

# SolutionModel

May 06, 2020 14:55

## Return Types

ALIGNMENT ANCHOR BEVELTYPE CURSOR DEFAULTS FONTSTYLE MEDIAOPTION PAGEORIENTATION PRINTSLIDING SCROLLBAR  
TITLEJUSTIFICATION TITLEPOSITION UNITS JSForm JSDataSourceNode JSComponent JSComponent JSCalculation JSField JSLayoutContainer  
JSField JSField JSField JSField JSField JSField JSField JSList JSInsetList JSComponent JSComponent JSMethod JSComponent JSPart  
JSRelation JSRelationItem JSStyle JSComponent JSTab JSMedia JSValueList JSVariable JSPart JSPart JSTitle JSComponent

## Supported Clients

SmartClient WebClient NGClient MobileClient

## Methods Summary

JSComponent	cloneComponent(newName, component)	Makes an exact copy of the given component (JSComponent/JSField/JSLabel) and gives it a new name.
JSComponent	cloneComponent(newName, component, newParentForm)	Makes an exact copy of the given component (JSComponent/JSField/JSLabel), gives it a new name and moves it to a new parent form, specified as a parameter.
JSForm	cloneForm(newName, jsForm)	Makes an exact copy of the given form and gives it the new name.
String	createBevelBorder(bevel_type, highlight_outer_color, highlight_inner_color, shadow_outer_color, shadow_inner_color)	Create a bevel border string.
String	createEmptyBorder(top_width, right_width, bottom_width, left_width)	Create an empty border string.
String	createEtchedBorder(bevel_type, highlight_color, shadow_color)	Create an etched border string.
String	createFont(name, style, size)	Create a font string.
String	createLineBorder(thick, color)	Create a line border string.
String	createMatteBorder(top_width, right_width, bottom_width, left_width, color)	Create a matte border string.
String	createPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin)	Create a page format string.
String	createPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin, orientation)	Create a page format string.
String	createPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin, orientation, units)	Create a page format string.
String	createRoundedBorder(top_width, right_width, bottom_width, left_width, top_color, right_color, bottom_color, left_color, rounding_radius, border_style)	Create a special matte border string.
String	createSpecialMatteBorder(top_width, right_width, bottom_width, left_width, top_color, right_color, bottom_color, left_color, rounding_radius, dash_pattern)	Create a special matte border string.
String	createTitledBorder(title_text, font, color, title_justification, title_position)	Create a titled border string.
Array	getAllRelations()	Gets an array of all relations.
JSDataSourceNode	getDataSourceNode(dataSource)	Gets the specified data source node and returns information about the form (see JSDataSourceNode node).
JSForm	getForm(name)	Gets the specified form object and returns information about the form (see JSForm node).
Array	getForms()	Get an array of all forms.
Array	getForms(datasource)	Get an array of forms, that are all based on datasource/servername.
Array	getForms(server, tablename)	Get an array of forms, that are all based on datasource/servername and tablename.
JSMethod	getGlobalMethod(scopeName, name)	Gets an existing global method by the specified name.
Array	getGlobalMethods()	The list of all global methods.
Array	getGlobalMethods(scopeName)	The list of all global methods.
JSVariable	getGlobalVariable(scopeName, name)	Gets an existing global variable by the specified name.
Array	getGlobalVariables()	Gets an array of all global variables.
Array	getGlobalVariables(scopeName)	Gets an array of all global variables.
JSMedia	getMedia(name)	Gets the specified media object; can be assigned to a button/label.
Array	getMediaList()	Gets the list of all media objects.
Object	getObjectByUUID(uuid)	Retrieves an element by its uuid.
JSRelation	getRelation(name)	Gets an existing relation by the specified name and returns a JSRelation Object.
Array	getRelations(datasource)	Gets an array of all relations; or an array of all global relations if the specified table is NULL.
Array	getRelations(servername, tablename)	Gets an array of all relations; or an array of all global relations if the specified table is NULL.

Array	<a href="#">getScopeNames()</a>	Gets an array of all scope names used.
JSStyle	<a href="#">getStyle(name)</a>	Gets the style specified by the given name.
JSValueList	<a href="#">getValueList(name)</a>	Gets an existing valuelist by the specified name and returns a JSValueList Object that can be assigned to a field.
Array	<a href="#">getValueLists()</a>	Gets an array of all valuelists for the currently active solution.
JSForm	<a href="#">newForm(name)</a>	Creates a new JSForm Object.
JSForm	<a href="#">newForm(name, isResponsive)</a>	Create a responsive form:
JSForm	<a href="#">newForm(name, superForm)</a>	Creates a new form with the given JSForm as its super form.
JSForm	<a href="#">newForm(name, superForm, isResponsive)</a>	Creates a new form with the given JSForm as its super form.
JSForm	<a href="#">newForm(name, dataSource, isResponsive)</a>	Create a responsive form:
JSForm	<a href="#">newForm(name, dataSource, styleName, show_in_menu, width, height)</a>	Creates a new JSForm Object.
JSForm	<a href="#">newForm(name, serverName, tableName, styleName, show_in_menu, width, height)</a>	Creates a new JSForm Object.
JSMETHOD	<a href="#">newGlobalMethod(scopeName, code)</a>	Creates a new global method with the specified code in a scope.
JSVariable	<a href="#">newGlobalVariable(scopeName, name, type)</a>	Creates a new global variable with the specified name and number type.
JSMedia	<a href="#">newMedia(name, bytes)</a>	Creates a new media object that can be assigned to a label or a button.
JSMedia	<a href="#">newMedia(name, bytes)</a>	Creates a new media object for things like a CSS or LESS file that can be set as the clients solution style.
JSRelation	<a href="#">newRelation(name, primaryDataSource, foreignDataSource, joinType)</a>	Creates a new JSRelation Object with a specified name; includes the primary datasource, foreign datasource and the type of join for the new relation.
JSStyle	<a href="#">newStyle(name, content)</a>	Creates a new style with the given css content string under the given name.
JSValueList	<a href="#">newValueList(name, type)</a>	Creates a new valuelist with the specified name and number type.
Boolean	<a href="#">removeForm(name)</a>	Removes the specified form during the persistent connected client session.
Boolean	<a href="#">removeGlobalMethod(scopeName, name)</a>	Removes the specified global method.
Boolean	<a href="#">removeGlobalVariable(scopeName, name)</a>	Removes the specified global variable.
Boolean	<a href="#">removeMedia(name)</a>	Removes the media item specified by name.
Boolean	<a href="#">removeRelation(name)</a>	Removes the relation specified by name.
Boolean	<a href="#">removeStyle(name)</a>	Removes the specified style.
Boolean	<a href="#">removeValueList(name)</a>	Removes the specified valuelist.
JSForm	<a href="#">revertForm(name)</a>	Reverts the specified form to the original (blueprint) version of the form; will result in an exception error if the form is not an original form.
JSMETHOD	<a href="#">wrapMethodWithArguments(method, args)</a>	Get a JSMETHOD instance with arguments to be assigned to an event.

## Methods Details

### cloneComponent(newName, component)

Makes an exact copy of the given component (JSComponent/JSField/JSLabel) and gives it a new name.

#### Parameters

[String](#) newName the new name of the cloned component  
[JSComponent](#) component the component to clone

#### Returns

[JSComponent](#) the exact copy of the given component

#### Supported Clients

SmartClient, WebClient, NGClient

#### Sample

```
// get an existing field to clone.
var field = solutionModel.getForm("formWithField").getField("fieldName");
// make a clone/copy of the field
var clone = solutionModel.cloneComponent("clonedField",field);
```

### cloneComponent(newName, component, newParentForm)

Makes an exact copy of the given component (JSComponent/JSField/JSLabel), gives it a new name and moves it to a new parent form, specified as a parameter.

#### Parameters

[String](#) newName the new name of the cloned component  
[JSComponent](#) component the component to clone  
[JSForm](#) newParentForm the new parent form

#### Returns

[JSComponent](#) the exact copy of the given component

#### Supported Clients

SmartClient, WebClient, NGClient

**Sample**

```
// get an existing field to clone.
var field = solutionModel.getForm("formWithField").getField("fieldName");
// get the target form for the copied/cloned field
var form = solutionModel.getForm("targetForm");
// make a clone/copy of the field and re parent it to the target form.
var clone = solutionModel.cloneComponent("clonedField",field,form);
// show it
forms["targetForm"].controller.show();
```

**cloneForm(newName, jsForm)**

Makes an exact copy of the given form and gives it the new name.

**Parameters**

**String** newName the new name for the form clone  
**JSForm** jsForm the form to be cloned

**Returns**

**JSForm** a JSForm

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
// get an existing form
var form = solutionModel.getForm("existingForm")
// make a clone/copy from it
var clone = solutionModel.cloneForm("clonedForm", form)
// add a new label to the clone
clone.newLabel("added label",50,50,80,20);
// show it
forms["clonedForm"].controller.show();
```

**createBevelBorder(bevel\_type, highlight\_outer\_color, highlight\_inner\_color, shadow\_outer\_color, shadow\_inner\_color)**

Create a bevel border string.

**Parameters**

**Number** bevel\_type bevel border type (SM\_BEVELTYPE.RAISED or SM\_BEVELTYPE.LOWERED)  
**String** highlight\_outer\_color bevel border highlight outer color  
**String** highlight\_inner\_color bevel border highlight inner color  
**String** shadow\_outer\_color bevel border shadow outer color  
**String** shadow\_inner\_color bevel border shadow outer color

**Returns**

**String**

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.borderColor = solutionModel.createBevelBorder(SM_BEVELTYPE.RAISED,'#ff0000','#00ff00','#ff0000','#00ff00');
```

**createEmptyBorder(top\_width, right\_width, bottom\_width, left\_width)**

Create an empty border string.

**Parameters**

**Number** top\_width top width of empty border in pixels  
**Number** right\_width right width of empty border in pixels  
**Number** bottom\_width bottom width of empty border in pixels  
**Number** left\_width left width of empty border in pixels

**Returns**

**String**

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.borderType = solutionModel.createEmptyBorder(1,1,1,1);
```

**createEtchedBorder(bevel\_type, highlight\_color, shadow\_color)**

Create an etched border string.

**Parameters**

**Number** bevel\_type bevel border type  
**String** highlight\_color bevel border highlight color  
**String** shadow\_color bevel border shadow color

**Returns****String****Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.borderType = solutionModel.createEtchedBorder(SM_BEVELTYPE.RAISED, '#ff0000', '#00ff00');
```

**createFont(name, style, size)**

Create a font string.

**Parameters**

**String** name the name of the font  
**Number** style the style of the font (PLAIN, BOLD, ITALIC or BOLD+ITALIC)  
**Number** size the font size

**Returns****String****Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
var component = form.getComponent("someComponent");
component.fontType = solutionModel.createFont('Arial', SM_FONTSTYLE.BOLD, 14);
```

**createLineBorder(thick, color)**

Create a line border string.

**Parameters**

**Number** thick border thickness in pixels  
**String** color color of the line border

**Returns****String****Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.borderType = solutionModel.createLineBorder(1, '#ff0000');
```

**createMatteBorder(top\_width, right\_width, bottom\_width, left\_width, color)**

Create a matte border string.

**Parameters**

Number	top_width	top width of matte border in pixels
Number	right_width	right width of matte border in pixels
Number	bottom_width	bottom width of matte border in pixels
Number	left_width	left width of matte border in pixels
String	color	border color

**Returns**

String

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.borderType = solutionModel.createMatteBorder(1,1,1,1,"#00ff00");
```

**createPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin)**

Create a page format string.

Note: The unit specified for width, height and all margins MUST be the same.

**Parameters**

Number	width	the specified width of the page to be printed.
Number	height	the specified height of the page to be printed.
Number	leftmargin	the specified left margin of the page to be printed.
Number	rightmargin	the specified right margin of the page to be printed.
Number	topmargin	the specified top margin of the page to be printed.
Number	bottommargin	the specified bottom margin of the page to be printed.

**Returns**

String

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.defaultPageFormat = solutionModel.createPageFormat(612,792,72,72,72,72,SM_ORIENTATION.PORTRAIT,SM_UNITS.PIXELS);
```

**createPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin, orientation)**

Create a page format string.

Note: The unit specified for width, height and all margins MUST be the same.

**Parameters**

Number	width	the specified width of the page to be printed.
Number	height	the specified height of the page to be printed.
Number	leftmargin	the specified left margin of the page to be printed.
Number	rightmargin	the specified right margin of the page to be printed.
Number	topmargin	the specified top margin of the page to be printed.
Number	bottommargin	the specified bottom margin of the page to be printed.
Number	orientation	the specified orientation of the page to be printed; the default is Portrait mode

**Returns**

String

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.defaultPageFormat = solutionModel.createPageFormat(612,792,72,72,72,72,SM_ORIENTATION.PORTRAIT,SM_UNITS.PIXELS);
```

**createPageFormat(width, height, leftmargin, rightmargin, topmargin, bottommargin, orientation, units)**

---

Create a page format string.

Note: The unit specified for width, height and all margins MUST be the same.

#### Parameters

Number width	the specified width of the page to be printed.
Number height	the specified height of the page to be printed.
Number leftmargin	the specified left margin of the page to be printed.
Number rightmargin	the specified right margin of the page to be printed.
Number topmargin	the specified top margin of the page to be printed.
Number bottommargin	the specified bottom margin of the page to be printed.
Number orientation	the specified orientation of the page to be printed; the default is Portrait mode
Number units	the specified units for the width and height of the page to be printed; the default is pixels

#### Returns

String

#### Supported Clients

SmartClient,WebClient,NGClient

#### Sample

```
var form = solutionModel.getForm("someForm");
form.defaultPageFormat = solutionModel.createPageFormat(612,792,72,72,72,72,SM_ORIENTATION.PORTRAIT,SM_UNITS.PIXELS);
```

### **createRoundedBorder(top\_width, right\_width, bottom\_width, left\_width, top\_color, right\_color, bottom\_color, left\_color, rounding\_radius, border\_style)**

Create a special matte border string.

#### Parameters

Number top_width	top width of matte border in pixels
Number right_width	right width of matte border in pixels
Number bottom_width	bottom width of matte border in pixels
Number left_width	left width of matte border in pixels
String top_color	top border color
String right_color	right border color
String bottom_color	bottom border color
String left_color	left border color
Array rounding_radius	array with width/height of the arc to round the corners
Array border_style	the border styles for the four margins(top/left/bottom/left)

#### Returns

String

#### Supported Clients

SmartClient,WebClient,NGClient

#### Sample

```
var form = solutionModel.getForm("someForm");
// create a rectangle border (no rounded corners) and continous line
form.borderType = solutionModel.createSpecialMatteBorder(1,1,1,1,"#00ff00","#00ff00","#00ff00","#00ff00",0,null);
// create a border with rounded corners and dashed line (25 pixels drawn, then 25 pixels skipped)
// rounding_radius is an array of up to 8 numbers, order is: top-left,top-right,bottom-right,bottom-left
// (repeated twice - for width and height)
// form.borderType = solutionModel.createSpecialMatteBorder(1,1,1,1,"#00ff00","#00ff00","#00ff00","#00ff00",new
Array(10,10,10,10),new Array(25,25));
```

### **createSpecialMatteBorder(top\_width, right\_width, bottom\_width, left\_width, top\_color, right\_color, bottom\_color, left\_color, rounding\_radius, dash\_pattern)**

Create a special matte border string.

**Parameters**

Number	top_width	top width of matte border in pixels
Number	right_width	right width of matte border in pixels
Number	bottom_width	bottom width of matte border in pixels
Number	left_width	left width of matte border in pixels
String	top_color	top border color
String	right_color	right border color
String	bottom_color	bottom border color
String	left_color	left border color
Number	rounding_radius	width of the arc to round the corners
Array	dash_pattern	the dash pattern of border stroke

**Returns**

String

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
// create a rectangle border (no rounded corners) and continous line
form.borderType = solutionModel.createSpecialMatteBorder(1,1,1,1,"#00ff00","#00ff00","#00ff00","#00ff00",0,null);
// create a border with rounded corners and dashed line (25 pixels drawn, then 25 pixels skipped)
// form.borderType = solutionModel.createSpecialMatteBorder(1,1,1,1,"#00ff00","#00ff00","#00ff00","#00ff00",10,
new Array(25,25));
```

**createTitledBorder(title\_text, font, color, title\_justification, title\_position)**

Create a titled border string.

**Parameters**

String	title_text	the text from border
String	font	title text font string
String	color	border color
Number	title_justification	title text justification
Number	title_position	bevel title text position

**Returns**

String

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.getForm("someForm");
form.borderType = solutionModel.createTitledBorder('Test',solutionModel.createFont('Arial',SM_FONTSTYLE.PLAIN,
10),'#ff0000',SM_TITLEJUSTIFICATION.CENTER,SM_TITLEPOSITION.TOP);
```

**getAllRelations()**

Gets an array of all relations.

**Returns**

Array an array of all relations (all elements in the array are of type JSRelation)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var relations = solutionModel.getAllRelations();
if (relations.length != 0)
    for (var i in relations)
        application.output(relations[i].name);
```

**getDataSourceNode(dataSource)**

Gets the specified data source node and returns information about the form (see JSDataSourceNode node). The JSDataSourceNode holds all calculations and foundset methods.

**Parameters**

[String](#) dataSource table data source

**Returns**

[JSDataSourceNode](#) a JSDataSourceNode

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var dsnode = solutionModel.getDataSourceNode('db:/example_data/customers');
var c = dsnode.getCalculation("myCalculation");
application.output("Name: " + c.getName() + ", Stored: " + c.isStored());
```

**getForm(name)**

Gets the specified form object and returns information about the form (see JSForm node).

**Parameters**

[String](#) name the specified name of the form

**Returns**

[JSForm](#) a JSForm

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var myForm = solutionModel.getForm('existingFormName');
//get the style of the form (for all other properties see JSForm node)
var styleName = myForm.styleName;
```

**getForms()**

Get an array of all forms.

**Returns**

[Array](#) an array of JSForm type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var forms = solutionModel.getForms()
for (var i in forms)
    application.output(forms[i].name)
```

**getForms(datasource)**

Get an array of forms, that are all based on datasource/servername.

**Parameters**

[String](#) datasource the datasource or servername

**Returns**

[Array](#) an array of JSForm type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var forms = solutionModel.getForms(datasource)
for (var i in forms)
    application.output(forms[i].name)
```

**getForms(server, tablename)**

Get an array of forms, that are all based on datasource/servername and tablename.



**Parameters**

[String](#) server the datasource or servename

[String](#) tablename the tablename

**Returns**

[Array](#) an array of JSForm type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var forms = solutionModel.getForms(datasource,tablename)
for (var i in forms)
    application.output(forms[i].name)
```

**getGlobalMethod(scopeName, name)**

Gets an existing global method by the specified name.

**Parameters**

[String](#) scopeName the scope in which the method is searched

[String](#) name the name of the specified global method

**Returns**

[JSMethod](#) a JSMethod

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var method = solutionModel.getGlobalMethod('globals', 'nameOfGlobalMethod');
if (method != null) application.output(method.code);
```

**getGlobalMethods()**

The list of all global methods.

**Returns**

[Array](#) an array of JSMethod type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var methods = solutionModel.getGlobalMethods('globals');
for (var x in methods)
    application.output(methods[x].getName());
```

**getGlobalMethods(scopeName)**

The list of all global methods.

**Parameters**

[String](#) scopeName limit to global methods of specified scope name

**Returns**

[Array](#) an array of JSMethod type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var methods = solutionModel.getGlobalMethods('globals');
for (var x in methods)
    application.output(methods[x].getName());
```

**getGlobalVariable(scopeName, name)**

---

Gets an existing global variable by the specified name.

**Parameters**

[String](#) scopeName the scope in which the variable is searched  
[String](#) name the specified name of the global variable

**Returns**

[JSVariable](#) a JSVariable

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var globalVariable = solutionModel.getGlobalVariable('globals', 'globalVariableName');
application.output(globalVariable.name + " has the default value of " + globalVariable.defaultValue);
```

**getGlobalVariables()**

Gets an array of all global variables.

**Returns**

[Array](#) an array of JSVariable type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var globalVariables = solutionModel.getGlobalVariables('globals');
for (var i in globalVariables)
    application.output(globalVariables[i].name + " has the default value of " + globalVariables[i].
defaultValue);
```

**getGlobalVariables(scopeName)**

Gets an array of all global variables.

**Parameters**

[String](#) scopeName limit to global vars of specified scope name

**Returns**

[Array](#) an array of JSVariable type elements

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var globalVariables = solutionModel.getGlobalVariables('globals');
for (var i in globalVariables)
    application.output(globalVariables[i].name + " has the default value of " + globalVariables[i].
defaultValue);
```

**getMedia(name)**

Gets the specified media object; can be assigned to a button/label.

**Parameters**

[String](#) name the specified name of the media object

**Returns**

[JSMedia](#) a JSMedia element

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var myMedia = solutionModel.getMedia('button01.gif')
//now set the imageMedia property of your label or button
//myButton.imageMedia = myMedia
// OR
//myLabel.imageMedia = myMedia
```

**getMediaList()**

Gets the list of all media objects.

**Returns**

[Array](#) a list with all the media objects.

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var mediaList = solutionModel.getMediaList();
if (mediaList.length != 0 && mediaList != null) {
    for (var x in mediaList) {
        application.output(mediaList[x]);
    }
}
```

**getObjectByUUID(uuid)**

Retrieves an element by its uuid.

**Parameters**

[Object](#) uuid element uuid

**Returns**

[Object](#) found element

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
solutionModel.getObjectByUUID(uuid)
```

**getRelation(name)**

Gets an existing relation by the specified name and returns a JSRelation Object.

**Parameters**

[String](#) name the specified name of the relation

**Returns**

[JSRelation](#) a JSRelation

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var relation = solutionModel.getRelation('name');
application.output("The primary server name is " + relation.primaryServerName);
application.output("The primary table name is " + relation.primaryTableName);
application.output("The foreign table name is " + relation.foreignTableName);
application.output("The relation items are " + relation.getRelationItems());
```

**getRelations(datasource)**

Gets an array of all relations; or an array of all global relations if the specified table is NULL.

**Parameters**

[String](#) datasource the specified name of the datasource for the specified table

**Returns**

[Array](#) an array of all relations (all elements in the array are of type JSRelation)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var relations = solutionModel.getRelations('server_name','table_name');
if (relations.length != 0)
    for (var i in relations)
        application.output(relations[i].name);
```

**getRelations(servername, tablename)**

Gets an array of all relations; or an array of all global relations if the specified table is NULL.

**Parameters**

[String](#) servername the specified name of the server for the specified table

[String](#) tablename the specified name of the table

**Returns**

[Array](#) an array of all relations (all elements in the array are of type JSRelation)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var relations = solutionModel.getRelations('server_name','table_name');
if (relations.length != 0)
    for (var i in relations)
        application.output(relations[i].name);
```

**getScopeNames()**

Gets an array of all scope names used.

**Returns**

[Array](#) an array of String scope names

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var scopeNames = solutionModel.getScopeNames();
for (var name in scopeNames)
    application.output(name);
```

**getStyle(name)**

Gets the style specified by the given name.

**Parameters**

[String](#) name the specified name of the style

**Returns**

[JSStyle](#) a JSStyle

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var style = solutionModel.getStyle('my_existing_style')
style.content = 'combobox { color: #0000ff;font: italic 10pt "Verdana";}'
```

**getValueList(name)**

---

Gets an existing valuelist by the specified name and returns a JSValueList Object that can be assigned to a field.

NOTE: Changes to valuelist should be done before showing any form that has component using the valuelist.

#### Parameters

[String](#) name the specified name of the valuelist

#### Returns

[JSValueList](#) a JSValueList object

#### Supported Clients

SmartClient,WebClient,NGClient,MobileClient

#### Sample

```
var myValueList = solutionModel.getValueList('myValueListHere')
//now set the valueList property of your field
//myField.valuelist = myValueList
```

### getValueLists()

Gets an array of all valuelists for the currently active solution.

NOTE: Changes to valuelist should be done before showing any form that has component using the valuelist.

#### Returns

[Array](#) an array of JSValueList objects

#### Supported Clients

SmartClient,WebClient,NGClient,MobileClient

#### Sample

```
var valueLists = solutionModel.getValueLists();
if (valueLists != null && valueLists.length != 0)
    for (var i in valueLists)
        application.output(valueLists[i].name);
```

### newForm(name)

Creates a new JSForm Object.

#### Parameters

[String](#) name the specified name of the form

#### Returns

[JSForm](#) a new JSForm object

#### Supported Clients

SmartClient,WebClient,NGClient

#### Sample

```
var myForm = solutionModel.newForm('newForm')
```

### newForm(name, isResponsive)

Create a responsive form:

#### Parameters

[String](#) name The name of the new form

[Boolean](#) isResponsive if true will create an responsive form, otherwise an absolute layout form

#### Returns

[JSForm](#) a new JSForm object

#### Supported Clients

SmartClient,WebClient,NGClient

**Sample**

```
var frm = solutionModel.newForm('test', true);
var c = frm.newLayoutContainer(1);
```

**newForm(name, superForm)**

Creates a new form with the given JSForm as its super form.

**Parameters**

**String** name The name of the new form  
**JSForm** superForm the super form that will extended from, see JSform.setExtendsForm();

**Returns**

**JSForm** a new JSForm object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
//creates 2 forms with elements on them; shows the parent form, waits 2 seconds and shows the child form
var mySuperForm = solutionModel.newForm('mySuperForm', 'db:/my_server/my_table', null, false, 800, 600);
var label1 = mySuperForm.newLabel('LabelName', 20, 20, 120, 30);
label1.text = 'DataProvider';
label1.background = 'red';
mySuperForm.newTextField('myDataProvider', 140, 20, 140,20);
forms['mySuperForm'].controller.show();
application.sleep(2000);
var mySubForm = solutionModel.newForm('mySubForm', mySuperForm);
var label2 = mySuperForm.newLabel('SubForm Label', 20, 120, 120, 30);
label2.background = 'green';
forms['mySuperForm'].controller.recreateUI();
forms['mySubForm'].controller.show();
```

**newForm(name, superForm, isResponsive)**

Creates a new form with the given JSForm as its super form.  
 Use this function in the case when the super form is a logical form (no parts/UI).

**Parameters**

**String** name The name of the new form  
**JSForm** superForm the super form that will extended from, see JSform.setExtendsForm();  
**Boolean** isResponsive;

**Returns**

**JSForm** a new JSForm object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
//creates 2 forms with elements on them; shows the parent form, waits 2 seconds and shows the child form
var mySuperForm = solutionModel.newForm('mySuperForm', 'db:/my_server/my_table', null, false, 800, 600);
var label1 = mySuperForm.newLabel('LabelName', 20, 20, 120, 30);
label1.text = 'DataProvider';
label1.background = 'red';
mySuperForm.newTextField('myDataProvider', 140, 20, 140,20);
forms['mySuperForm'].controller.show();
application.sleep(2000);
var mySubForm = solutionModel.newForm('mySubForm', mySuperForm);
var label2 = mySuperForm.newLabel('SubForm Label', 20, 120, 120, 30);
label2.background = 'green';
forms['mySuperForm'].controller.recreateUI();
forms['mySubForm'].controller.show();
```

**newForm(name, dataSource, isResponsive)**

Create a responsive form:

**Parameters**

**String** name        The name of the new form  
**String** dataSource    the form datasource  
**Boolean** isResponsive if true will create an responsive form, otherwise an absolute layout form

**Returns**

**JSForm** a new JSForm object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var frm = solutionModel.newForm('test','db:/my_server/my_table', true);
var c = frm.newLayoutContainer(1);
```

**newForm(name, dataSource, styleName, show\_in\_menu, width, height)**

Creates a new JSForm Object.

NOTE: See the JSForm node for more information about form objects that can be added to the new form.

**Parameters**

**String** name        the specified name of the form  
**String** dataSource    the specified name of the datasource for the specified table  
**String** styleName    the specified style  
**Boolean** show\_in\_menu if true show the name of the new form in the menu; or false for not showing  
**Number** width        the width of the form in pixels  
**Number** height       the height of the form in pixels

**Returns**

**JSForm** a new JSForm object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var myForm = solutionModel.newForm('newForm', 'db:/my_server/my_table', 'myStyleName', false, 800, 600)
//now you can add stuff to the form (under JSForm node)
//add a label
myForm.newLabel('Name', 20, 20, 120, 30)
//add a "normal" text entry field
myForm.newTextField('dataProviderNameHere', 140, 20, 140,20)
```

**newForm(name, serverName, tableName, styleName, show\_in\_menu, width, height)**

Creates a new JSForm Object.

NOTE: See the JSForm node for more information about form objects that can be added to the new form.

**Parameters**

**String** name        the specified name of the form  
**String** serverName    the specified name of the server for the specified table  
**String** tableName    the specified name of the table  
**String** styleName    the specified style  
**Boolean** show\_in\_menu if true show the name of the new form in the menu; or false for not showing  
**Number** width        the width of the form in pixels  
**Number** height       the height of the form in pixels

**Returns**

**JSForm** a new JSForm object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var myForm = solutionModel.newForm('newForm', 'my_server', 'my_table', 'myStyleName', false, 800, 600)
//With only a datasource:
//var myForm = solutionModel.newForm('newForm', datasource, 'myStyleName', false, 800, 600)
//now you can add stuff to the form (under JSForm node)
//add a label
myForm.newLabel('Name', 20, 20, 120, 30)
//add a "normal" text entry field
myForm.newTextField('dataProviderNameHere', 140, 20, 140,20)
```

**newGlobalMethod(scopeName, code)**

Creates a new global method with the specified code in a scope.

**Parameters**

**String** scopeName the scope in which the method is created  
**String** code the specified code for the global method

**Returns**

**JSMethod** a JSMethod object

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var method = solutionModel.newGlobalMethod('globals', 'function myglobalmethod(){foundset.newRecord()}')
```

**newGlobalVariable(scopeName, name, type)**

Creates a new global variable with the specified name and number type.

NOTE: The global variable number type is based on the value assigned from the SolutionModel-JSVariable node; for example: JSVariable.INTEGER.

**Parameters**

**String** scopeName the scope in which the variable is created  
**String** name the specified name for the global variable  
**Number** type the specified number type for the global variable

**Returns**

**JSVariable** a JSVariable object

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var myGlobalVariable = solutionModel.newGlobalVariable('globals', 'newGlobalVariable', JSVariable.INTEGER);
myGlobalVariable.defaultValue = 12;
//myGlobalVariable.defaultValue = "{a:'First letter',b:'Second letter'}" // an js object, type must be media.
//myGlobalVariable.defaultValue = "some text"; // Use two pairs of quotes if you want to assign a String as
default value.
```

**newMedia(name, bytes)**

Creates a new media object that can be assigned to a label or a button.

**Parameters**

**String** name The name of the new media  
**Array** bytes The content

**Returns**

**JSMedia** a JSMedia object

**Supported Clients**

SmartClient,WebClient,NGClient



**Sample**

```
var myMedia = solutionModel.newMedia('button01.gif',bytes)
//now set the imageMedia property of your label or button
//myButton.imageMedia = myMedia
// OR
//myLabel.imageMedia = myMedia
```

**newMedia(name, bytes)**

Creates a new media object for things like a CSS or LESS file that can be set as the clients solution style. The stringContents is converted to bytes through the UTF-8 charset.

**Parameters**

**String** name The name of the new media  
**Object** bytes The content

**Returns**

**JSMedia** a JSMedia object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var myMedia = solutionModel.newMedia('button01.gif',stringContent)
//now set the imageMedia property of your label or button
//myButton.imageMedia = myMedia
// OR
//myLabel.imageMedia = myMedia
```

**newRelation(name, primaryDataSource, foreignDataSource, joinType)**

Creates a new JSRelation Object with a specified name; includes the primary datasource, foreign datasource and the type of join for the new relation.

**Parameters**

**String** name the specified name of the new relation  
**String** primaryDataSource the specified name of the primary datasource  
**String** foreignDataSource the specified name of the foreign datasource  
**Number** joinType the type of join for the new relation; JSRelation.INNER\_JOIN, JSRelation.LEFT\_OUTER\_JOIN

**Returns**

**JSRelation** a JSRelation object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var rel = solutionModel.newRelation('myRelation', myPrimaryDataSource, myForeignDataSource, JSRelation.
INNER_JOIN);
application.output(rel.getRelationItems());
```

**newStyle(name, content)**

Creates a new style with the given css content string under the given name.

NOTE: Will throw an exception if a style with that name already exists.

**Parameters**

**String** name the name of the new style  
**String** content the css content of the new style

**Returns**

**JSStyle** a JSStyle object

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var form = solutionModel.newForm('myForm', 'db:/my_server/my_table', null, true, 1000, 800);
if (form.transparent == false)
{
    var style = solutionModel.newStyle('myStyle', 'form { background-color: yellow; }');
    style.text = style.text + 'field { background-color: blue; }';
    form.styleName = 'myStyle';
}
var field = form.newField('columnTextDataProvider', JSField.TEXT_FIELD, 100, 100, 100, 50);
forms['myForm'].controller.show();
```

**newValueList(name, type)**

Creates a new valuelist with the specified name and number type.

**Parameters**

**String** name the specified name for the valuelist

**Number** type the specified number type for the valuelist; may be JSValueList.CUSTOM\_VALUES, JSValueList.DATABASE\_VALUES, JSValueList.EMPTY\_VALUE\_ALWAYS, JSValueList.EMPTY\_VALUE\_NEVER

**Returns**

**JSValueList** a JSValueList object

**Supported Clients**

SmartClient, WebClient, NGClient, MobileClient

**Sample**

```
var vl1 = solutionModel.newValueList("customText", JSValueList.CUSTOM_VALUES);
vl1.customValues = "customvalue1\ncustomvalue2";
var vl2 = solutionModel.newValueList("customid", JSValueList.CUSTOM_VALUES);
vl2.customValues = "customvalue1|1\ncustomvalue2|2";
var form = solutionModel.newForm("customValueListForm", controller.getDataSource(), null, true, 300, 300);
var combo1 = form.newComboBox("scopes.globals.text", 10, 10, 120, 20);
combo1.valuelist = vl1;
var combo2 = form.newComboBox("scopes.globals.id", 10, 60, 120, 20);
combo2.valuelist = vl2;
```

**removeForm(name)**

Removes the specified form during the persistent connected client session.

NOTE: Make sure you call history.remove first in your Servoy method (script).

**Parameters**

**String** name the specified name of the form to remove

**Returns**

**Boolean** true is form has been removed, false if form could not be removed

**Supported Clients**

SmartClient, WebClient, NGClient, MobileClient

**Sample**

```
//first remove it from the current history, to destroy any active form instance
var success = history.removeForm('myForm')
//removes the named form from this session, please make sure you called history.remove() first
if(success)
{
    solutionModel.removeForm('myForm')
}
```

**removeGlobalMethod(scopeName, name)**

Removes the specified global method.

**Parameters**

**String** scopeName the scope in which the method is declared

**String** name the name of the global method to be removed

**Returns**

**Boolean** true if the removal was successful, false otherwise

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var m1 = solutionModel.newGlobalMethod('globals', 'function myglobalmethod1(){application.output("Global Method 1");}');
var m2 = solutionModel.newGlobalMethod('globals', 'function myglobalmethod2(){application.output("Global Method 2");}');

var success = solutionModel.removeGlobalMethod('globals', 'myglobalmethod1');
if (success == false) application.output('!!! myglobalmethod1 could not be removed !!!');

var list = solutionModel.getGlobalMethods('globals');
for (var i = 0; i < list.length; i++) {
    application.output(list[i].code);
}
```

**removeGlobalVariable(scopeName, name)**

Removes the specified global variable.

**Parameters**

**String** scopeName the scope in which the variable is declared  
**String** name the name of the global variable to be removed

**Returns**

**Boolean** true if the removal was successful, false otherwise

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var v1 = solutionModel.newGlobalVariable('globals', 'globalVar1', JSVariable.INTEGER);
var v2 = solutionModel.newGlobalVariable('globals', 'globalVar2', JSVariable.TEXT);

var success = solutionModel.removeGlobalVariable('globals', 'globalVar1');
if (success == false) application.output('!!! globalVar1 could not be removed !!!');

var list = solutionModel.getGlobalVariables('globals');
for (var i = 0; i < list.length; i++) {
    application.output(list[i].name + '[' + list[i].variableType + ']: ' + list[i].variableType);
}
```

**removeMedia(name)**

Removes the media item specified by name.

**Parameters**

**String** name the name of the media item to be removed

**Returns**

**Boolean** true if the removal was successful, false otherwise

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```

var bytes1 = plugins.file.readFile('D:/Imgs/imagel.png');
var image1 = solutionModel.newMedia('imagel.png', bytes1);
var bytes2 = plugins.file.readFile('D:/Imgs/image2.jpg');
var image2 = solutionModel.newMedia('image2.jpg',bytes2);
var bytes3 = plugins.file.readFile('D:/Imgs/image3.jpg');
var image3 = solutionModel.newMedia('image3.jpg',bytes3);

var f = solutionModel.newForm("newForm",databaseManager.getDataSource('example_data', 'orders'),null,false,
500,350);
var l = f.newLabel('', 20, 70, 300, 200);
l.imageMedia = image1;
l.borderType = solutionModel.createLineBorder(4,'#ff0000');
forms["newForm"].controller.show();

var status = solutionModel.removeMedia('imagel.jpg');
if (status) application.output("imagel.png has been removed");
else application.output("imagel.png has not been removed");

var mediaList = solutionModel.getMediaList();
for (var i = 0; i < mediaList.length; i++) {
    application.output(mediaList[i].getName() + ":" + mediaList[i].mimeType);
}

```

**removeRelation(name)**

Removes the relation specified by name. You cannot remove the relation if it is touched within the application. So even if you remove all the ui elements using it, like tabs, it still can't be removed, because of underlying created and cached data.

**Parameters**

[String](#) name the name of the relation to be removed

**Returns**

[Boolean](#) true if the removal was successful, false otherwise

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```

var success = solutionModel.removeRelation('myRelation');
if (success) { application.output("Relation has been removed");}
else {application.output("Relation could not be removed");}

```

**removeStyle(name)**

Removes the specified style.

**Parameters**

[String](#) name the name of the style to be removed

**Returns**

[Boolean](#) true if the removal was successful, false otherwise

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```

var s = solutionModel.newStyle("smStyle1",'form { background-color: yellow; }');
var status = solutionModel.removeStyle("smStyle1");
if (status == false) application.output("Could not remove style.");
else application.output("Style removed.");

```

**removeValueList(name)**

Removes the specified valuelist.

**Parameters**

[String](#) name name of the valuelist to be removed

**Returns**

[Boolean](#) true if the removal was successful, false otherwise

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
var vlName = "customValueList";
var vl = solutionModel.newValueList(vlName,JSValueList.CUSTOM_VALUES);
vl.customValues = "customvalue1\ncustomvalue2";

var status = solutionModel.removeValueList(vlName);
if (status) application.output("Removal has been done.");
else application.output("ValueList not removed.");

var vls = solutionModel.getValueLists();
if (vls != null) {
    for (var i = 0; i < vls.length; i++) {
        application.output(vls[i]);
    }
    application.output("");
}
```

**revertForm(name)**

Reverts the specified form to the original (blueprint) version of the form; will result in an exception error if the form is not an original form.

NOTE: Make sure you call `history.remove` first in your Servoy method (script) or call `form.controller.recreateUI()` before the script ends.

**Parameters**

[String](#) name the specified name of the form to revert

**Returns**

[JSForm](#) a JSForm object

**Supported Clients**

SmartClient,WebClient,NGClient,MobileClient

**Sample**

```
// revert the form to the original solution form, removing any changes done to it through the solution model.
var revertedForm = solutionModel.revertForm('myForm')
// add a label on a random place.
revertedForm.newLabel("MyLabel",Math.random()*100,Math.random()*100,80,20);
// make sure that the ui is up to date.
forms.myForm.controller.recreateUI();
```

**wrapMethodWithArguments(method, args)**

Get a `JSMETHOD` instance with arguments to be assigned to an event.

**Parameters**

[JSMETHOD](#) method JSMETHOD to be assigned to an event

[Array](#) args positional arguments

**Returns**

[JSMETHOD](#) a JSMETHOD

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var str = "John's Bookstore"
var form = solutionModel.getForm('orders')
var button = form.getButton('abutton')
var method = form.getFormMethod('doit') // has 4 arguments: event (fixed), boolean, number and string
// string arguments have to be quoted, they are interpreted before the method is called
var quotedString = ""+utils.stringReplace(str, "'", "\\')+""
// list all arguments the method has, use nulls for fixed arguments (like event)
button.onAction = solutionModel.wrapMethodWithArguments(method, [null, true, 42, quotedString])
```