Javacript API exercise

Table of Contents

knowledge@outsystems.com

www.outsystems.com

Outline

In this exercise we will focus on the predefined platform ***$keywords*** available for use in the Javascript code.

Upon completion, we'll have built several screen actions that, by using JavaScript, can execute common application operations like checking a role or navigate to another screen.

Resources

This exercise can be set up on a new application. For further mention on this exercise it is implicit to have a completely blank Reactive Web App with one single UI module.

Scenario

In this exercise, we'll start by creating a new Reactive Web application named ***JavascriptFrontend***, and add to it a single Reactive Web module with a name of ***JavascriptAPIDemo***, that will implement the application's UI.

How-To

In this section, we will provide detailed step-by-step instructions on how to complete the exercise. **If you already finished the exercise on your own, great! You don't need to do it again.** If you didn't finish the exercise, that's fine, we are here to help you!

Getting Started

To start this exercise, we need first to create a new Reactive Web Application from scracth and name it **JavascriptFrontend**. For the time being you are free to choose any color in the app name and description form or optionally upload a custom icon. Next, please start to create a Reactive Web Module type and name it **JavascriptAPIDemo** and we are done.

Before we start, it is necessary to setup a new screen and it's content:

Create a new Blank Screen and name it **Home**. As it is the only Screen in your module automatically will be set as your Module default screen.

1. Using the $parameters keyword

In this section we are going to create an empty container and a button on our screen. The button screen action will use the Javascript tool node to execute a piece of code that will hide the container. We will make use of the **$parameters** keyword to have access to the Javascript node input parameters value.

On the **Home** screen:

1) insert a new Container widget into the MainContent section of the Screen and name it ***MyContainer***.
    - Inside the container insert the following text: **"My name is MyContainer"**.
2) Outside the Container place a new text input widget and set the Variable to a new local text variable named ***WidgetName***.
3) Go to the screen CSS editor and add a new CSS style class as follows:

```
.invisible-element{
display: none;
}
```

4) After the input widget, place a new button with the text "*Hide element*" and set

the "On Click" event to a new Screen Action with the name ***HideElementWithName***.

5) Go to the ***HideElementWithName*** action flow and drag a JavaScript node after the Start node.

6) Open the Javascript node and on the JavaScript Editor popup:

   a) add a new Input Parameter named ***WidgetIdentifier*** of Text datatype

   b) add the following Javascript code:

```
var element =
document.getElementById($parameters.WidgetIdentifier);
if(element !== null){
element.classList.add("invisible-element");
}
```

7) Close the Javascript editor and on the Properties for the Javascript node, set the ***WidgetIdentifier*** input paramenter to be the local screen variable ***WidgetName***.

After you complete these steps the screen content should be as the image below.

Publish your changes and open the page in the browser.

Test the input field and fill in the input with **"MyContainer"** and click the button. What happened? Did the container text disappeared?

Congratulations! You've created a sample of JavaScript code inside a screen action. This code hides a given widget on a screen by setting input parameters to the Javascript node with the widget name.

2. Using the $actions keyword

In this section we will build a new reusable screen action that, through Javascript, will be able change a page element background color. Then, by using the **$actions** of the Javascript API, we will execute the reusable action on other screen actions flows with additional Javascript nodes.

Let's add three new Container widgets in the screen and give them names: ***MyContainer1***, ***MyContainer2*** and ***MyContainer3***.

At the screen CSS Editor add two new CSS style classes as follows:

```
.container{
padding: 10px;
background-color: cyan;
}

.pink-background{
background-color: pink;
}
```

Go to each one of the new three Containers **Properties** tab and set the **Style Classes** property to: ***"container"***.

At the canvas, did you see any changes on the containers layout?

2.1. Reusable Client Action

1) Go to the Logic tab and create a new Client action named ***SetWidgetBackGroundColor*** with the following action parameters:

- an input parameter named ***WidgetName*** of Text datatype and mandatory.
- an input parameter named ***BackgroundColor*** of Text datatype and mandatory.

1) Open the ***SetWidgetBackGroundColor*** client action and add a new Javascript node after the Start node.

2) Open the Javascript node and on the Javascript Editor popup:

   - add a new Input Parameter named ***WidgetIdentifier*** of Text datatype
   - add a new Input Parameter named ***WidgetBackgroundColor*** of Text datatype
   - add the following Javascript code:

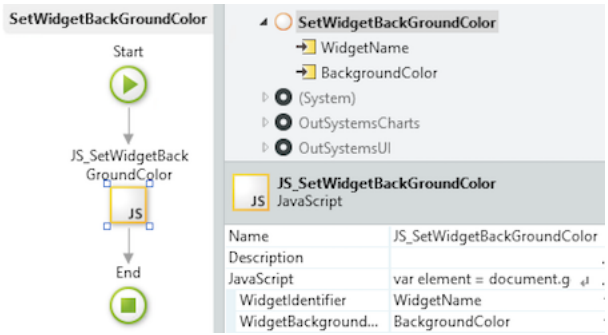
```
var element =
document.getElementById($parameters.WidgetIdentifier);

if(element !== null){
element.classList.add($parameters.WidgetBackgroundColor+"-
background");
}
```

3) Close the Javascript editor window and on on the Properties for the Javascript node:

   - set the ***WidgetIdentifier*** input parameter to be the action's ***WidgetName*** input parameter.
   - set the ***WidgetBackgroundColor*** input parameter to be the action's ***BackgroundColor*** input parameter.

After you complete these steps the action details should be as follows.



2.2. Call an action

Go back to the screen and create:

1) a new local screen variable named **InputColor** of Text datatype.
2) a new local screen named **NrOfWidgets** of Integer datatype.

At the screen canvas:

1) add a new expression widget and set it's value to:

```
If(NrOfWidgets <> 0, NrOfWidgets + " widget(s) are now "+
InputColor, "")
```

2) add a new input widget and set it's Variable to be the **InputColor** local variable just created.

On the right hand side of the input widget place a new Button widget with the text "*Set Color*" and set its "On Click" event to a new Screen Action with the name ***SetBackgroundColorToElementsOfClass***.

On screen the expected result would be similar to:

On the new ***SetBackgroundColorToElementsOfClass*** screen action, add a new input parameter of Text datatype named ***ClassName***, setting Mandatory to **No**, and the Default Value to ***"container"***.

Next, go to the ***SetBackgroundColorToElementsOfClass*** action flow and place a JavaScript node after the Start node.

Open the newly added JavaScript node and, on the Editor popup window:

1) add a new Input Parameter named ***ClassName*** of Text datatype
2) add a new Input Parameter named ***WidgetBackgroundColor*** of Text datatype
3) add a new Output Parameter named ***NrElements*** of Integer datatype
4) add the following JavaScript code:

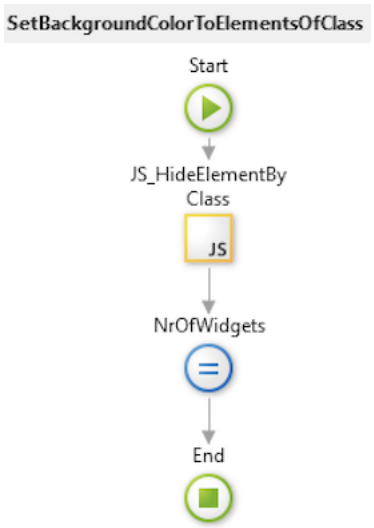
```
var elements =
document.getElementsByClassName($parameters.ClassName);

for (var i=0, len=elements.length|0; i<len; i=i+1|0){

if(elements[i] !== null && elements[i].id !== ""){
$actions.SetWidgetBackGroundColor(elements[i].id,
$parameters.WidgetBackgroundColor);
$parameters.NrElements++;
}
}
```

5) Close the JavaScript editor window and check the JavaScript element properties to assign values to its parameters:
   - assign to the ***ClassName*** input parameter the value of the ***ClassName*** action input parameter.
   - assign to the ***WidgetBackgroundColor*** input parameter the value of the ***InputColor*** local screen variable.
6) Add a new Assign node after the JavaScript node, and set the value of the **NrOfWidgets** local screen variable to be the **NrElements** output parameter of the Javascript node.

After you complete these steps the flow should be as the image below.



Publish your changes and open the page in the browser.

Test the input field that is next to the **Set Color** Button and fill in the input with "pink" and click the button.

What happened?

Was any container updated to have a pink background?

Did the number of updated containers match the expression stating how many widgets are now pink?

Extra challenge!

Create new CSS style classes with additional background colors and name the classes so they can be used by the ***SetWidgetBackGroundColor*** client action.

3. Using the $roles keyword

3.1. Verify a user role

Go to the Logic tab and create a new role named ***Manager**.

Also in the Logic tab create a new Client action named ***IsManager*** with a boolean output parameter named **HasRole**.

Edit the action flow an place a new JavaScript node after the Start node.

Open the Javascript node and on the JavaScript editor popup:

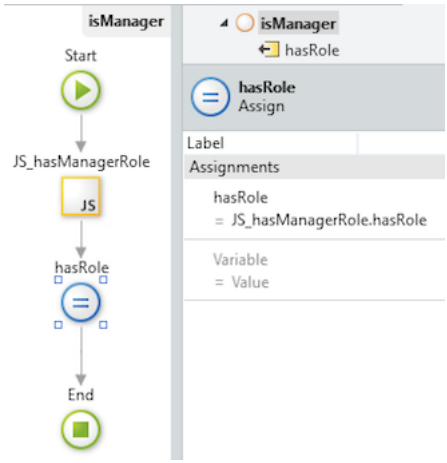
1) add a new Output Parameter named ***HasManagerRole*** of Boolean datatype

2) add the following JavaScript code:

```
$parameters.HasManagerRole =
$public.Security.checkIfCurrentUserHasRole($roles.Manager);
```

3) Close the Javascript editor window and place a new Assign node after the Javascript node. Set the value of the **HasRole** output parameter of the client action to be the **HasManagerRole** output parameter of the JavaScript.

After you complete these steps the flow should be as the image below.



Choose one of the previously created buttons and edit its screen actions flow. After the Start node, execute the **IsManager** client action. Afterwards add an If statement to create a flow branch in order to stop the execution when the user does not have the Manager role.

The If Condition should be:

```
IsManager.HasRole
```

On the False branch place a new Feedback Message node stating that the user does not have the necessary role. The True branch should continue with the remanining existing flow.

After you complete these steps the flow should look as follows.

Publish your changes and open the page in the browser.

Test the Button action you have just modified and check if you got the message. Why?

Do the necessary changes in the Users application so that the user stops seeing the message.

4. Using the $public keyword

4.1. Showing a Feedback message

Create a new Screen and call it "About".

Go to the Screen Properties, and on the Events separator, choose the ***OnReady*** event and create a new screen action to handle it.

By default the new screen action will be named ***OnReady*** too.

Edit the ***OnReady*** action flow and place a new JavaScript node after the start node.

Open the Javascript node and add the following code on the JavaScript Editor popup:

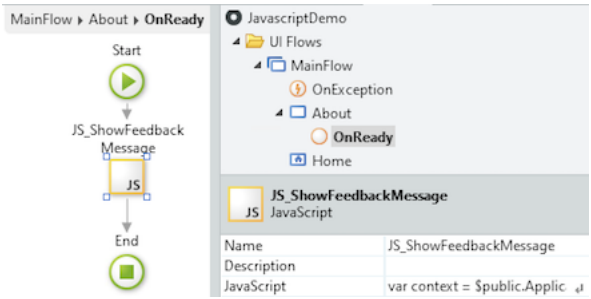
```
var context = $public.ApplicationContext.getCurrentContext();

var screenName = context.screenName;

var message = "Welcome to "+screenName;

$public.FeedbackMessage.showFeedbackMessage(message, 0, false, "",
true, true);
```

After you complete these steps the flow should be as the image below.



4.2. Navigate to a screen

Go back to the **Home** screen and create a new screen action named * **NavigateToScreen**. Add a new input parameter of Text datatype named **InputScreenName**.

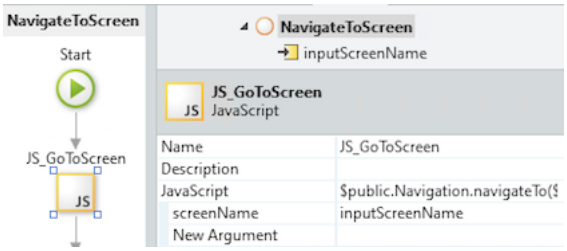
Edit the action flow and place a new JavaScript node after the Start node.

Open the JavaScript node and add the following JavaScript code:

```
$public.Navigation.navigateTo($parameters.InputScreenName);
```

Close the Javascript editor window.

After you complete these steps the action and JavaScript node details should look as follows.



On the screen create a new link and set the On Click event to the ***NavigateToScreen*** action.

Set its **InputScreenName** manually with name of the newly created screen (**"About"**).

The link details should be like the image below.

Publish your changes and open the inital page in the browser.

Test the link, and check if the navigation to the new screen occurs.

Also when the new screen is displayed you should get a welcome feedback message with the screen name.