Solution Model

In This Chapter

Purpose

The Solution Model is a feature since Servoy version 4.1 which allows you to manipulate different kinds of objects on-the-fly through scripting. You can perform actions such as:

- Create objects and set their properties
- Clone objects
- Manipulate properties of existing objects
- Revert objects to its original design-time state
- Remove existing objects

What It Is

The Solution Model is the blueprint of your developed solution. You can modify its blueprint during runtime, but the object still needs to be actually built to become available to the user. Compare it to building a house and where certain rules apply when you want to make changes to the already built house.

For example, if you want to change the window frames of your house, you can just remove the existing frames and replace them by new ones. However, if you want to change the foundation of your house then you need to completely tear down your house, replace the foundation and then build up your house again. This same analogy also applies to the Solution Model.

In Servoy there are two separate layers; the Solution Model layer (blueprint) and the runtime layer (the built version of the blueprint). In Servoy you can change certain elements within the runtime layer without having to touch the Solution Model layer (like changing the position of a button or the font of a label). However, if these changes are not within the blueprint and the object needs to be recreated according to the blueprint, then these changes will be lost.

It's also possible to create new instances of a form within Servoy (by using the function *application.createNewFormInstance()*). These are copies of the actual house, rather than copies of the blueprint. It is not possible for the Solution Model to grab the properties of these copies, but only from the original. So when you try to grab the new instance of a form by using the Solution Model, then it will retrieve the original form from which the new instance has been created and not the new instance itself.

Limitations

Even though the Solution Model allows a wide variety of objects that can be manipulated, there are some limitations. The following objects are (currently) not included:

- Aggregations
- Security
- Databases and their structure
- Solutions and modules

> **Note**
>
> Manipulating solutions and modules are not applicable for the Solution Model because they are not relevant during deployment. At this point the collection of solutions and modules have become one flat solution. Therefore, no (references to) solutions and modules can be made with the Solution Model.

Functionality and Basic Rules

The Solution Model has certain types of functions:

Type	Purpose
Clone	Copies of a certain objects can be created
Create	These factory functions allow you to create specific objects which can be used at properties of other objects
Get	Allows you to retrieve objects to be manipulated by the Solution Model
New	Allows you to create new objects to be manipulated by the Solution Model
Remove	Allow you to remove existing objects
wrapMethodWithArguments	Allows you to get a method, wrap it with arguments and assign it to an event.

With the Solution Model you can control different types of objects. Referring to these objects is done through so-called *JS Objects* and consist of constants, properties and/or functions.

The following list shows the objects which can be controlled by the Solution Model, its corresponding *JS Object* and from which *JS Object* it needs to be referenced from in order to use it:

Object	JS Object	Referenced from JS Object
Calculations	JSCalculation	
Styles	JSStyle	
Global variables	JSVariable	
Global methods	JSMethod	
Forms	JSForm	
Components	JSComponent	JSForm
Form variables	JSVariable	JSForm
Form methods	JSMethod	JSForm
Form parts	JSPart	JSForm
Beans	JSBean	JSForm
Buttons	JSButton	JSForm
Fields	JSField	JSForm
Labels	JSLabel	JSForm
Parts	JSPart	JSForm
Portals	JSPortal	JSForm
Tab panels	JSTabPanel	JSForm
Tabs	JSTab	JSTabPanel
Relations	JSRelation	
Relation items	JSRelationItem	JSRelation
Value lists	JSValueList	
Media	JSMedia	

> **Note**
>
> A list of all these *JS Objects* and their corresponding constants, properties and functions can be found under the *SolutionModel* node in the Solution Explorer of the Developer.

Components basically work as an umbrella for all objects which are referenced from the *JSForm* object. By retrieving an object as a component, you can change properties which are shared by all these objects or conveniently clone the object including its referenced objects and properties. Check the examples at components for more information.

When creating objects you need to make sure that the same object with the same name does not already exists. In this case you need to choose another name or remove the existing object.

As explained as an analogy in a previous section, some changes made with the Solution Model have such an impact that on other existing objects, that some of them need to be destroyed and recreated again.

> **Note**
>
> Changed objects which affect a visible form but is not that form itself or any object referenced from that form should be removed (or reverted) redisplayed.
>
> Changed form objects any object referenced from that form should be recreated or removed (or reverted) and redisplayed.
>
> Check the examples at forms for more information.

By default, changes made by the Solution Model are not persistent and are only valid for the current session in which these changes are made.

> **Note**
>
> As of Servoy version 6 a new method exists, called *servoyDeveloper.save()*. By running this method in the Debug Client or the Command Console of the Developer, changes by the Solution Model are being pushed back and saved into the worspace.

Examples

This section shows some (simple) examples of how different objects can be controlled with the Solution Model. Per object you will find how to:

- Create an object
- Retrieve an existing object
- Change a property of an object
- Remove an existing object

Create

As described in a previous section, the Solution Model includes some factury functions which allow you to create certain layout object which can be used at forms.

To create a new blue line border of width 1 and use that on an existing form:

```
var border = solutionModel.createLineBorder(1, 'blue');
var form = solutionModel.getForm('myForm');
form.borderType = border;
```

Calculations

During development, calculations can be created as stored or unstored. As soon as the name of a calculation corresponds with a database column, it automatically becomes a stored calculation. The same rule applies for the Solution Model.

To create a new calculation with name *myCalculation* on table *customers* of server *example_data* which returns *1*:

```
var calculation = solutionModel.newCalculation('function myCalculation() { return 1; }', JSVariable.INTEGER, 'db:
/example_data/customers');
```

To get the existing calculation *myCalculation* on table *customers* of server *example_data*:

```
var calculation = solutionModel.getCalculation('myCalculation', 'db:/example_data/customers');
```

To output whether or not this calculation is a stored one:

```
application.output('Stored calculation: ' + calculation.isStored();
```

To remove existing calculation *myCalculation*:

```
solutionModel.removeCalculation('myCalculation', 'db:/example_data/customers');
```

Styles

To create a new stylesheet with name *myStyle* with a default style class for forms:

```
var style = solutionModel.newStyle('myStyle' 'form { background-color: transparent; }';
```

To get existing stylesheet *myStyle*:

```
var style = solutionModel.getStyle('myStyle')
```

To add a default style class for fields:

```
style.text += 'field { background-color: blue; }';
```

To remove existing stylesheet *myStyle*:

```
solutionModel.removeStyle('myStyle');
```

Global Variables

To create a new global variable with name *myGlobalVariable* of type *TEXT*:

```
var globalVariable = solutionModel.newGlobalVariable('myGlobalVariable', JSVariable.TEXT);
```

To get existing global variable *myGlobalVariable*:

```
var globalVariable = solutionModel.getGlobalVariable('myGlobalVariable');
```

To change its default value to *abc*:

```
myGlobalVariable.defaultValue = 'abc';
```

To remove existing global variable *myGlobalVariable*:

```
solutionModel.removeGlobalVariable('myGlobalVariable');
```

Global Methods

To create a new global method with name *myGlobalMethod*:

```
var globalMethod = solutionModel.newGlobalMethod('function myGlobalMethod() { currentcontroller.newRecord(); }');
```

To get existing global method *myGlobalMethod*:

```
var globalMethod = solutionModel.getGlobalMethod('myGlobalMethod');
```

To make it appear in the menu:

```
globalMethod.showInMenu = true;
```

To remove existing global method *myGlobalMethod*:

```
solutionModel.removeGlobalMethod('myGlobalMethod');
```

Forms

Forms are most commonly used with the Solution Model and have some additional functions. The following basic functions are available for this type of objects:

- Create new forms
- Get existing forms
- Clone an existing form
- Remove an existing form
- Revert an existing form to its original design-time state

To create a new form with name *myForm*:

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

To get existing form *myForm*:

```
var form = solutionModel.getForm('myForm');
```

To disable its navigator:

```
form.navigator = SM_DEFAULTS.NONE;
```


Note

When changing an existing form which has already been loaded, you need to refresh it before the end of the method by using *controller.recreateUI()*. If the form is not refreshed then this will result in an error.

When changes are made to underlying objects which have effect on the loaded form, you need to remove (or revert) the form and and redisplay it again. Check out the examples below on how to do this.

You also need to take the following into consideration:

- when *controller.recreateUI()* is called on a form, all elements are recreated based on the solution Model for that form. Any other runtime changes to the elements will be lost, like changing the style of a button or a dynamically added tab.
- Any reference to form elements which have been manipulated on the form that is stored will become invalid, as the element is recreated.
- Function *controller.recreateUI()* cannot be used while a *Drag 'n' Drop* operation is underway on the form.

To clone existing form *myForm* to cloned form *myClonedForm*:

```
var form = solutionModel.getForm('myForm');
var clonedForm = solutionModel.cloneForm('myClonedForm', form);
```

To remove existing form *myForm*:

```
var success = history.removeForm('myForm');
if (success) {
        solutionModel.removeForm('myForm');
}
```

To revert existing form *myForm* to its original design-time state:

```
var success = history.removeForm('myForm');
if (success) {
        solutionModel.revertForm('myForm');
}
```

> **Note**
>
> Before removing or reverting the form by using the Solution Model it's important to remove any active instances of this form from the history stack.

> **Note**
>
> You can only revert a form when it exists as a physical form created in design-time. Reverting a form which is created by the Solution Model will result in an error.

Components

Components allow you to retrieve all objects which are referenced from the *JSForm* object. By retrieving an object as a component, you can change properties which are shared by all these objects or conveniently clone the object including its referenced objects and properties.

To retrieve component with name *myButton*:

```
var component = form.getComponent('myButton');
```

To hide the element:

```
component.visible = true;
```

To retrieve all available components on the form:

```
var components = form.getComponents();
```

To change the width of all elements to *200*:

```
for (var i = 0; i < components.length; i++) {
        components[i].width = 200;
}
```

To clone an existing component:

```
solutionModel.cloneComponent('myClonedComponent', component);
```

> **Note**
>
> To test to what type of object the retrieved component belongs to, you need to use the JavaScript operator *instanceof*. For example, if you want to find out if the component is a button, use: *component instanceof JSButton*

Form Variables

To create a new form variable with name *myFormVariable*:

```
var formVariable = form.newVariable('myFormVariable', JSVariable.TEXT );
```

To get existing form variable *myFormVariable*:

```
var formVariable = form.getVariable('myFormVariable');
```

To change its default value to *abc*:

```
formVariable.defaultValue = 'abc';
```

To remove existing form variable *myFormVariable*:

```
form.removeVariable('myFormVariable');
```

Form Methods

To create a new form method with name *myFormMethod*:

```
var formMethod = form.newMethod('function myFormMethod() {controller.newRecord(); }');
```

To get existing form method *myFormMethod*:

```
var formMethod = form.getMethod('myformMethod');
```

To make it appear in the menu:

```
globalMethod.showInMenu = true;
```

To remove existing form method *myFormMethod*:

```
form.removeMethod('myFormMethod');
```

Form Parts

To create a new body:

```
var part = form.newPart(JSPart.BODY, 20);
```

To retrieve the existing body part:

```
var part = form.getPart(JSPart.BODY);
```

To change its background to white:

```
part.background = 'white';
```

To remove the existing body part:

```
form.removePart(JSPart.BODY);
```

Beans

To create a tree view with name *myBean*:

```
var bean = form.newBean('myBean', 'com.servoy.extensions.beans.dbtreeview.DBTreeView', 200, 200, 300, 300);
```

To get existing bean *myBean*:

```
form.getBean('myBean');
```

To change its anchoring to top, left and bottom:

```
bean.anchors = SM_ANCHOR.NORTH | SM_ANCHOR.WEST | SM_ANCHOR.SOUTH;
```

To remove existing bean *myBean*:

```
form.removeBean('myBean');
```

Buttons

To create a new button with name *myButton* and text *Text* and attaching global method *myGlobalMethod* to it:

```
var globalMethod = solutionModel.getGlobalMethod('myGlobalMethod');
var button = form.newButton('Text', 0, 0, 80, 20, globalMethod);
button.name = 'myButton';
```

To get existing button *myButton*:

```
var button = solutionModel.getButton('myButton');
```

To change its height to *30*:

```
button.height = 30;
```

To remove existing button *myButton*:

```
form.removeButton('myButton');
```

Fields

To create a new text field with name *myField* with form variable *myFormVariable* as its dataprovider:

```
var formVariable = form.getFormVariable('myFormVariable');
var field = form.newField(formVariable, JSField.TEXT_FIELD, 0, 0, 100, 200);
field.name = 'myField';
```

To get existing field *myField*:

```
var field = form.getField('myField');
```

To change its display type to a *text area*:

```
field.displayType = JSField.TEXT_AREA;
```

To remove existing field *myField*:

```
form.removeField('myField');
```

Labels

To create a new label with name *myLabel* with text *Text*:

```
var label = form.newLabel('Text', 0, 0, 100, 20);
label.name = 'myLabel';
```

To get existing label *myLabel*:

```
var label = form.getLabel('myLabel');
```

To change its horizontal alignment to *center*:

```
label.horizontalAlignment = SM_ALIGNMENT.CENTER;
```

To remove existing label *myLabel*:

```
form.removeLabel('myLabel');
```

Portals

To create a new portal with name *myPortal* based on relation *myRelation*:

```
var relation = solutionModel.getRelation('myRelation');
var portal = form.newPortal('myPortal', relation, 0, 0, 500, 500);
```

To get existing portal *myPortal*:

```
var portal = form.getPortal('myPortal');
```

To make it resizable:

```
portal.resizable = true;
```

To remove existing portal *myPortal*:

```
form.removePortal('myPortal');
```

Tab Panels

To create a new tab with name *myTabPanel*:

```
var tabPanel = form.newTabPanel('myTabPanel', 0, 0, 500, 500);
```

To get existing tab panel *myTabPanel*:

```
var tabPanel = form.getTabPanel('myTabPanel);
```

To add a new tab with name *myTab* based on relation *myRelation*:

```
var relation = solutionModel.getRelation('myRelation');
var tab = tabPanel.newTab('myTab', 'Text', myRelatedForm, relation);
```

To remove existing tab panel *myTabPanel*:

```
form.removeTabPanel('myTabPanel');
```

Relations

To create a new relation with name *myRelation* between tables *customers* and *orders*:

```
var relation = solutionModel.newRelation('myRelation', 'db:/example_data/customers', 'db:/example_data/orders',
JSRelation.INNER_JOIN);
```

To get existing relation *myRelation*:

```
var relation = solutionModel.getRelation('myRelation');
```

To create new a new relation item based on fields *id* and *customer_id*:

```
relation.newRelationItem('id', '=', 'customer_id');
```

To remove existing relation *myRelation*:

```
solutionModel.removeRelation('myRelation');
```

Value Lists

To create a new value list with name *myValueList*:

```
var valueList = solutionModel.newValueList('myValueList', JSValueList.CUSTOM_VALUES);
```

To get existing value list *myValueList*:

```
var valueList = solutionModel.getValueList('myValueList');
```

To set custom values for the value list:

```
valueList.customValues = '1\n2';
```

To remove existing value list *myValueList*:

```
solutionModel.removeValueList('myValueList');
```

Media

To create new media with name *myMedia*:

```
var media = solutionModel.newMedia('myMedia', plugins.http.getMediaData('http://www.servoy.com/images
/logo_servoy.gif'));
```

To get existing media *myMedia*:

```
var media = solutionModel.getMedia('myMedia');
```

To change its content:

```
media.bytes = plugins.http.getMediaData('http://servoy.com/images/headerimages/open_source.jpg');
```

To remove existing media *myMedia*:

```
solutionModel.removeMedia('myMedia');
```


Note

For all examples of functions, properties and functions, check out the Programming Reference Guide on the Wiki or select one form the Solution Explorer in the Developer and choose *Move sample*.