SolutionModel

Return Types

ALIGNMENT ANCHOR BEVELTYPE CURSOR DEFAULTS FONTSTYLE JSBean JSButton JSCalculation JSComponent JSField JSForm JSLabel JSMedia JSMethod JSPart JSPortal JSRelation JSRelationItem JSStyle JSTab JSTabPanel JSValueList JSVariable MEDIAOPTION PAGEORIENTATION PRINTSLIDING SCROLLBAR TITLEJUSTIFICATION TITLEPOSITION UNITS

Method 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, [orientation], [units]) Create a page format 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.
JSCalculation	#getCalculation(name, datasource) Get an existing calculation for the given name and datasource.
JSCalculation[]	#getCalculations(datasource) Gets all the calculations for the given datasource.
JSForm	#getForm(name) Gets the specified form object and returns information about the form (see JSForm node).
JSForm[]	#getForms() Get an array of all forms.
JSForm[]	#getForms(datasource) Get an array of forms, that are all based on datasource/servername.
JSForm[]	#getForms(server, tablename) Get an array of forms, that are all based on datasource/servername and tablename.
JSMethod	#getGlobalMethod(name) Gets an existing global method by the specified name.
JSMethod[]	#getGlobalMethods() The list of all global methods.
JSVariable	#getGlobalVariable(name) Gets an existing global variable by the specified name.
JSVariable[]	#getGlobalVariables() Gets an array of all global variables.
JSMedia	#getMedia(name) Gets the specified media object; can be assigned to a button/label.
JSMedia[]	#getMediaList() Gets the list of all media objects.
JSRelation	#getRelation(name) Gets an existing relation by the specified name and returns a JSRelation Object.
JSRelation[]	#getRelations([primary_server_name/primary_data_source], [primary_table_name]) Gets an array of all relations; or an array of all global relations if the specified table is NULL.
JSStyle	#getStyle(name) Gets the style specified by the given name.
JSValueList	#getValueList(name) Gets an existing valuelist by the specified name and returns a JSValueList Object that can be assigned to a field.

JSValueList	#getValueLists() Gets an array of all valuelists for the currently active solution.
[]	
JSCalculation	#newCalculation(code, type, datasource) Creates a new calculation for the given code and the type, if it builds on a column (name is a column name) then type will be ignored.
JSCalculation	#newCalculation(code, datasource) Creates a new calculation for the given code, the type will be the column where it could be build on (if name is a column name), else it will default to JSVariable.
JSForm	#newForm(name, superForm) Creates a new form with the given JSForm as its super form.
JSForm	#newForm(name, dataSource, styleName, show_in_menu, width, height) Creates a new JSForm Object.
JSForm	#newForm(name, serverName, tableName, styleName, show_in_menu, width, height) Creates a new JSForm Object.
JSMethod	#newGlobalMethod(code) Creates a new global method with the specified code.
JSVariable	#newGlobalVariable(name, type) Creates a new global variable with the specified name and number type.
JSMedia	#newMedia(name, bytes) Creates a new media object that can be assigned to a label or a button.
JSRelation	#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.
JSRelation	#newRelation(name, primaryServerName, primaryTableName, foreignServerName, foreignTableName, joinType) Creates a new JSRelation Object with a specified name; includes the primary server and table name, foreign server and table name, and the type of join for the new relation.
JSStyle	#newStyle(name, content) Creates a new style with the given css content string under the given name.
JSValueList	#newValueList(name, type) Creates a new valuelist with the specified name and number type.
Boolean	#removeCalculation(name, datasource) Removes the calculation specified by name and datasource.
Boolean	#removeForm(name) Removes the specified form during the persistent connected client session.
Boolean	#removeGlobalMethod(name) Removes the specified global method.
Boolean	#removeGlobalVariable(name) Removes the specified global variable.
Boolean	#removeMedia(name) Removes the media item specified by name.
Boolean	#removeRelation(name) Removes the relation specified by name.
Boolean	#removeStyle(name) Removes the specified style.
Boolean	#removeValueList(name) Removes the specified valuelist.
JSForm	#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.
JSMethod	#wrapMethodWithArguments(method, args) Get a JSMethod instance with arguments to be assigned to an event.

Method Details

cloneComponent

JSComponent **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

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

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.

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

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

JSForm **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

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

String **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

Sample

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

createEmptyBorder

String **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

Sample

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

createEtchedBorder

String **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

Sample

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

createFont

String **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

Sample

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

createLineBorder

String **createLineBorder**(thick, color)

Create a line border string.

Parameters

{Number} thick – border thickness in pixels
{String} color – color of the line border

Returns

String

Sample

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

createMatteBorder

String **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

Sample

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

createPageFormat

String **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

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

Returns

String

Sample

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

createSpecialMatteBorder

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.

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
{Number[]} dash_pattern – the dash pattern of border stroke

Returns

String

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

String **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

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);
```

getCalculation

JSCalculation **getCalculation**(name, datasource)

Get an existing calculation for the given name and datasource.

Parameters

{String} name – The name of the calculation

{String} datasource – The datasource the calculation belongs to.

Returns

JSCalculation

Sample

```
var calc = solutionModel.newCalculation("function myCalculation() { return 123; }", JSVariable.INTEGER, "db:
/example_data/customers");
var calc2 = solutionModel.newCalculation("function myCalculation2() { return '20'; }", "db:/example_data
/customers");
var calc3 = solutionModel.newCalculation("function myCalculation3() { return 'Hello World!'; }",
JSVariable.TEXT, "db:/example_data/employees");

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

var allCalcs = solutionModel.getCalculations("db:/example_data/customers");
for (var i = 0; i < allCalcs.length; i++) {
        application.output(allCalcs[i]);
}
```

getCalculations

JSCalculation[] **getCalculations**(datasource)

Gets all the calculations for the given datasource.

Parameters

{String} datasource – The datasource the calculations belong to.

Returns

JSCalculation[]

Sample

```
var calc = solutionModel.newCalculation("function myCalculation() { return 123; }", JSVariable.INTEGER, "db:
/example_data/customers");
var calc2 = solutionModel.newCalculation("function myCalculation2() { return '20'; }", "db:/example_data
/customers");
var calc3 = solutionModel.newCalculation("function myCalculation3() { return 'Hello World!'; }",
JSVariable.TEXT, "db:/example_data/employees");

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

var allCalcs = solutionModel.getCalculations("db:/example_data/customers");
for (var i = 0; i < allCalcs.length; i++) {
        application.output(allCalcs[i]);
}
```

getForm

JSForm **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

Sample

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

getForms

JSForm[] **getForms**()

Get an array of all forms.

Returns

JSForm[] – an array of JSForm type elements

Sample

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

getForms

JSForm[] **getForms**(datasource)

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

Parameters

{String} datasource – the datasource or servername

Returns

JSForm[] – an array of JSForm type elements

Sample

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

getForms

JSForm[] **getForms**(server, tablename)

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

Parameters

{String} server – the datasource or servername

{String} tablename – the tablename

Returns

JSForm[] – an array of JSForm type elements

Sample

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

getGlobalMethod

JSMethod **getGlobalMethod**(name)

Gets an existing global method by the specified name.

Parameters

{String} name – the name of the specified global method

Returns

JSMethod – a JSMethod

Sample

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

getGlobalMethods

JSMethod[] **getGlobalMethods**()

The list of all global methods.

Returns

JSMethod[] – an array of JSMethod type elements

Sample

```
var methods = solutionModel.getGlobalMethods();
	if (methods != null)
		for (var x in methods)
			application.output(methods[x].getName());
```

getGlobalVariable

JSVariable **getGlobalVariable**(name)

Gets an existing global variable by the specified name.

Parameters

{String} name – the specified name of the global variable

Returns

JSVariable – a JSVariable

Sample

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

getGlobalVariables

JSVariable[] **getGlobalVariables**()

Gets an array of all global variables.

Returns

JSVariable[] – an array of JSVariable type elements

Sample

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

getMedia

JSMedia **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

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

JSMedia[] **getMediaList**()

Gets the list of all media objects.

Returns

JSMedia[] – a list with all the media objects.

Sample

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

getRelation

JSRelation **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
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
JSRelation[] **getRelations**([primary_server_name/primary_data_source], [primary_table_name])
Gets an array of all relations; or an array of all global relations if the specified table is NULL.
Parameters
[primary_server_name/primary_data_source] – the specified name of the server or datasource for the specified table
[primary_table_name] – the specified name of the table
Returns
JSRelation[] – an array of all relations (all elements in the array are of type JSRelation)
Sample

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

getStyle
JSStyle **getStyle**(name)
Gets the style specified by the given name.
Parameters
{String} name – the specified name of the style
Returns
JSStyle – a JSStyle
Sample

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

getValueList
JSValueList **getValueList**(name)
Gets an existing valuelist by the specified name and returns a JSValueList Object that can be assigned to a field.
Parameters
{String} name – the specified name of the valuelist
Returns
JSValueList – a JSValueList object
Sample

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

getValueLists
JSValueList[] **getValueLists**()
Gets an array of all valuelists for the currently active solution.
Returns
JSValueList[] – an array of JSValueList objects

Sample

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

newCalculation

JSCalculation **newCalculation**(code, type, datasource)

Creates a new calculation for the given code and the type, if it builds on a column (name is a column name) then type will be ignored.

Parameters

{String} code – The code of the calculation, this must be a full function declaration.
{Number} type – The type of the calculation, one of the JSVariable types.
{String} datasource – The datasource this calculation belongs to.

Returns

JSCalculation

Sample

```
var calc = solutionModel.newCalculation("function myCalculation() { return 123; }", JSVariable.INTEGER, "db:
/example_data/customers");
var calc2 = solutionModel.newCalculation("function myCalculation2() { return '20'; }", "db:/example_data
/customers");
var calc3 = solutionModel.newCalculation("function myCalculation3() { return 'Hello World!'; }",
JSVariable.TEXT, "db:/example_data/employees");

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

var allCalcs = solutionModel.getCalculations("db:/example_data/customers");
for (var i = 0; i < allCalcs.length; i++) {
	application.output(allCalcs[i]);
}
```

newCalculation

JSCalculation **newCalculation**(code, datasource)

Creates a new calculation for the given code, the type will be the column where it could be build on (if name is a column name),
else it will default to JSVariable.TEXT;

Parameters

{String} code – The code of the calculation, this must be a full function declaration.
{String} datasource – The datasource this calculation belongs to.

Returns

JSCalculation

Sample

```
var calc = solutionModel.newCalculation("function myCalculation() { return 123; }", JSVariable.INTEGER, "db:
/example_data/customers");
var calc2 = solutionModel.newCalculation("function myCalculation2() { return '20'; }", "db:/example_data
/customers");
var calc3 = solutionModel.newCalculation("function myCalculation3() { return 'Hello World!'; }",
JSVariable.TEXT, "db:/example_data/employees");

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

var allCalcs = solutionModel.getCalculations("db:/example_data/customers");
for (var i = 0; i < allCalcs.length; i++) {
	application.output(allCalcs[i]);
}
```

newForm

JSForm **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
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', 'myServerName', 'myTableName', 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

JSForm **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
Sample

```
var myForm = solutionModel.newForm('newForm', 'myServer', 'myTable', '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

JSForm **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
Sample

```
var myForm = solutionModel.newForm('newForm', 'myServer', 'myTable', '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

JSMethod **newGlobalMethod**(code)

Creates a new global method with the specified code.

Parameters

{String} code – the specified code for the global method

Returns

JSMethod – a JSMethod object

Sample

```
var method = solutionModel.newGlobalMethod('function myglobalmethod(){currentcontroller.newRecord()}')
```

newGlobalVariable

JSVariable **newGlobalVariable**(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} name – the specified name for the global variable

{Number} type – the specified number type for the global variable

Returns

JSVariable – a JSVariable object

Sample

```
var myGlobalVariable = solutionModel.newGlobalVariable('newGlobalVariable',JSVariable.INTEGER);
        myGlobalVariable.defaultValue = 12;
```

newMedia

JSMedia **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

{byte[]} bytes – The content

Returns

JSMedia – a JSMedia object

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
```

newRelation

JSRelation **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

Sample

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

newRelation

JSRelation **newRelation**(name, primaryServerName, primaryTableName, foreignServerName, foreignTableName, joinType)

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

Parameters
{String} name – the specified name of the new relation
{String} primaryServerName – the specified name of the primary server
{String} primaryTableName – the specified name of the primary table
{String} foreignServerName – the specified name of the foreign server
{String} foreignTableName – the specified name of the foreign table
{Number} joinType – the type of join for the new relation; JSRelation.INNER_JOIN, JSRelation.LEFT_OUTER_JOIN
Returns
JSRelation – a JSRelation object
Sample

```
var rel = solutionModel.newRelation
('myRelation','myPrimaryServerName','myPrimaryTableName','myForeignServerName','myForeignTableName',JSRelation.
INNER_JOIN);
application.output(rel.getRelationItems());
```

newStyle

JSStyle **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
Sample

```
var form = solutionModel.newForm('myForm','myServer','myTable',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

JSValueList **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
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("globals.text",10,10,120,20);
combo1.valuelist = vl1;
var combo2 = form.newComboBox("globals.id",10,60,120,20);
combo2.valuelist = vl2;
```

removeCalculation

Boolean **removeCalculation**(name, datasource)

Removes the calculation specified by name and datasource.
Parameters
{String} name – the name of the calculation to be removed
{String} datasource – the datasource the calculation belongs to

Returns
Boolean – true if the removal was successful, false otherwise
Sample

```
var calc1 = solutionModel.newCalculation("function myCalculation1() { return 123; }", JSVariable.INTEGER, "db:
/example_data/customers");
var calc2 = solutionModel.newCalculation("function myCalculation2() { return '20'; }", "db:/example_data
/customers");

var c = solutionModel.getCalculation("myCalculation1", "db:/example_data/customers");
application.output("Name: " + c.getName() + ", Stored: " + c.isStored());

solutionModel.removeCalculation("myCalculation1", "db:/example_data/customers");
c = solutionModel.getCalculation("myCalculation1", "db:/example_data/customers");
if (c != null) {
                application.output("myCalculation could not be removed.");
}

var allCalcs = solutionModel.getCalculations("db:/example_data/customers");
for (var i = 0; i < allCalcs.length; i++) {
                application.output(allCalcs[i]);
}
```

removeForm

Boolean **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
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

Boolean **removeGlobalMethod**(name)
Removes the specified global method.
Parameters
{String} name – the name of the global method to be removed
Returns
Boolean – true if the removal was successful, false otherwise
Sample

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

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

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

removeGlobalVariable

Boolean **removeGlobalVariable**(name)
Removes the specified global variable.

Parameters
{String} name – the name of the global variable to be removed
Returns
Boolean – true if the removal was successful, false otherwise
Sample

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

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

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

removeMedia

Boolean **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
Sample

```
var bytes1 = plugins.file.readFile('D:/Imgs/image1.png');
var image1 = solutionModel.newMedia('image1.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",currentcontroller.getDataSource(),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('image1.jpg');
if (status) application.output("image1.png has been removed");
else application.output("image1.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

Boolean **removeRelation**(name)
Removes the relation specified by name.
Parameters
{String} name – the name of the relation to be removed
Returns
Boolean – true if the removal was successful, false otherwise
Sample

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

removeStyle

Boolean **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
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

Boolean **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
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

JSForm **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
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

JSMethod **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
{Object[]} args – positional arguments
Returns
JSMethod – a JSMethod

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)
```