or path relative to the project)
Radio Button Group	A value defined for the widget
Slider	Any value in the range of the widget
Switch	Pass true to set to "on" or false to set to "off"
Text	Any value
Text Input	Any value

Example

The following example assigns an image in our project's images folder to an Image widget:

```
$$('image1').setValue("images/myImage.jpg")
```

setWidth( )

void **setWidth**(Number *width*)

Parameter	Type	Description
width	Number	Width of the widget

Description

Use the **setWidth( )** function to set the width of the widget by passing a new value to the *width* parameter.

show( )

void **show**( )

Description

With this method, you can show a hidden widget on the Interface page. You can hide a widget by using the **hide( )**

method or toggle (if it's visible, hide it and if it's hidden, show it) it by calling **toggle( )**.

toggle( )

void **toggle**(String *mode*)

Parameter	Type	Description
mode	String	Pass "visibility" to hide the widget (visibility:"hidden"). Otherwise, it will be removed from the Interface page (display:"none").

Description

This method allows you to toggle the display of the widget. If it is visible, it is hidden and if it is hidden, it is made visible. You can also use the **hide( )** method to hide the widget and the **show( )** method to show it.

unlink( )

void **unlink**(Widget *widget*)

Parameter	Type	Description
widget	Widget	Widget to unlink

Description

unlink( ) allows you to unlink one widget from another widget.

This function is useful when you create your own widget and want to link multiple widgets together. We have linked multiple widgets (a **Menu Bar** and one or more **Container** widgets) for a **Tab View** widget.

To find out which widgets are linked to a specific widget, you can use the **getLinks( )** function.

Example

The following example unlinks a Button widget and a Text Input widget (if they were previously linked):

```
$$('saveDialogButton').unlink($$('nameField'));
```

WYSIWYG Editor

The **WYSIWYG Editor** widget inherits from the Widget class; however, not all the properties and functions are available to it. This widget has two functions:

- **getWysiwygInstance( )**: Return the WYSIWYG Editor's instance to then access its attributes, functions, and methods.
- **getControlManager( )**: Return the WYSIWYG Editor's instance Control Manager so that you can access the Control Manager's methods.

To retrieve the HTML text in the WYSIWYG Editor, use the **getValue( )** method.

To set the HTML text in the WYSIWYG Editor, use the **setValue( )** method.

getControlManager( )

Object **getControlManager**

Returns	Object	Control Manager for the WYSIWYG Editor

Description

getControlManager( ) returns the WYSIWYG Editor's Control Manager so that you can access its methods. The Control Manager is useful if you want to enable and/or disable the WYSIWYG Editor's buttons.

To find out more about the editor's attributes, events, and methods, view Tiny MCE's Control Manager.

Example

The following example allows you to disable the Save button in the WYSIWYG Editor's toolbar:

```
$$('wysiwyg1').getControlManager().setDisabled('save',true);
```

getWysiwygInstance( )

Object **getWysiwygInstance**

Returns	Object	WYSIWYG Instance

Description

getWysiwygInstance( ) returns an instance of the WYSIWYG Editor so that allows you to access the WYSIWYG Editor's attributes, events, and methods. To find out more about the editor's attributes, events, and methods, view Tiny MCE's API.

Example

The following example hides the WYSIWYG Editor and displays the HTML text in an editable text area:

```
$$('wysiwyg1').getWysiwygInstance().hide();
```

The following is displayed after the call above:

Résumé

```
<h3 align="center"><strong>Pete Miller</strong><strong></strong></h3>
<p align="center"><em>123 Main Street</em><em></em></p>
<p align="center"><em>San Jose, CA</em><em></em></p>
<p style="text-align: center;"><em>(408) 555-1212</em></p>
<p align="center"><em><br /></em></p>
<h3>Job Objective</h3>
<p>I am an experienced JavaScript Programmer seeking employment in a company that can fully utilize my skills and
expertise.</p>
<p> </p>
<h3>Work History</h3>
<p><em><strong>Acme Company</strong></em><em><strong></strong></em></p>
<p><em>January 2011 - present</em><em></em></p>
<p>I am currently working as a JavaScript Programmer where I develop interesting web applications.<strong></strong></p>
<p><strong><br /></strong></p>
<p><em><strong>Another Company</strong></em><em><strong></strong></em></p>
<p><em>March 2009 - January 2011<br /></em></p>
<p>I worked as a junior programmer on many different projects.<strong><strong><br /> </strong></strong></p>
<p><strong><br /></strong></p>
<p><em><strong>First Company</strong></em><em><strong></strong></em></p>
<p><em>October 2007 - <em>March 2009</em><br /></em></p>
```

To return to the WYSIWYG Editor, you can write the following:

```
$$('wysiwyg1').getWysiwygInstance().show();
```

Résumé