

JSButton

Extends

[JSComponent](#)

Property Summary

Number	anchors Enables a component to stick to a specific side of form and/or to grow or shrink when a window is resized.
String	background The background color of the component.
String	borderType The type, color and style of border of the component.
String	dataProviderID The dataprovider of the component.
Boolean	displaysTags Flag that enables or disables merging of data inside components using tags (placeholders).
Boolean	enabled The enable state of the component, default true.
String	fontType The font type of the component.
String	foreground The foreground color of the component.
Number	formIndex The Z index of this component.
String	format The format that should be applied when displaying data(using dataProviderID) in the label/button.
String	groupID A String representing a group ID for this component.
Number	height The height in pixels of the component.
Number	horizontalAlignment Horizontal alignment of the text inside the component.
JSMedia	imageMedia The image Media object that should be displayed inside the component.
String	labelFor Some components can be set to be labels of other components.
String	margin The margins of the component.
Number	mediaOptions Options to scale the image Media object that is displayed inside the component.
String	mnemonic The keyboard shortcut that activates this component.
String	name The name of the component.
JSMethod	onAction The method that is executed when the component is clicked.
JSMethod	onDoubleClick The method that is executed when the component is double clicked.
JSMethod	onRender The method that is executed when the component is rendered.
JSMethod	onRightClick The method that is executed when the component is right clicked.
Number	printSliding Enables an element to resize based on its content and/or move when printing.
Boolean	printable Flag that tells if the component should be printed or not when the form is printed.
Number	rolloverCursor The cursor that is shown as the mouse is rolled over the component.
JSMedia	rolloverImageMedia The roll over image Media object used.
Number	rotation The rotation of the element.
Boolean	showClick When set, the element will show the clicked state when selected.

Boolean	showFocus When set the text of an element will showfocus when selected.
String	styleClass The name of the style class that should be applied to this component.
Number	tabSeq An index that specifies the position of the component in the tab sequence.
String	text The text that is displayed inside the component.
String	toolTipText The text displayed when hovering over the component with a mouse cursor.
Boolean	transparent Flag that tells if the component is transparent or not.
Number	verticalAlignment The vertical alignment of the text inside the component.
Boolean	visible The visible property of the component, default true.
Number	width The width in pixels of the component.
Number	x The x coordinate of the component on the form.
Number	y The y coordinate of the component on the form.

Method Summary

Object	getDesignTimeProperty() Get a design-time property of an element.
String[]	getDesignTimePropertyNames() Get the design-time properties of an element.
UUID	getUUID() Returns the UUID of this component.
Object	putDesignTimeProperty() Set a design-time property of an element.
Object	removeDesignTimeProperty() Clear a design-time property of an element.

Property Details

anchors

Enables a component to stick to a specific side of form and/or to grow or shrink when a window is resized.

If opposite anchors are activated then the component with grow or shrink with the window. For example if Top and Bottom are activated, then the component will grow/shrink when the window is vertically resized. If Left and Right are activated then the component will grow/shrink when the window is horizontally resized.

If opposite anchors are not activated, then the component will keep a constant distance from the sides of the window which correspond to the activated anchors.

Returns

[Number](#)

Sample

```

var form = solutionModel.newForm('mediaForm', 'db:/example_data/parent_table', null, false, 400, 300);
var stretchAllDirectionsLabel = form.newLabel('Stretch all directions', 10, 10, 380, 280);
stretchAllDirectionsLabel.background = 'red';
stretchAllDirectionsLabel.anchors = SM_ANCHOR.ALL;
var stretchVerticallyLabel = form.newLabel('Stretch vertically', 10, 10, 190, 280);
stretchVerticallyLabel.background = 'green';
stretchVerticallyLabel.anchors = SM_ANCHOR.WEST | SM_ANCHOR.NORTH | SM_ANCHOR.SOUTH;
var stretchHorizontallyLabel = form.newLabel('Stretch horizontally', 10, 10, 380, 140);
stretchHorizontallyLabel.background = 'blue';
stretchHorizontallyLabel.anchors = SM_ANCHOR.NORTH | SM_ANCHOR.WEST | SM_ANCHOR.EAST;
var stickToTopLeftCornerLabel = form.newLabel('Stick to top-left corner', 10, 10, 200, 100);
stickToTopLeftCornerLabel.background = 'orange';
stickToTopLeftCornerLabel.anchors = SM_ANCHOR.NORTH | SM_ANCHOR.WEST; // This is equivalent to SM_ANCHOR.DEFAULT
var stickToBottomRightCornerLabel = form.newLabel('Stick to bottom-right corner', 190, 190, 200, 100);
stickToBottomRightCornerLabel.background = 'pink';
stickToBottomRightCornerLabel.anchors = SM_ANCHOR.SOUTH | SM_ANCHOR.EAST;

```

background

The background color of the component.

Returns

[String](#)

Sample

```

// This property can be used on all types of components.
// Here it is illustrated only for labels and fields.
var greenLabel = form.newLabel('Green',10,10,100,50);
greenLabel.background = 'green'; // Use generic names for colors.
var redField = form.newField('parent_table_text',JSField.TEXT_FIELD,10,110,100,30);
redField.background = '#FF0000'; // Use RGB codes for colors.

```

borderType

The type, color and style of border of the component.

Returns

[String](#)

Sample

```

//HINT: To know exactly the notation of this property set it in the designer and then read it once out
through the solution model.
var field = form.newField('my_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
field.borderType = solutionModel.createLineBorder(1,'#ff0000');

```

dataProviderID

The dataprovider of the component.

Returns

[String](#)

Sample

```

// Normally the dataprovider is specified when a component is created.
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 40, 100, 20);
// But it can be modified later if needed.
field.dataProviderID = 'parent_table_id';

```

displaysTags

Flag that enables or disables merging of data inside components using tags (placeholders). Tags (or placeholders) are words surrounded by %% on each side. There are data tags and standard tags. Data tags consist in names of data providers surrounded by %%. Standard tags are a set of predefined tags that are made available by the system.

See the "Merging data" section for more details about tags.

The default value of this flag is "false", that is merging of data is disabled by default.

Returns

[Boolean](#)

Sample

```
var label = form.newLabel('You are viewing record no. %%parent_table_id%%. You are running on server %%
serverURL%%.',
                                10, 10, 600, 100);

label.displaysTags = true;
```

enabled

The enable state of the component, default true.

Returns

[Boolean](#)

Sample

```
var form = solutionModel.newForm('printForm', 'db:/example_data/parent_table', null, false, 400, 300);
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
field.enabled = false;
```

fontType

The font type of the component.

Returns

[String](#)

Sample

```
var label = form.newLabel('Text here', 10, 50, 100, 20);
label.fontType = solutionModel.createFont('Times New Roman',1,14);
```

foreground

The foreground color of the component.

Returns

[String](#)

Sample

```
// This property can be used on all types of components.
// Here it is illustrated only for labels and fields.
var labelWithBlueText = form.newLabel('Blue text', 10, 10, 100, 30);
labelWithBlueText.foreground = 'blue'; // Use generic names for colors.
var fieldWithYellowText = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 50, 100, 20);
fieldWithYellowText.foreground = '#FFFF00'; // Use RGB codes for colors.
```

formIndex

The Z index of this component. If two components overlap, then the component with higher Z index is displayed above the component with lower Z index.

Returns

[Number](#)

Sample

```
var labelBelow = form.newLabel('Green', 10, 10, 100, 50);
labelBelow.background = 'green';
labelBelow.formIndex = 10;
var fieldAbove = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 30);
fieldAbove.background = '#FF0000';
fieldAbove.formIndex = 20;
```

format

The format that should be applied when displaying data(using dataProviderID) in the label/button. Some examples are "%", "dd-MM-yyyy", "MM-dd-yyyy", etc.

Returns

[String](#)

Sample

```
var label = form.newLabel('', 10, 10, 100, 100);
label.format = '$#.00';
```

groupID

A String representing a group ID for this component. If several components have the same group ID then they belong to the same group of components. Using the group itself, all components can be disabled/enabled or made invisible/visible.

The group id should be a javascript compatible identifier to allow access of the group in scripting.

Returns

[String](#)

Sample

```
var form = solutionModel.newForm('someForm', 'db:/example_data/parent_table', null, false, 400, 300);
var label = form.newLabel('Green', 10, 10, 100, 20);
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 40, 100, 20);
label.groupID = 'someGroup';
field.groupID = 'someGroup';
forms['someForm'].elements.someGroup.enabled = false;
```

height

The height in pixels of the component.

Returns

[Number](#)

Sample

```
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
application.output('original width: ' + field.width);
application.output('original height: ' + field.height);
field.width = 200;
field.height = 100;
application.output('modified width: ' + field.width);
application.output('modified height: ' + field.height);
```

horizontalAlignment

Horizontal alignment of the text inside the component. Can be one of LEFT, CENTER or RIGHT.

Note that this property does not refer to the horizontal alignment of the component inside the form.

Returns

[Number](#)

Sample

```
var leftAlignedLabel = form.newLabel('LEFT', 10, 10, 300, 20);
leftAlignedLabel.horizontalAlignment = SM_ALIGNMENT.LEFT;
var hCenteredLabel = form.newLabel('CENTER', 10, 40, 300, 20);
hCenteredLabel.horizontalAlignment = SM_ALIGNMENT.CENTER;
var rightAlignedLabel = form.newLabel('RIGHT', 10, 70, 300, 20);
rightAlignedLabel.horizontalAlignment = SM_ALIGNMENT.RIGHT;
```

imageMedia

The image Media object that should be displayed inside the component.

Returns

[JSMedia](#)

Sample

```
var ballBytes = plugins.file.readFile('d:/ball.jpg');
var ballImage = solutionModel.newMedia('ball.jpg', ballBytes);
var label = form.newLabel('', 10, 10, 100, 100);
label.imageMedia = ballImage;
```

labelFor

Some components can be set to be labels of other components. This is useful in two situations. In table view mode it is used for constructing the header of the table. In record view mode, by setting mnemonics on the label, keyboard shortcuts can be used to set the focus to fields.

Returns

[String](#)

Sample

```
var labelOne = form.newLabel('Label One', 10, 10, 100, 20);
var fieldOne = form.newField('parent_table_id', JSField.TEXT_FIELD, 120, 10, 100, 20);
fieldOne.name = 'fieldOne';
labelOne.labelFor = 'fieldOne';
labelOne.mnemonic = 'O';
```

margin

The margins of the component. They are specified in this order, separated by commas: top, left, bottom, right.

Returns

[String](#)

Sample

```
var label = form.newLabel('Label', 10, 10, 150, 150);
label.background = 'yellow';
label.margin = '10,20,30,40';
```

mediaOptions

Options to scale the image Media object that is displayed inside the component. Can be set to one or a combination of CROP, REDUCE, ENLARGE and KEEPASPECT.

REDUCE will scale down the image if the component is smaller than the image. REDUCE combined with KEEPASPECT will reduce the image, but keep its aspect ratio. This is useful when the component has other proportions than the image.

ENLARGE will scale up the image if the component is larger than the image. ENLARGE combined with KEEPASPECT will scale up the image while keeping its aspect ratio.

CROP will leave the image at its original size. If the component is smaller than the image this will result in only a part of the image showing up.

Returns

[Number](#)

Sample

```
// Load two images, a big one and a small one.
var bigBytes = plugins.file.readFile('d:/big.jpg');
var bigImage = solutionModel.newMedia('big.jpg', bigBytes);
var smallBytes = plugins.file.readFile('d:/small.jpg');
var smallImage = solutionModel.newMedia('small.jpg', smallBytes);
// Put the big image in several small labels, with different media options.
var smallLabelWithBigImageReduceKeepAspect = form.newLabel('', 10, 10, 50, 50);
smallLabelWithBigImageReduceKeepAspect.imageMedia = bigImage;
smallLabelWithBigImageReduceKeepAspect.background = 'yellow';
smallLabelWithBigImageReduceKeepAspect.mediaOptions = SM_MEDIAOPTION.REDUCE | SM_MEDIAOPTION.KEEPASPECT;
var smallLabelWithBigImageReduceNoAspect = form.newLabel('', 70, 10, 50, 50);
smallLabelWithBigImageReduceNoAspect.imageMedia = bigImage;
smallLabelWithBigImageReduceNoAspect.background = 'yellow';
smallLabelWithBigImageReduceNoAspect.mediaOptions = SM_MEDIAOPTION.REDUCE;
var smallLabelWithBigImageCrop = form.newLabel('', 130, 10, 50, 50);
smallLabelWithBigImageCrop.imageMedia = bigImage;
smallLabelWithBigImageCrop.background = 'yellow';
smallLabelWithBigImageCrop.mediaOptions = SM_MEDIAOPTION.CROP;
// Put the small image in several big labels, with different media options.
var bigLabelWithSmallImageEnlargeKeepAspect = form.newLabel('', 10, 70, 200, 100);
bigLabelWithSmallImageEnlargeKeepAspect.imageMedia = smallImage;
bigLabelWithSmallImageEnlargeKeepAspect.background = 'yellow';
bigLabelWithSmallImageEnlargeKeepAspect.mediaOptions = SM_MEDIAOPTION.ENLARGE | SM_MEDIAOPTION.KEEPASPECT;
var bigLabelWithSmallImageEnlargeNoAspect = form.newLabel('', 10, 180, 200, 100);
bigLabelWithSmallImageEnlargeNoAspect.imageMedia = smallImage;
bigLabelWithSmallImageEnlargeNoAspect.background = 'yellow';
bigLabelWithSmallImageEnlargeNoAspect.mediaOptions = SM_MEDIAOPTION.ENLARGE;
var bigLabelWithSmallImageCrop = form.newLabel('', 10, 290, 200, 100);
bigLabelWithSmallImageCrop.imageMedia = smallImage;
bigLabelWithSmallImageCrop.background = 'yellow';
bigLabelWithSmallImageCrop.mediaOptions = SM_MEDIAOPTION.CROP; // This does not do any cropping actually if
the label is larger than the image.
```

mnemonic

The keyboard shortcut that activates this component. A letter must be specified, and the actual shortcut will be combination of ALT + the specified letter.

This property can be used in two ways. Normally the keyboard shortcut activates the onClick event of the component. But if the "labelFor" property is set for the component, then the keyboard shortcut will move the focus to the component whose label this component is.

Returns

[String](#)

Sample

```
var m = form.newMethod('function onClick() { application.output("I was clicked."); }');
var btn = form.newButton('I am a button', 10, 40, 200, 20, m);
btn.mnemonic = 'B'; // When ALT-B is pressed the mouse will respond as if clicked.
var labelOne = form.newLabel('Label One', 10, 10, 100, 20);
var fieldOne = form.newField('parent_table_id', JSField.TEXT_FIELD, 120, 10, 100, 20);
fieldOne.name = 'fieldOne';
labelOne.labelFor = 'fieldOne';
labelOne.mnemonic = 'O'; // When ALT-O is pressed the focus will move to fieldOne.
```

name

The name of the component. Through this name it can also accessed in methods.

Returns

[String](#)

Sample

```
var form = solutionModel.newForm('someForm', 'db:/example_data/parent_table', null, false, 620, 300);
var label = form.newLabel('Label', 10, 10, 150, 150);
label.name = 'myLabel'; // Give a name to the component.
forms['someForm'].controller.show()
// Now use the name to access the component.
forms['someForm'].elements['myLabel'].text = 'Updated text';
```

onAction

The method that is executed when the component is clicked.

Returns

[JSMethod](#)

Sample

```
var doNothingMethod = form.newMethod('function doNothing() { application.output("Doing nothing."); }');
var onClickMethod = form.newMethod('function onClick(event) { application.output("I was clicked at " + event.
getTimestamp()); }');
var onDoubleClickMethod = form.newMethod('function onDoubleClick(event) { application.output("I was double-
clicked at " + event.getTimestamp()); }');
var onRightClickMethod = form.newMethod('function onRightClick(event) { application.output("I was right-
clicked at " + event.getTimestamp()); }');
// At creation the button has the 'doNothing' method as onClick handler, but we'll change that later.
var btn = form.newButton('I am a button', 10, 40, 200, 20, doNothingMethod);
btn.onAction = onClickMethod;
btn.onDoubleClick = onDoubleClickMethod;
btn.onRightClick = onRightClickMethod;
```

onDoubleClick

The method that is executed when the component is double clicked.

Returns

[JSMethod](#)

Sample

```
var doNothingMethod = form.newMethod('function doNothing() { application.output("Doing nothing."); }');
var onClickMethod = form.newMethod('function onClick(event) { application.output("I was clicked at " + event.
getTimestamp()); }');
var onDoubleClickMethod = form.newMethod('function onDoubleClick(event) { application.output("I was double-
clicked at " + event.getTimestamp()); }');
var onRightClickMethod = form.newMethod('function onRightClick(event) { application.output("I was right-
clicked at " + event.getTimestamp()); }');
// At creation the button has the 'doNothing' method as onClick handler, but we'll change that later.
var btn = form.newButton('I am a button', 10, 40, 200, 20, doNothingMethod);
btn.onAction = onClickMethod;
btn.onDoubleClick = onDoubleClickMethod;
btn.onRightClick = onRightClickMethod;
```

onRender

The method that is executed when the component is rendered.

Returns

[JSMethod](#)

Sample

```
label.onRender = form.newMethod('function onRender(event) { event.getElement().bgcolor = \'#00ff00\' }');
```

onRightClick

The method that is executed when the component is right clicked.

Returns

[JSMethod](#)

Sample

```
var doNothingMethod = form.newMethod('function doNothing() { application.output("Doing nothing."); }');
var onClickMethod = form.newMethod('function onClick(event) { application.output("I was clicked at " + event.
getTimestamp()); }');
var onDoubleClickMethod = form.newMethod('function onDoubleClick(event) { application.output("I was double-
clicked at " + event.getTimestamp()); }');
var onRightClickMethod = form.newMethod('function onRightClick(event) { application.output("I was right-
clicked at " + event.getTimestamp()); }');
// At creation the button has the 'doNothing' method as onClick handler, but we'll change that later.
var btn = form.newButton('I am a button', 10, 40, 200, 20, doNothingMethod);
btn.onAction = onClickMethod;
btn.onDoubleClick = onDoubleClickMethod;
btn.onRightClick = onRightClickMethod;
```

printSliding

Enables an element to resize based on its content and/or move when printing.
The component can move horizontally or vertically and can grow or shrink in height and width, based on its content and the content of neighboring components.

Returns

[Number](#)

Sample

```
var form = solutionModel.newForm('printForm', 'db:/example_data/parent_table', null, false, 400, 300);
var slidingLabel = form.newLabel('Some long text here', 10, 10, 5, 5);
slidingLabel.printSliding = SM_PRINT_SLIDING.GROW_HEIGHT | SM_PRINT_SLIDING.GROW_WIDTH;
slidingLabel.background = 'gray';
forms['printForm'].controller.showPrintPreview();
```

printable

Flag that tells if the component should be printed or not when the form is printed.

By default components are printable.

Returns

[Boolean](#)

Sample

```
var form = solutionModel.newForm('printForm', 'db:/example_data/parent_table', null, false, 400, 300);
var printedField = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
var notPrintedField = form.newField('parent_table_id', JSField.TEXT_FIELD, 10, 40, 100, 20);
notPrintedField.printable = false; // This field won't show up in print preview and won't be printed.
forms['printForm'].controller.showPrintPreview();
```

rolloverCursor

The cursor that is shown as the mouse is rolled over the component.

Possible options are DEFAULT and HAND. Note that roll over cursor is not supported in Smart client for list view and tableview forms.

Returns

[Number](#)

Sample

```
var label = form.newLabel('Move the mouse over me', 10, 10, 200, 200);
label.rolloverCursor = SM_CURSOR.HAND_CURSOR;
```

rolloverImageMedia

The roll over image Media object used. It will only work if a property image is also used.

When the mouse is moved over the component, this image Media will be displayed.

When the mouse is moved out of the component, whatever text or image was being initially displayed will be restored. Note that roll over image is not supported in Smart client for list view and tableview forms.

Returns[JSMedia](#)**Sample**

```
var ballBytes = plugins.file.readFile('d:/ball.jpg');
var ballImage = solutionModel.newMedia('ball.jpg', ballBytes);
var mapBytes = plugins.file.readFile('d:/map.jpg');
var mapImage = solutionModel.newMedia('map.jpg', mapBytes);
var label = form.newLabel('', 10, 10, 200, 200);
label.imageMedia = mapImage;
label.rolloverImageMedia = ballImage;
```

rotation

The rotation of the element. You can choose 0, 90, 180, or 270 and the label is rotated accordingly.
This property also applies to buttons and images.

Returns[Number](#)**Sample**

```
var m = form.newMethod('function onClick() { application.output("I was clicked."); }');
var label = form.newLabel('I am a label', 10, 10, 200, 200, m);
label.rotation = 90;
var btn = form.newButton('And I am a button', 10, 220, 200, 20, m);
btn.rotation = 180;
```

showClick

When set, the element will show the clicked state when selected.
Applies to labels and buttons and images only.

Returns[Boolean](#)**Sample**

```
// Create a form method.
var m = form.newMethod('function onClick() { application.output("I was clicked."); }');
// Create a label with the method attached to its onClick event.
var label = form.newLabel('I am a label', 10, 10, 200, 20, m);
// By default the label does not visually react to clicks, but we can enable this.
// Basically the label will now behave as a button does.
label.showClick = true;
// Create a button with the same method attached to its onClick event.
var btn = form.newButton('And I am a button', 10, 40, 200, 20, m);
// By default the button visually reacts to onClick, but we can disable this.
// Then the button will behave like a label does.
btn.showClick = false;
```

showFocus

When set the text of an element will showfocus when selected.
Applies to labels and buttons only.
The text property for the element MUST be filled in first.

NOTE: The TAB key may also be used to select the element, depending on the operating system being used and the selected LAF.

Returns[Boolean](#)

Sample

```
var m = form.newMethod('function onClick() { application.output("I was clicked."); }');
var label = form.newLabel('I am a label', 10, 10, 200, 20, m);
label.showFocus = false;
var btn = form.newButton('And I am a button', 10, 40, 200, 20, m);
btn.showFocus = false;
```

styleClass

The name of the style class that should be applied to this component.

When defining style classes for specific component types, their names must be prefixed according to the type of the component. For example in order to define a class names 'fancy' for fields, in the style definition the class must be named 'field.fancy'. If it would be intended for labels, then it would be named 'label.fancy'. When specifying the class name for a component, the prefix is dropped however. Thus the field or the label will have its styleClass property set to 'fancy' only.

Returns

[String](#)

Sample

```
var form = solutionModel.newForm('printForm', 'db:/example_data/parent_table', null, false, 400, 300);
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
var style = solutionModel.newStyle('myStyle', 'field.fancy { background-color: yellow; }');
form.styleName = 'myStyle'; // First set the style on the form.
field.styleClass = 'fancy'; // Then set the style class on the field.
```

tabSeq

An index that specifies the position of the component in the tab sequence. The components are put into the tab sequence in increasing order of this property. A value of 0 means to use the default mechanism of building the tab sequence (based on their location on the form). A value of -2 means to remove the component from the tab sequence.

Returns

[Number](#)

Sample

```
// Create three fields. Based on how they are placed, by default they will come one
// after another in the tab sequence.
var fieldOne = form.newField('parent_table_id', JSField.TEXT_FIELD, 10, 10, 100, 20);
var fieldTwo = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 40, 100, 20);
var fieldThree = form.newField('parent_table_id', JSField.TEXT_FIELD, 10, 70, 100, 20);
// Set the third field come before the first in the tab sequence, and remove the
// second field from the tab sequence.
fieldOne.tabSeq = 2;
fieldTwo.tabSeq = SM_DEFAULTS.IGNORE;
fieldThree.tabSeq = 1;
```

text

The text that is displayed inside the component.

Returns

[String](#)

Sample

```
// In general the text is specified when creating the component.
var label = form.newLabel('Initial text', 10, 10, 100, 20);
// But it can be changed later if needed.
label.text = 'Changed text';
```

toolTipText

The text displayed when hovering over the component with a mouse cursor.

NOTE:

HTML should be used for multi-line tooltips; you can also use any valid HTML tags to format tooltip text. For example:
<html>This includesbolded text and
BLUE text as well.</html>

Returns[String](#)**Sample**

```
var label = form.newLabel('Stop the mouse over me!', 10, 10, 200, 20);  
label.setToolTipText = 'I\'m the tooltip. Do you see me?';
```

transparent

Flag that tells if the component is transparent or not.

The default value is "false", that is the components are not transparent.

Returns[Boolean](#)**Sample**

```
// Load an image from disk and create a Media object based on it.  
var imageBytes = plugins.file.readFile('d:/ball.jpg');  
var media = solutionModel.newMedia('ball.jpg', imageBytes);  
// Put on the form a label with the image.  
var image = form.newLabel('', 10, 10, 100, 100);  
image.imageMedia = media;  
// Put two fields over the image. The second one will be transparent and the  
// image will shine through.  
var nonTransparentField = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 20, 100, 20);  
var transparentField = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 50, 100, 20);  
transparentField.transparent = true;
```

verticalAlignment

The vertical alignment of the text inside the component. Can be one of TOP, CENTER or BOTTOM.

Note that this property does not refer to the vertical alignment of the component inside the form.

Returns[Number](#)**Sample**

```
var topAlignedLabel = form.newLabel('TOP', 400, 10, 50, 300);  
topAlignedLabel.verticalAlignment = SM_ALIGNMENT.TOP;  
var vCenterAlignedLabel = form.newLabel('CENTER', 460, 10, 50, 300);  
vCenterAlignedLabel.verticalAlignment = SM_ALIGNMENT.CENTER;  
var bottomAlignedLabel = form.newLabel('BOTTOM', 520, 10, 50, 300);  
bottomAlignedLabel.verticalAlignment = SM_ALIGNMENT.BOTTOM;
```

visible

The visible property of the component, default true.

Returns[Boolean](#)**Sample**

```
var form = solutionModel.newForm('printForm', 'db:/example_data/parent_table', null, false, 400, 300);  
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);  
field.visible = false;
```

width

The width in pixels of the component.

Returns

Number

Sample

```
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
application.output('original width: ' + field.width);
application.output('original height: ' + field.height);
field.width = 200;
field.height = 100;
application.output('modified width: ' + field.width);
application.output('modified height: ' + field.height);
```

x

The x coordinate of the component on the form.

Returns

Number

Sample

```
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
application.output('original location: ' + field.x + ', ' + field.y);
field.x = 90;
field.y = 90;
application.output('changed location: ' + field.x + ', ' + field.y);
```

y

The y coordinate of the component on the form.

Returns

Number

Sample

```
var field = form.newField('parent_table_text', JSField.TEXT_FIELD, 10, 10, 100, 20);
application.output('original location: ' + field.x + ', ' + field.y);
field.x = 90;
field.y = 90;
application.output('changed location: ' + field.x + ', ' + field.y);
```

Method Details**getDesignTimeProperty**

Object **getDesignTimeProperty** ()

Get a design-time property of an element.

Returns

Object

Sample

```
var frm = solutionModel.getForm('orders')
var fld = frm.getField('fld')
var prop = fld.getDesignTimeProperty('myprop')
```

getDesignTimePropertyNames

String[] **getDesignTimePropertyNames** ()

Get the design-time properties of an element.

Returns[String\[\]](#)**Sample**

```
var frm = solutionModel.getForm('orders')
var fld = frm.getField('fld')
var propNames = fld.getDesignTimePropertyNames()
```

getUUID[UUID](#) **getUUID ()**

Returns the UUID of this component.

Returns[UUID](#)**Sample**

```
var button_uuid = solutionModel.getForm("my_form").getButton("my_button").getUUID();
application.output(button_uuid.toString());
```

putDesignTimeProperty[Object](#) **putDesignTimeProperty ()**

Set a design-time property of an element.

Returns[Object](#)**Sample**

```
var frm = solutionModel.getForm('orders')
var fld = frm.getField('fld')
fld.putDesignTimeProperty('myprop', 'strawberry')
```

removeDesignTimeProperty[Object](#) **removeDesignTimeProperty ()**

Clear a design-time property of an element.

Returns[Object](#)**Sample**

```
var frm = solutionModel.getForm('orders')
var fld = frm.getField('fld')
fld.removeDesignTimeProperty('myprop')
```