JSForm

Property Summary

String	dataSource The names of the database server and table that this form is linked to.
String	name The name of the form.
Object	navigator The navigator is a form that usually handles navigation in application.
JSMethod	onHide The method that is triggered when another form is being activated.
JSMethod	onLoad The method that is triggered when a form is loaded/reloaded from the repository; used to alter elements, set globals, hide toolbars, etc; onShow method can also be assigned.
JSMethod	onRecordSelection The method that is triggered each time a record is selected.
JSMethod	onShow The method that is triggered EVERY TIME the form is displayed; an argument must be passed to the method if this is the first time the form is displayed.
String	serverName Get the server name used by this form.
String	tableName The [name of the table/SQL view].

Method Summary

JSBean	getBean(name) Returns a JSBean that has the given name.
JSButton	getButton(name) Returns a JSButton that has the given name.
JSButton[]	getButtons() Returns all JSButtons of this form, including the ones without a name.
JSComponent	getComponent(name) Returns a JSComponent that has the given name; if found it will be a JSField, JSLabel, JSButton, JSPortal, JSBean or JSTabPanel.
JSComponent[]	getComponents() Returns a array of all the JSComponents that a form has; they are of type JSField,JSLabel,JSButton,JSPortal,JSBean or JSTabPanel.
JSField	getField(name) The field with the specified name.
JSField[]	getFields() Returns all JSField objects of this form, including the ones without a name.
JSFooter	getFooter() Get the Footer part on the form if it exists.
JSHeader	getHeader() Get the Header part on the form if it exists.
JSInsetList	getInsetList(name) Returns an existing inset list.
JSInsetList[]	getInsetLists() Gets all insets lists on the form.
JSLabel	getLabel(name) Returns a JSLabel that has the given name.
JSLabel[]	getLabels() Returns all JSLabels of this form (not including its super form), including the ones without a name.
JSMethod	getMethod(name) Gets an existing form method for the given name.
JSMethod[]	getMethods() Returns all existing form methods for this form.
JSVariable	getVariable(name) Gets an existing form variable for the given name.
JSVariable[]	getVariables() An array consisting of all form variables for this form.
JSBean	newBean(name, y) Creates a new JSBean object on the form.
JSButton	newButton(txt, y, jsmethod) Creates a new button on the form with the given text.

JSCalendar	newCalendar(dataprovider, y) Creates a new JSCalendar field on the form.
JSCalendar	newCalendar(dataprovider, y) Creates a new JSCalendar field on the form.
JSChecks	newCheck(dataprovider, y) Creates a new JSChecks field on the form.
JSChecks	newCheck(dataprovider, y) Creates a new JSChecks field on the form.
JSCombobox	newCombobox(dataprovider, y) Creates a new JSCombobox field on the form.
JSCombobox	newCombobox(dataprovider, y) Creates a new JSCombobox field on the form.
JSField	newField(dataprovider, type, y) Creates a new JSField object on the form .
JSField	newField(dataprovider, type, y) Creates a new JSField object on the form .
JSFooter	newFooter() Creates a new Footer part on the form.
JSHeader	newHeader() Creates a new Header part on the form.
JSInsetList	newInsetList(yLocation, relationName, headerText, textDataProviderID) Creates a new inset list mobile component in the given form.
JSLabel	newLabel(txt, y) Creates a new JSLabel object on the form.
JSMethod	newMethod(code) Creates a new form JSMethod - based on the specified code.
JSPassword	newPassword(dataprovider, y) Creates a new JSPassword field on the form.
JSPassword	newPassword(dataprovider, y) Creates a new JSPassword field on the form.
JSRadios	newRadios(dataprovider, y) Creates a new JSRadios field on the form.
JSRadios	newRadios(dataprovider, y) Creates a new JSRadios field on the form.
JSTextArea	newTextArea(dataprovider, y) Creates a new JSTextArea field on the form.
JSTextArea	newTextArea(dataprovider, y) Creates a new JSTextArea field on the form.
JSText	newTextField(dataprovider, y) Creates a new JSText field on the form.
JSText	newTextField(dataprovider, y) Creates a new JSText field on the form.
JSVariable	newVariable(name, type) Creates a new form JSVariable - based on the name of the variable object and the number type, uses the SolutionModel JSVariable constants.
JSVariable	newVariable(name, type, defaultValue) Creates a new form JSVariable - based on the name of the variable object , the type and it's default value , uses the SolutionModel JSVariable constants.
Boolean	removeBean(name) Removes a JSBean that has the specified name.
Boolean	removeButton(name) Removes a JSButton that has the specified name.
Boolean	removeComponent(name) Removes a component (JSLabel, JSButton, JSField, JSPortal, JSBean, JSTabpanel) that has the given name.
Boolean	removeField(name) Removes a JSField that has the given name.
Boolean	removeFooter() Removes a JSFooter if it exists.
Boolean	removeHeader() Removes a JSHeader if it exists.
Boolean	removeInsetList(name) Removes inset list from the form.
Boolean	removeLabel(name) Removes a JSLabel that has the given name.
Boolean	removeMethod(name) Removes a form JSMethod - based on the specified code.
Boolean	removeVariable(name) Removes a form JSVariable - based on the name of the variable object.

Property Details

dataSource

The names of the database server and table that this form is linked to.

Returns

String

Sample

```
var myForm = solutionModel.newForm('newForm', 'db:/a_server/a_table', 'aStyleName', false, 800, 600)
myForm.dataSource = 'db:/anotherServerName/anotherTableName'
```

name

The name of the form.

Returns

String

Sample

```
var form = solutionModel.newForm('myForm',myDatasource,null,true,800,600);
var formName = form.name;
application.output(formName);
```

navigator

The navigator is a form that usually handles navigation in application. It is displayed on left side of the screen. Can also have value SM_DEFAULTS.NONE (no navigator) or SM_DEFAULTS.IGNORE (reuse current form navigator).

Returns

Object

Sample

```
var aForm = solutionModel.newForm('newForm1', myDatasource);
// you can also use SM_DEFAULTS.INGORE to just reuse the navigator that is already set.
// here we assign an other new form as the navigator.
var aNavigator = solutionModel.newForm('navForm', myDatasource);
aForm.navigator = aNavigator;
```

onHide

The method that is triggered when another form is being activated.
NOTE: If the onHide method returns false, the form can be prevented from hiding.
For example, when using onHide with showFormInDialog, the form will not close by clicking the dialog close box (X).

Returns

JSMethod

Sample

```
form.onShow = form.newMethod('function onShow(firstShow, event) { application.output("onShow intercepted on "
+ event.getFormName() + ". first show? " + firstShow); return false; }');
form.onHide = form.newMethod('function onHide(event) { application.output("onHide blocked on " + event.
getFormName()); return false; }');
```

onLoad

The method that is triggered when a form is loaded/reloaded from the repository; used to alter elements, set globals, hide toolbars,
etc; onShow method can also be assigned.
NOTE: onShow should be used to access current foundset dataproviders; onLoad cannot be used because the foundset data is not loaded until after the form is loaded.
Also calls to loadRecords() should be done in the onShow method and not in the onLoad method
If you call loadRecords() in the onShow method, you may want to set the namedFoundSet property of the form to 'empty' to prevent the first default form query.
NOTE: the onLoad event bubbles down, meaning that the onLoad is first fired on the parent then on a tab in a tabpanel (and in tab of that tab panels if you are 3 deep)

Returns

JSMethod

Sample

```
form.onLoad = form.newMethod('function onLoad(event) { application.output("onLoad intercepted on " + event.
getFormName()); }');
form.onUnLoad = form.newMethod('function onUnLoad(event) { application.output("onUnLoad intercepted on " +
event.getFormName()); }');
```

onRecordSelection

The method that is triggered each time a record is selected.
If a form is in List view or Special table view - when the user clicks on it.
In Record view - after the user navigates to another record using the slider or clicks up or down for next/previous record.
NOTE: Data and Servoy tag values are returned when the onRecordSelection method is executed.

Returns

JSMethod

Sample

```
form.onRecordEditStart = form.newMethod('function onRecordEditStart(event) { application.output
("onRecordEditStart intercepted on " + event.getFormName()); }');
form.onRecordEditStop = form.newMethod('function onRecordEditStop(record, event) { application.output
("onRecordEditStop intercepted on " + event.getFormName() + ". record is: " + record); }');
form.onRecordSelection = form.newMethod('function onRecordSelection(event) { application.output
("onRecordSelection intercepted on " + event.getFormName()); }');
```

onShow

The method that is triggered EVERY TIME the form is displayed; an argument must be passed to the method if this is the first time the form is displayed.

NOTE: onShow can be used to access current foundset dataproviders; onLoad cannot be used because the foundset data is not loaded until after the form is loaded.

NOTE: the onShow event bubbles down, meaning that the onShow event of a form displayed in a tabPanel is fired after the onShow event of the parent.

Returns

JSMethod

Sample

```
form.onShow = form.newMethod('function onShow(firstShow, event) { application.output("onShow intercepted on "
+ event.getFormName() + ". first show? " + firstShow); return false; }');
form.onHide = form.newMethod('function onHide(event) { application.output("onHide blocked on " + event.
getFormName()); return false; }');
```

serverName

Get the server name used by this form.

Returns

String

Sample

```
var form = solutionModel.newForm('myForm',myDatasource,null,true,800,600);
form.serverName = 'anotherServerName';
var theServerName = form.serverName;
application.output(theServerName);
```

tableName

The [name of the table/SQL view].[the name of the database server connection] the form is based on.

Returns

String

Sample

```
var aForm = solutionModel.newForm('newForm1', myDatasource, null, true, 800, 600);
aForm.tableName = 'anotherTableOfMine'
if (forms['newForm1'].controller.find())
{
        columnTextDataProvider = '=aSearchedValue'
        columnNumberDataProvider = '>10';
        forms['newForm1'].controller.search()
}
```

Method Details

getBean

JSBean **getBean** (name)
Returns a JSBean that has the given name.
Parameters

{String} name - the specified name of the bean
Returns

JSBean - a JSBean object
Sample

```
var btn = myForm.getBean("mybean");
application.output(mybean.className);
```

getButton

JSButton **getButton** (name)
Returns a JSButton that has the given name.
Parameters

{String} name - the specified name of the button
Returns

JSButton - a JSButton object
Sample

```
var btn = myForm.getButton("hello");
application.output(btn.text);
```

getButtons

JSButton[] **getButtons** ()
Returns all JSButtons of this form, including the ones without a name.
Returns

JSButton[] - the list of all JSbuttons on this forms
Sample

```
var buttons = myForm.getButtons();
for (var b in buttons)
{
        if (buttons[b].name != null)
                application.output(buttons[b].name);
        else
                application.output(buttons[b].text + " has no name ");
}
```

getComponent

JSComponent **getComponent** (name)

Returns a JSComponent that has the given name; if found it will be a JSField, JSLabel, JSButton, JSPortal, JSBean or JSTabPanel.

Parameters

{String} name - the specified name of the component

Returns

JSComponent - a JSComponent object (might be a JSField, JSLabel, JSButton, JSPortal, JSBean or JSTabPanel)

Sample

```
var frm = solutionModel.getForm("myForm");
var cmp = frm.getComponent("componentName");
application.output("Component type and name: " + cmp);
```

getComponents

JSComponent[] **getComponents** ()

Returns a array of all the JSComponents that a form has; they are of type JSField,JSLabel,JSButton,JSPortal,JSBean or JSTabPanel.

Returns

JSComponent[] - an array of all the JSComponents on the form.

Sample

```
var form = solutionModel.getForm("myForm");
var components = form.getComponents();
for (var i in components)
	application.output("Component type and name: " + components[i]);
```

getField

JSField **getField** (name)

The field with the specified name.

Parameters

{String} name - the specified name of the field

Returns

JSField - a JSField object

Sample

```
var form = solutionModel.getForm("myForm");
var field = form.getField("myField");
application.output(field.dataProviderID);
```

getFields

JSField[] **getFields** ()

Returns all JSField objects of this form, including the ones without a name.

Returns

JSField[] - all JSField objects of this form

Sample

```
var frm = solutionModel.getForm("myForm");
var fields = frm.getFields();
for (var f in fields)
{
	var fname = fields[f].name;
	if (fname != null)
		application.output(fname);
}
```

getFooter

JSFooter **getFooter** ()
Get the Footer part on the form if it exists.
Returns

JSFooter - A JSFooter or null when not found.
Sample

```
var footer = form.getFooter();
```

getHeader

JSHeader **getHeader** ()
Get the Header part on the form if it exists.
Returns

JSHeader - A JSHeader or null when not found.
Sample

```
var header = form.getHeader();
```

getInsetList

JSInsetList **getInsetList** (name)
Returns an existing inset list.
Parameters

{String} name - the inset list's name.
Returns

JSInsetList - the existing inset list, or null if it does not exist.
Sample

```
var form = solutionModel.getForm("myform");
var insetList = form.getInsetList('mylist1');
```

getInsetLists

JSInsetList[] **getInsetLists** ()
Gets all insets lists on the form.
Returns

JSInsetList[]
Sample

```
var form = solutionModel.getForm('test');
var insetLists = form.getInsetLists();
```

getLabel

JSLabel **getLabel** (name)
Returns a JSLabel that has the given name.
Parameters

{String} name - the specified name of the label
Returns

JSLabel - a JSLabel object (or null if the label with the specified name does not exist)

Sample

```
var frm = solutionModel.getForm("myForm");
var label = frm.getLabel("myLabel");
application.output(label.text);
```

getLabels

JSLabel[] **getLabels** ()
Returns all JSLabels of this form (not including its super form), including the ones without a name.
Returns

JSLabel[] - all JSLabels on this form
Sample

```
var frm = solutionModel.getForm("myForm");
var labels = frm.getLabels();
for (var i in labels)
{
	var lname = labels[i].name;
	if (lname != null)
		application.output(lname);
}
```

getMethod

JSMethod **getMethod** (name)
Gets an existing form method for the given name.
Parameters

{String} name - the specified name of the method
Returns

JSMethod - a JSMethod object (or null if the method with the specified name does not exist)
Sample

```
var frm = solutionModel.getForm("myForm");
var method = frm.getMethod("myMethod");
application.output(method.code);
```

getMethods

JSMethod[] **getMethods** ()
Returns all existing form methods for this form.
Returns

JSMethod[] - all form methods for the form
Sample

```
var frm = solutionModel.getForm("myForm");
var methods = frm.getMethods();
for (var m in methods)
	application.output(methods[m].getName());
```

getVariable

JSVariable **getVariable** (name)
Gets an existing form variable for the given name.
Parameters

{String} name - the specified name of the variable
Returns

JSVariable - a JSVariable object

Sample

```
var frm = solutionModel.getForm("myForm");
        var fvariable = frm.getVariable("myVarName");
        application.output(fvariable.name + " has the default value of " + fvariable.defaultValue);
```

getVariables

JSVariable[] **getVariables** ()
An array consisting of all form variables for this form.
Returns

JSVariable[] - an array of all variables on this form
Sample

```
var frm = solutionModel.getForm("myForm");
var variables = frm.getVariables();
for (var i in variables)
        application.output(variables[i].name);
```

newBean

JSBean **newBean** (name, y)
Creates a new JSBean object on the form.
Parameters

{String} name - the specified name of the JSBean object
{Number} y - the vertical "y" position of the JSBean object, defines the order of elements on the form
Returns

JSBean - a JSBean object
Sample

```
var form = solutionModel.newForm('newForm1', 'db:/server1/table1');
var bean = form.newBean('bean', 1);
forms['newForm1'].controller.show();
```

newButton

JSButton **newButton** (txt, y, jsmethod)
Creates a new button on the form with the given text.
Parameters

{String} txt - the text on the button
{Number} y - the y coordinate of the button location on the form, defines the order of elements on the form
{JSMethod} jsmethod - the method assigned to handle an onAction event
Returns

JSButton - a new JSButton object
Sample

```
var form = solutionModel.newForm('newForm1', myDatasource);
var method = form.newMethod('function onAction(event) { application.output("onAction intercepted on " + event.
getFormName()); }');
var button = form.newButton('myButton', 1, method);
application.output("The new button: " + button.name + " has the following onAction event handling method
assigned " + button.onAction.getName());
```

newCalendar

JSCalendar **newCalendar** (dataprovider, y)
Creates a new JSCalendar field on the form.

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSCalendar - a new JSCalendar field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal', JSVariable.DATETIME);
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar', JSVariable.DATETIME);
var field = form.newCalendar(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newCalendar

JSCalendar **newCalendar** (dataprovider, y)

Creates a new JSCalendar field on the form.

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSCalendar - a new JSCalendar field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal', JSVariable.DATETIME);
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar', JSVariable.DATETIME);
var field = form.newCalendar(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newCheck

JSChecks **newCheck** (dataprovider, y)

Creates a new JSChecks field on the form.

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSChecks - a new JSChecks field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.INTEGER);
//x.defaultValue = "'1'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.INTEGER);
//x.defaultValue = "'1'";
var field = form.newCheck(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newCheck

JSChecks **newCheck** (dataprovider, y)

Creates a new JSChecks field on the form.

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSChecks - a new JSChecks field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.INTEGER);
//x.defaultValue = "'1'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.INTEGER);
//x.defaultValue = "'1'";
var field = form.newCheck(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newCombobox

JSCombobox **newCombobox** (dataprovider, y)

Creates a new JSCombobox field on the form.

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSCombobox - a new JSCombobox field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var field = form.newCombobox(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newCombobox

JSCombobox **newCombobox** (dataprovider, y)

Creates a new JSCombobox field on the form.

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSCombobox - a new JSCombobox field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var field = form.newCombobox(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newField

JSField **newField** (dataprovider, type, y)

Creates a new JSField object on the form .

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} type - the display type of the JSField object (see the Solution Model -> JSField node for display types)
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSField

Sample

```
var form = solutionModel.newForm('newForm1', myDatasource, null, true, 800, 600);
var variable = form.newVariable('myVar', JSVariable.TEXT);
variable.defaultValue = "'This is a default value (with triple quotes)!'";
var field = form.newField(variable, JSField.TEXT_FIELD, 1);
forms['newForm1'].controller.show();
```

newField

JSField **newField** (dataprovider, type, y)

Creates a new JSField object on the form .

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} type - the display type of the JSField object (see the Solution Model -> JSField node for display types)
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSField

Sample

```
var form = solutionModel.newForm('newForm1', myDatasource, null, true, 800, 600);
var variable = form.newVariable('myVar', JSVariable.TEXT);
variable.defaultValue = "'This is a default value (with triple quotes)!'";
var field = form.newField(variable, JSField.TEXT_FIELD, 1);
forms['newForm1'].controller.show();
```

newFooter

JSFooter **newFooter** ()

Creates a new Footer part on the form.

Returns

JSFooter - A JSFooter instance corresponding to the newly created Footer form part.

Sample

```
var footer = form.newFooter();
```

newHeader

JSHeader **newHeader** ()

Creates a new Header part on the form.

Returns

JSHeader - A JSHeader instance corresponding to the newly created Header form part.

Sample

```
var header = form.newHeader();
```

newInsetList

JSInsetList **newInsetList** (yLocation, relationName, headerText, textDataProviderID)

Creates a new inset list mobile component in the given form. The inset list will be populated based on the given datasource and relation.

Parameters

{Number} yLocation - the vertical location of the inset list in the form's components list.
{String} relationName - the relation used to show data, just like it would happen in a related tab-panel.
{String} headerText - can be null; it's a convenience argument for setting the title (header text) for the inset list.
{String} textDataProviderID - can be null; it's a convenience argument for setting the dataprovider that will be used to populate the main text area of the list's items.

Returns

JSInsetList - the newly created inset list.

Sample

```
var f = solutionModel.newForm("created_by_sm_1","db:/udm/contacts");
// create an inset list
var insetList = f.newInsetList(8,"accountmanager_to_companies","Companies","company_name");
insetList.subtextDataProviderID = "company_description";
insetList.onAction = f.newMethod("function buttonPressed() { plugins.dialogs.showWarningDialog('Title', 'inset
list clicked','OK'); }");
```

newLabel

JSLabel **newLabel** (txt, y)

Creates a new JSLabel object on the form.

Parameters

{String} txt - the specified text of the label object
{Number} y - the vertical "y" position of the label, defines the order of elements on the form
Returns

JSLabel - a JSLabel object
Sample

```
var form = solutionModel.newForm('newForm1', myDatasource);
var label = form.newLabel('The text on the label', 1);
forms['newForm1'].controller.show();
```

newMethod

JSMethod **newMethod** (code)
Creates a new form JSMethod - based on the specified code.
Parameters

{String} code - the specified code for the new method
Returns

JSMethod - a new JSMethod object for this form
Sample

```
var form = solutionModel.newForm('newForm1', myDatasource, null, true, 800, 600);
var method = form.newMethod('function aMethod(event){application.output("Hello world!");}');
var button = myListViewForm.newButton('Show message!',50,50,100,30,method);
forms['newForm1'].controller.show();
```

newPassword

JSPassword **newPassword** (dataprovider, y)
Creates a new JSPassword field on the form.
Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form
Returns

JSPassword - a new JSPassword field
Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
var field = form.newPassword(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newPassword

JSPassword **newPassword** (dataprovider, y)
Creates a new JSPassword field on the form.
Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form
Returns

JSPassword - a new JSPassword field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
var field = form.newPassword(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newRadios

JSRadios **newRadios** (dataprovider, y)

Creates a new JSRadios field on the form.

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSRadios - a new JSRadios field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var field = form.newRadios(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newRadios

JSRadios **newRadios** (dataprovider, y)

Creates a new JSRadios field on the form.

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSRadios - a new JSRadios field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var field = form.newRadios(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newTextArea

JSTextArea **newTextArea** (dataprovider, y)
Creates a new JSTextArea field on the form.

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSTextArea - a new JSTextArea field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var field = form.newTextArea(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newTextArea

JSTextArea **newTextArea** (dataprovider, y)
Creates a new JSTextArea field on the form.

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSTextArea - a new JSTextArea field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var field = form.newTextArea(x,1);
//or a column data provider as the dataprovider
//field.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newTextField

JSText **newTextField** (dataprovider, y)

Creates a new JSText field on the form.

Parameters

{JSVariable} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSText - a new JSText field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the Text Field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var textField = form.newTextField(x,1);
//or a column data provider as the dataprovider
//textField.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newTextField

JSText **newTextField** (dataprovider, y)

Creates a new JSText field on the form.

Parameters

{String} dataprovider - the specified dataprovider name/JSVariable of the JSField object
{Number} y - the vertical "y" position of the JSField object, defines the order of elements on the form

Returns

JSText - a new JSText field

Sample

```
var form = solutionModel.newForm('newForm1',myDatasource);
//choose the dataprovider or JSVariable you want for the Text Field
var x = null;
//global JSVariable as the dataprovider
//x = solutionModel.newGlobalVariable('globals', 'myGlobal',JSVariable.TEXT);
//x.defaultValue = "'Text from a global variable'";
//or a form JSVariable as the dataprovider
//x = form.newVariable('myFormVar',JSVariable.TEXT);
//x.defaultValue = "'Text from a form variable'";
var textField = form.newTextField(x,1);
//or a column data provider as the dataprovider
//textField.dataProviderID = columnTextDataProvider;
forms['newForm1'].controller.show();
```

newVariable

JSVariable **newVariable** (name, type)

Creates a new form JSVariable - based on the name of the variable object and the number type, uses the SolutionModel JSVariable constants.

Parameters

{String} name - the specified name of the variable
{Number} type - the specified type of the variable (see Solution Model -> JSVariable node constants)

Returns

JSVariable - a JSVariable object

Sample

```
var form = solutionModel.newForm('newForm1', myDatasource, null, true, 800, 600);
var variable = form.newVariable('myVar', JSVariable.TEXT , "'This is a default value (with triple quotes)!'");
//or variable = form.newVariable('myVar', JSVariable.TEXT)
//variable.defaultValue = "'This is a default value (with triple quotes)!'" // setting the default value after
the variable is created requires form recreation
//variable.defaultValue = "{a:'First letter',b:'Second letter'}"
var field = form.newField(variable, JSField.TEXT_FIELD, 100, 100, 200, 200);
forms['newForm1'].controller.show();
```

newVariable

JSVariable **newVariable** (name, type, defaultValue)

Creates a new form JSVariable - based on the name of the variable object , the type and it's default value , uses the SolutionModel JSVariable constants.

This method does not require the form to be destroyed and recreated. Use this method if you want to change the form's model without destroying the runtime form

Parameters

{String} name - the specified name of the variable
{Number} type - the specified type of the variable (see Solution Model -> JSVariable node constants)
{String} defaultValue - the default value as a javascript expression string

Returns

JSVariable - a JSVariable object

Sample

```
var form = solutionModel.newForm('newForm1', myDatasource, null, true, 800, 600);
var variable = form.newVariable('myVar', JSVariable.TEXT , "'This is a default value (with triple quotes)!'");
//or variable = form.newVariable('myVar', JSVariable.TEXT)
//variable.defaultValue = "'This is a default value (with triple quotes)!'" // setting the default value after
the variable is created requires form recreation
//variable.defaultValue = "{a:'First letter',b:'Second letter'}"
var field = form.newField(variable, JSField.TEXT_FIELD, 100, 100, 200, 200);
forms['newForm1'].controller.show();
```

removeBean

Boolean **removeBean** (name)

Removes a JSBean that has the specified name. Returns true if removal was successful, false otherwise.
Parameters

{String} name - the specified name of the JSBean to be removed
Returns

Boolean - true if the JSBean has been removed; false otherwise
Sample

```
var form = solutionModel.getForm('myform');
form.removeBean('mybean')
```

removeButton

Boolean **removeButton** (name)
Removes a JSButton that has the specified name. Returns true if removal was successful, false otherwise.
Parameters

{String} name - the specified name of the JSButton to be removed
Returns

Boolean - true if the JSButton has been removed; false otherwise
Sample

```
var form = solutionModel.newForm('newFormX',myDatasource,null,true,800,600);
var b1 = form.newButton('This is button1',100,100,200,50,null);
b1.name = 'b1';
var jsmethod = form.newMethod("function removeMe(event) { var form = solutionModel.getForm('newFormX'); if
(form.removeButton('b1') == true) application.output('Button has been removed ok'); else application.output
('Button could not be deleted'); forms['newFormX'].controller.recreateUI();}");
var b2 = form.newButton('Click here to remove button1',100,230,200,50,jsmethod);
b2.name = 'b2';
forms['newFormX'].controller.show();
```

removeComponent

Boolean **removeComponent** (name)
Removes a component (JSLabel, JSButton, JSField, JSPortal, JSBean, JSTabpanel) that has the given name. It is the same as calling "if(!removeLabel (name) && !removeButton(name) ....)".
Returns true if removal was successful, false otherwise.
Parameters

{String} name - the specified name of the component to be deleted
Returns

Boolean - true if component has been successfully deleted; false otherwise

Sample

```
var form = solutionModel.newForm('newFormX','db:/server1/parent_table',null,true,1000,750);
var jsbutton = form.newButton('JSButton to delete',100,100,200,50,null);
jsbutton.name = 'jsb';
var jslabel = form.newLabel('JSLabel to delete',100,200,200,50,null);
jslabel.name = 'jsl';
jslabel.transparent = false;
jslabel.background = 'green';
var jsfield = form.newField('scopes.globals.myGlobalVariable',JSField.TEXT_FIELD,100,300,200,50);
jsfield.name = 'jsf';
var relation = solutionModel.newRelation('parentToChild','db:/server1/parent_table','db:/server1/child_table',
JSRelation.INNER_JOIN);
relation.newRelationItem('parent_table_id', '=', 'child_table_id');
var jsportal = form.newPortal('jsp',relation,100,400,300,300);
jsportal.newField('child_table_id',JSField.TEXT_FIELD,200,200,120);
var childOne = solutionModel.newForm('childOne','db:/server1/child_table',null,false,400,300);
childOne.newField('child_table_id', JSField.TEXT_FIELD,10,10,100,20);
var childTwo = solutionModel.newForm('childTwo','server1','other_table',null,false,400,300);
childTwo.newField('some_table_id', JSField.TEXT_FIELD,10,10,100,100);
var jstabpanel = form.newTabPanel('jst',450,30,620,460);
jstabpanel.newTab('tab1','Child One',childOne,relation);
jstabpanel.newTab('tab2','Child Two',childTwo);
var jsmethod = form.newMethod("function removeMe(event) { var form = solutionModel.getForm('newFormX');\n if
((form.removeComponent('jsb') == true) && (form.removeComponent('jsl') == true) && (form.removeComponent
('jsf') == true) && (form.removeComponent('jsp') == true) & (form.removeComponent('jst') == true)) application.
output('Components removed ok'); else application.output('Some component(s) could not be deleted'); forms
['newFormX'].controller.recreateUI();}");
var removerButton = form.newButton('Click here to remove form components',450,500,250,50,jsmethod);
removerButton.name = 'remover';
forms['newFormX'].controller.show();
```

removeField

Boolean **removeField** (name)

Removes a JSField that has the given name. Returns true if removal was successful, false otherwise.

Parameters

{String} name - the specified name of the JSField to remove

Returns

Boolean - true is the JSField has been successfully removed; false otherwise

Sample

```
var form = solutionModel.newForm('newFormX',myDatasource,null,true,800,600);
var jsfield = form.newField(scopes.globals.myGlobalVariable,JSField.TEXT_FIELD,100,300,200,50);
jsfield.name = 'jsf';
var jsmethod = form.newMethod("function removeMe(event) { var form = solutionModel.getForm('newFormX');\n if
(form.removeComponent('jsf') == true) application.output('Field has been removed ok'); else application.output
('Field could not be deleted'); forms['newFormX'].controller.recreateUI();}");
var removerButton = form.newButton('Click here to remove the field',450,500,250,50,jsmethod);
removerButton.name = 'remover';
forms['newFormX'].controller.show();
```

removeFooter

Boolean **removeFooter** ()

Removes a JSFooter if it exists.

Returns

Boolean - true if the JSFooter has successfully been removed; false otherwise

Sample

```
var form = solutionModel.getForm('myform');
form.removeFooter()
```

removeHeader

Boolean **removeHeader** ()
Removes a JSHeader if it exists.
Returns

Boolean - true if the JSHeader has successfully been removed; false otherwise
Sample

```
var form = solutionModel.getForm('myform');
form.removeHeader()
```

removeInsetList

Boolean **removeInsetList** (name)
Removes inset list from the form.
Parameters

{String} name - Inset List name.
Returns

Boolean
Sample

```
var form = solutionModel.getForm('test');
form.removeInsetList('myinsetlist');
```

removeLabel

Boolean **removeLabel** (name)
Removes a JSLabel that has the given name. Returns true if removal successful, false otherwise
Parameters

{String} name - the specified name of the JSLabel to be removed
Returns

Boolean - true if the JSLabel with the given name has successfully been removed; false otherwise
Sample

```
var form = solutionModel.newForm('newFormX',myDatasource,null,true,1000,750);
var jslabel = form.newLabel('JSLabel to delete',100,200,200,50,null);
jslabel.name = 'jsl';
jslabel.transparent = false;
jslabel.background = 'green';
var jsmethod = form.newMethod("function removeMe(event) { var form = solutionModel.getForm('newFormX'); if
(form.removeComponent('jsl') == true) application.output('Label has been removed'); else application.output
('Label could not be deleted'); forms['newFormX'].controller.recreateUI();}");
var removerButton = form.newButton('Click here to remove the green label',450,500,250,50,jsmethod);
removerButton.name = 'remover';
forms['newFormX'].controller.show();
```

removeMethod

Boolean **removeMethod** (name)
Removes a form JSMethod - based on the specified code.
Parameters

{String} name - the specified name of the method
Returns

Boolean - true if method was removed successfully , false otherwise

Sample

```
var form = solutionModel.newForm('newForm1', null, null, true, 800, 600);
var hello = form.newMethod('function aMethod(event){application.output("Hello world!");}');
var removeMethod = form.newMethod('function removeMethod(event){ \
                                                                solutionModel.getForm(event.
getFormName()).removeMethod("aMethod"); \
                                                                forms[event.getFormName()].controller.
recreateUI();\
                                                                }');
var button1 = form.newButton('Call method!',50,50,120,30,hello);
var button2 = form.newButton('Remove Mehtod!',200,50,120,30,removeMethod);
forms['newForm1'].controller.show();
```

removeVariable

Boolean **removeVariable** (name)

Removes a form JSVariable - based on the name of the variable object.

Parameters

{String} name - the specified name of the variable

Returns

Boolean - true if removed, false otherwise (ex: no var with that name)

Sample

```
var form = solutionModel.newForm('newForm1', null, null, true, 800, 600);
var variable = form.newVariable('myVar', JSVariable.TEXT);
variable.defaultValue = "'This is a default value (with triple quotes)!'";
//variable.defaultValue = "{a:'First letter',b:'Second letter'}"
var field = form.newField(variable, JSField.TEXT_FIELD, 100, 100, 200, 200);
forms['newForm1'].controller.show();

variable = form.removeVariable('myVar');
application.sleep(4000);
forms['newForm1'].controller.recreateUI();
```