

# SolutionModel

## Return Types

DEFAULTS JSBean JSButton JSCalendar JSChecks JSCombobox JSComponent JSField JSFooter JSForm JSHeader JSInsetList JSLabel JSList JSMethod JSPassword JSRadios JText JTextArea JSTitle JSValueList JSVariable

## Method Summary

JSForm	<a href="#">getForm(name)</a> Gets the specified form object and returns information about the form (see JSForm node).
JSForm[]	<a href="#">getForms()</a> Get an array of all forms.
JSForm[]	<a href="#">getForms(datasource)</a> Get an array of forms, that are all based on datasource/servername.
JSForm[]	<a href="#">getForms(server, tablename)</a> Get an array of forms, that are all based on datasource/servername and tablename.
JSMethod	<a href="#">getGlobalMethod(scopeName, name)</a> Gets an existing global method by the specified name.
JSMethod[]	<a href="#">getGlobalMethods()</a> The list of all global methods.
JSMethod[]	<a href="#">getGlobalMethods(scopeName)</a> The list of all global methods.
JSVariable	<a href="#">getGlobalVariable(scopeName, name)</a> Gets an existing global variable by the specified name.
JSVariable[]	<a href="#">getGlobalVariables()</a> Gets an array of all global variables.
JSVariable[]	<a href="#">getGlobalVariables(scopeName)</a> Gets an array of all global variables.
JSList	<a href="#">getListForm(formName)</a> Returns an existing list form.
JSList[]	<a href="#">getListForms()</a> Get an array of all list-forms.
String[]	<a href="#">getScopeNames()</a> Gets an array of all scope names used.
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.
JSValueList[]	<a href="#">getValueLists()</a> Gets an array of all valuelists for the currently active solution.
JSForm	<a href="#">newForm(name, dataSource)</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.
JSList	<a href="#">newListForm(formName, dataSource, textDataProviderID)</a> Creates a new list form, similar to an inset list but without the inset list's header and relation.
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="#">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.

## Method Details

### 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** (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

## Sample

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

## getGlobalMethods

JSMethod[] **getGlobalMethods** (scopeName)

The list of all global methods.

## Parameters

{String} scopeName - limit to global methods of specified scope name

## Returns

JSMethod[] - an array of JSMethod type elements

## Sample

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

## getGlobalVariable

JSVariable **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

## Sample

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

## getGlobalVariables

[JSVariable\[\]](#) **getGlobalVariables** (scopeName)

Gets an array of all global variables.

### Parameters

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

### Returns

[JSVariable\[\]](#) - an array of JSVariable type elements

## Sample

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

## getListForm

[JSList](#) **getListForm** (formName)

Returns an existing list form.

### Parameters

formName - the form's name.

### Returns

[JSList](#) - the existing list form, or null if it does not exist.

## Sample

```
var list = solutionModel.getListForm('created_by_sm_2');
```

## getListForms

[JSList\[\]](#) **getListForms** ()

Get an array of all list-forms.

### Returns

[JSList\[\]](#) - an array of IBaseSHList type elements

## Sample

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

## getScopeNames

[String\[\]](#) **getScopeNames** ()

Gets an array of all scope names used.

### Returns

[String\[\]](#) - an array of String scope names

## Sample

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

## 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);
```

## newForm

[JSForm](#) **newForm** (name, dataSource)

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

### Returns

[JSForm](#) - a new JSForm object

### Sample

```
var myForm = solutionModel.newForm('newForm', 'db:/my_server/my_table')
//now you can add stuff to the form (under JSForm node)
//add a label
myForm.newLabel('Name', 1)
//add a "normal" text entry field
myForm.newTextField('dataProviderNameHere', 2)
```

## newGlobalMethod

[JSMethod](#) **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

## Sample

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

## newGlobalVariable

[JSVariable](#) **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

## 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.
```

## newListForm

[JSList](#) **newListForm** (formName, dataSource, textDataProviderID)

Creates a new list form, similar to an inset list but without the inset list's header and relation.

The result will be an independent form which behaves like a mobile list.

### Parameters

[{String}](#) formName - the new form's name.

[{String}](#) dataSource - the list will be populated based on this datasource.

[{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

[JSList](#) - the newly created list form.

## Sample

```
var f = solutionModel.newForm("created_by_sm_1", "udm", "contacts", null, false, 100, 380);
// create a button to go to it on the main form
b = f.newButton("Show created list form", 0, 9, 10, 10,
    f.newMethod("function showListForm() { forms.created_by_sm_2.controller.show(); }"));
// create the actual list form
var list = f.createListForm('created_by_sm_2', databaseManager.getDataSource("udm", "contacts"), "name_first");
list.onAction = solutionModel.getForm('created_by_sm_2').newMethod("function goBack() { history.back(); }");
```

## 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 v11 = solutionModel.newValueList("customText", JSValueList.CUSTOM_VALUES);
v11.customValues = "customvalue1\ncustomvalue2";
var v12 = solutionModel.newValueList("customid", JSValueList.CUSTOM_VALUES);
v12.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 = v11;
var combo2 = form.newComboBox("scopes.globals.id", 10, 60, 120, 20);
combo2.valuelist = v12;
```

## 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** (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

### 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

**Boolean** **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

## 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);
}
```

## 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();
```