SolutionModel

Return Types

ALIGNMENT ANCHOR CURSOR DEFAULTS JSButton JSComponent JSField JSForm JSLabel JSMedia JSMethod JSPart JSPortal JSRelation JSRelationItem JSStyle JSTab JSTabPanel JSValueList JSVariable MEDIAOPTION PRINTSLIDING SCROLLBAR

Method Summery

JSComponent	#cloneComponent(newName, component)
JSComponent	#cloneComponent(newName, component, [newParentForm]) Makes an exact copy of the given component (JSComponent/JSField/JSLabel), gives it a new name and optionally moves it to a new parent form.
JSForm	#cloneForm(newName, jsForm) Makes an exact copy of the given form and gives it the new name.
JSForm	#getForm(name) Gets the specified form object and returns information about the form (see JSForm node).
JSForm[]	#getForms()
JSForm[]	#getForms(datasource)
JSForm[]	#getForms([server], [tablename]) Get an array of forms, that are all based on datasource/servername or 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.
JSForm	#newForm(name, server_name|data_source, [table_name], style, 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, primary_server_name|primary_data_source, [primary_table_name], foreign_server_name|foreign_data_source, [foreign_table_name], join_type) Creates a new JSRelation Object with a specified name; includes the primary datasource, optional table name, foreign datasource, optional foreign 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	#removeForm(name) Removes the specified form during the persistent connected client session.
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.

Method Details

cloneComponent

JSComponent **cloneComponent**(newName, component, [newParentForm])

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

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();
```

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**([server], [tablename])

Get an array of forms, that are all based on datasource/servername or 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)
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);
```

newForm

JSForm **newForm**(name, server_name|data_source, [table_name], style, 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

name – the specified name of the form
server_name|data_source – the specified name of the server or datasource for the specified table
[table_name] – the specified name of the table
style – the specified style
show_in_menu – if true show the name of the new form in the menu; or false for not showing
width – the width of the form in pixels
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)
```

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, primary_server_name|primary_data_source, [primary_table_name], foreign_server_name|foreign_data_source, [foreign_table_name], join_type)

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

Parameters

name – the specified name of the new relation
primary_server_name|primary_data_source – the specified name of the primary server or datasource
[primary_table_name] – the specified name of the primary table
foreign_server_name|foreign_data_source – the specified name of the foreign server or datasource
[foreign_table_name] – the specified name of the foreign table
join_type – 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;
```

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')
}
```

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();
```