
Apps Framework API

Version 1.00

Samsung Smart TV

1. FRAMEWORK API	4
1.1. BASIC FUNCTIONS	4
1.1.1. exit().....	4
1.1.2. returnFocus().....	5
1.1.3. loadJS()	9
1.1.4. readfile().....	12
1.1.5. getInfo()	12
1.1.6. setData().....	12
1.1.7. getData()	13
1.1.8. plugin()	16
1.2. KEY EVENT FUNCTION	19
1.2.1. Block().....	19

Preface

- **Purpose of Document**

This document is written for the users of Samsung Smart TV SDK, who would like to use the Apps Framework and its inbuilt functions for the first time. This document contains detailed examples to understand the syntax of the Framework API functions better, and to make use of them in the applications. After reading the document, the reader should be able to make full and effective use of the functions in API framework and create basic applications.

- **Target Readers**

This document targets the first time users of Samsung Smart TV SDK and its new inclusion, the Samsung WYSIWYG Editor. However, the reader is expected to have basic knowledge about Java Scripting, CSS and HTML. The document explains only the use of Samsung Smart TV SDK and its features. There are no explanations provided about the programming syntax and conventions of the scripting language and the programming language used.

1. Framework API

The Samsung Smart TV SDK framework API offers a list of functions that operate on the object instance of the framework. All these functions enhance the functionality of the framework. This section explains the uses of these functions with an example wherever possible.

1.1. Basic Functions

The basic functions offered by the framework are as follows. These functions are called by the framework within itself to carry out some specific tasks. However, these functions are also available to the developer for programming.

- *exit()*
- *returnFocus()*
- *loadJS()*
- *readFile()*
- *getInfo()*
- *getData()*
- *setData()*
- *plugin()*

1.1.1. *exit()*

Syntax	<i>\$.sf.exit (Boolean)</i> <i>Boolean: true will exit to the TV screen</i> <i>False will exit to the application manager.</i>
Description	This function is called to exit out of the application and return to the Application manager or the TV screen. It does this depending on the type of argument it takes.

To try it on the ENTER key on the remote control just add the line of code in the ENTER case of the `handleKeyDown` event.

```
case $.sfKey.ENTER:  
    $.sf.exit(false);  
    break;
```



Fig 1 Emulator with Contents Home/Application Manager

Note that, there is an option called Emulator -> Run Emulator with Contents Home, in the Samsung Smart TV SDK. The Contents home in the SDK is just how the Application Manager will look like, on the Samsung Smart TV. Therefore, in order to see the application exiting to the application manager, when pressing the ENTER key (as in the code above), run the emulator with the Contents Home and then choose the application from there. Now press the ENTER key to see the application exit to the Application manager.

1.1.2.returnFocus()

Syntax	<code>\$.sf.returnFocus()</code>
Description	The return focus function returns the focus to the framework.

A new example for this function is given. A keen observer would have already noticed a list of inbuilt functions that is being called by the framework, whenever the application is manipulated with the components. It is usually displayed in the log manager. This is one such function. For example, if you have already experimented with the Pop up or Date picker components, you would

Apps Framework API

have noticed that whenever a response is given for the Popup/Date picker, and the popup box closes, the `$.sf.returnFocus()` appears in the command window of the Samsung Smart TV SDK. This is because, the popup and a few other components has some focus stealing functions that steals the focus from the framework/scene onto itself. So upon closing the Popup, the framework calls the above function to return the focus to the framework.

Example

Step1: - Open the Samsung WYSIWYG Editor and add two buttons and one popup.

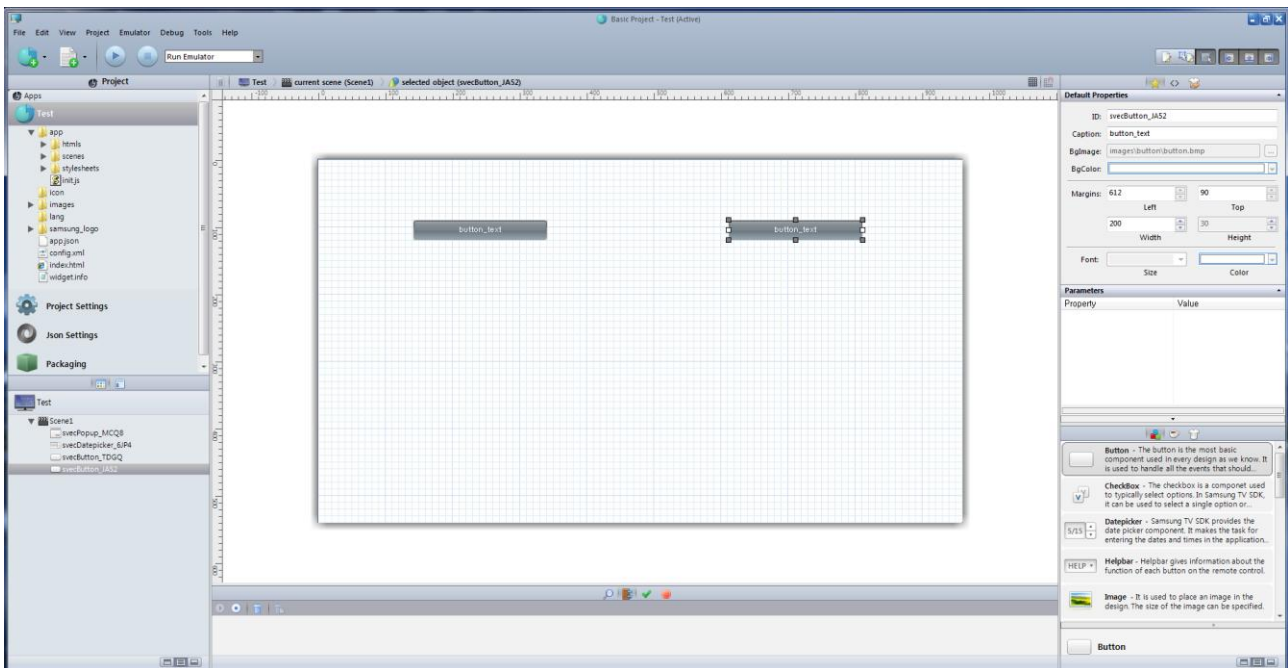


Fig 2 WYSIWYG Editor with 2 Buttons and popup (invisible)

For better understanding add a popup component with 2 buttons OK and CANCEL. Now, the popup is present in the above editor, however the popup is not seen unless `show()` property of the popup is set to true during runtime as explained in the above example. Now export the design.

Step2: - Now in `Scene1.js` that appears, add the following code first to navigate/switch between the two buttons just added

```
function SceneScene1(options) {  
    this.options = options;  
    this.index = 1;  
}  
SceneScene1.prototype.initialize = function () {
```

Apps Framework API

```
    alert("SceneScene1.initialize()");  
  
    // this function will be called only once when the scene manager show this scene first time  
  
    // initialize the scene controls and styles, and initialize your variables here  
  
    // scene HTML and CSS will be loaded before this function is called  
  
    $('#svecPopup_ok_cancel_50Z0').sfPopup({text:'popup text', Num:'2', callback:  
function(rlt){$.sfScene.get('Scene1').popupcallback(rlt)}});  
  
    $('#svecButton_YQ4I').sfButton({text:'button_bgimage', width:'200px'});  
    $('#svecButton_HBST').sfButton({text:'button_bgimage', width:'200px'});  
    $('#svecButton_YQ4I').sfButton('focus');  
}  
SceneScene1.prototype.handleKeyDown = function (keyCode) {  
    alert("SceneScene1.handleKeyDown(" + keyCode + ")");  
  
    //TODO : write an key event handler when this scene get focused  
  
    switch (keyCode) {  
        case $.sfKey.LEFT:  
            this.index = 1;  
            if (this.index == 1)  
            {  
                $('#svecButton_YQ4I').sfButton('focus');  
                $('#svecButton_HBST').sfButton('blur');  
            }  
            break;  
        case $.sfKey.RIGHT:  
            this.index = this.index+1;  
            if(this.index == 2)  
            {  
                $('#svecButton_HBST').sfButton('focus');  
                $('#svecButton_YQ4I').sfButton('blur');  
            }  
  
            break;  
    }  
}
```

Step3: - To show the popup box add the following code. The popup is displayed on pressing UP key of the remote control.

Apps Framework API

```
case $.sfKey.UP:  
    $('#svecButton_YQ4I').sfButton('focus');  
    $('#svecPopup_ok_cancel_50Z0').sfPopup('show');  
  
break;
```

If noticed, it is seen that the popup that appears, steals the focus from the screen and you will now be able to switch between only 2 buttons (OK and CANCEL) which are present on the popup that appears. It is no longer possible to switch between the two buttons you have added in the design.

Step4: - Now in order to return the focus to the framework without allowing the Popup to steal the focus, and be able to again switch between the buttons that were added previously, insert the following lines of code appropriately.

```
case $.sfKey.UP:  
    $('#svecButton_YQ4I').sfButton('focus');  
    $('#svecPopup_ok_cancel_50Z0').sfPopup('show');  
    $.sf.returnFocus();  
  
break;
```

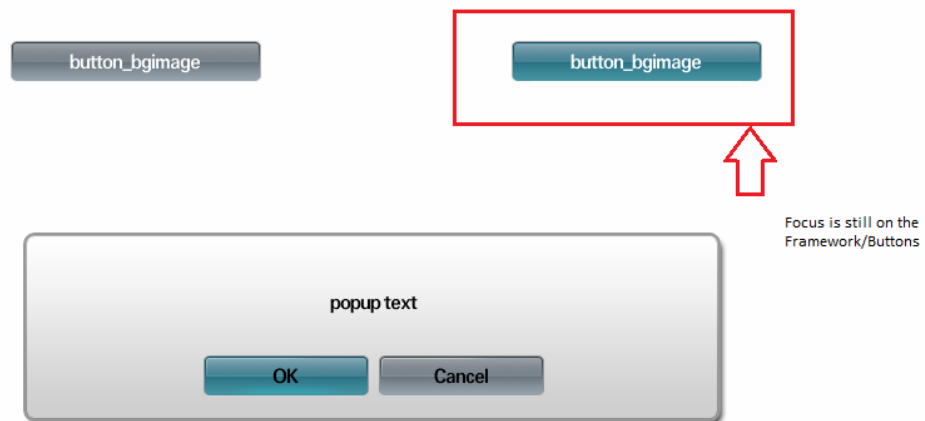


Fig 3 Emulator with Focus on the framework (invisible)

If the focus has to be returned to the popup again, set the `show()` property of the popup to true again or remove the `returnFocus` function that has been just included. Note that this is just an example to explain how the function works. `returnFocus()` function is mostly used by the framework within itself. The developer might hardly find the need to use them in the application.

1.1.3.loadJS()

Syntax	<code>\$.sf.loadJS('JavaScriptFileName.js')</code> <code>JavaScriptFileName.js</code> (String)
Description	Loads the JavaScript files dynamically

Often, a developer might need to add files dynamically within the application on the fly, thereby reducing the time and memory that it takes to load the files initially. This functionality can be made possible by the function `loadJS`.

An example application is shown below. Before that, there are a few things that a developer might need to take a note of. While loading a JavaScript file, the `loadJS()` function searches for the file, starting from the (your installation folder)/MyApps/ directory. So, any file that the user might wish to load in an application must be present inside this folder. And proper paths for files placed anywhere deeper within directory, has to be specified appropriately.

For this example, a JavaScript file `api.js` to add two numbers is written and placed in MyApps folder. There is a Scene1 with a label that is created and exported from the Samsung WYSIWYG Editor. The file `api.js` is loaded in `Scene1.js` when required and the result of this sum is printed in the label.

Example

Step1: - Create a Scene with a label and import it to the Samsung Smart TV SDK.

Step2: - In the MyApps folder that is created by default (when exporting the design) add a new file called as `api.js` and add the following JavaScript code to add two numbers.

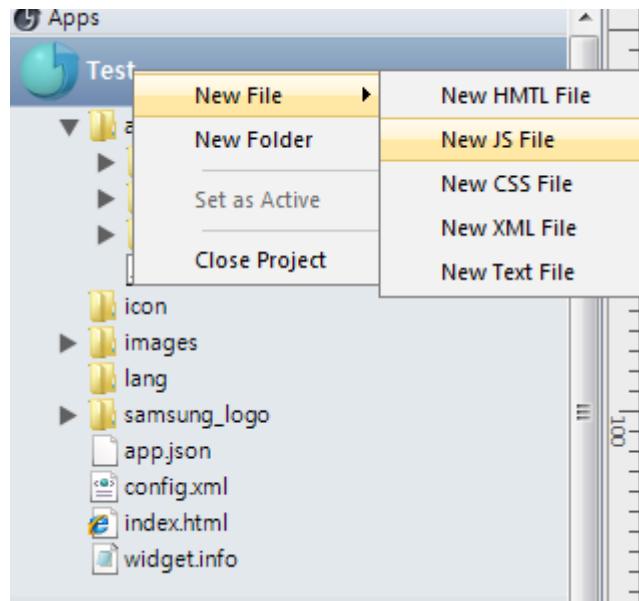


Fig 4 Project Explorer showing how to add a new JavaScript file in the project folder

```
function Foo()
{
    this.x = 1;
}
Foo.prototype.AddX = function(y)    // Define Method
{
    this.x += y;
    return this.x;
}
obj = new Foo;
var ans = obj.AddX(5);
```


Step3: - Now write code in the Scene1.js to load the file into the application when needed and show the value.

```
SceneScene1.prototype.handleKeyDown = function (keyCode) {
    alert("SceneScene1.handleKeyDown(" + keyCode + ")");
    //TODO : write an key event handler when this scene get focused
    switch (keyCode) {
        case $.sfKey.LEFT:
            break;
        case $.sfKey.RIGHT:
            break;
        case $.sfKey.UP:
```

Apps Framework API

```
        break;
    case $.sfKey.DOWN:
        $('#svecLabel_UZIQ').html("The answer from api . js is " + " " + ans);
        // $('#svecLabel_UZIQ').sfLabel({text:'Label is printing Answer from api . js. The answer = ' + " " + ans , width:'410px'});
        break;
    case $.sfKey.ENTER:
        $.sf.loadJS('api.js');
        break;

    default:
        alert(" Unhandled key !");
        break;
}
}
```



The answer from api.js is 6

Fig 5 Emulator output of loadJS

The JavaScript file gets loaded when the ENTER key on the remote is pressed and the result is displayed in the label on pressing the DOWN key. Note that if loadJS() is given in one of the functions like initialize(), the file will be loaded when the application starts. This is because,

initialize() is called when the application is initialized. And if this happens, the entire purpose of the function is lost. Hence the developer needs to make a wise decision of when to load the file appropriately. The value of the result can be displayed in the Label in either ways shown above. Try both of them.

1.1.4.readFile()

Syntax	<i>\$.sf.readFile('FileName')</i> <i>FileName</i> (String)
Description	Similar to loadJS() function, readFile() reads a file and returns the contents of file in String format.

1.1.5.getInfo()

Syntax	<i>\$.sf.getInfo(value)</i> <i>Value</i> (String) <i>language, country, modelid, area, firmware</i> (any one at a time)
Description	The getInfo() function is used to get TV Environment values.

The TV can have lot of information about its operating environment such as country, language, modelID, etc. It is very likely that a user might want to use these values for developing an application.

For example, the TV might have information about the present country in its internal memory. A developer might want to restrict certain content of his application, based on the country of the viewer. Therefore, the developer might want to use this information about the country, that is present in the internal memory of the TV. getInfo() is very useful for this purpose. It helps the developer to retrieve the required information from the TV's internal memory. Another developer might want to automatically select a language based on the language of the TV. Likewise, this function can find several other applications.

1.1.6.setData()

Syntax	<i>\$.sf.setData(key, value)</i> <i>Key</i> (String) <i>Name of the environment variable</i> <i>value</i> (String) <i>value of the variable</i>
Description	This function is used to set the value of an application environment value. This data is application specific, and can be set as per application requirements.

1.1.7. *getData()*

Syntax	<code>\$.sf.getData(key)</code> <i>Key</i> (String) Name of the environment variable
Description	This function is used to get the value of the environment variable that has been set by the user.

This function can be used only after using `setData()`. Because, it is possible to get a value only after the value has been set.

Example

The example program for the four functions above has been given in the same application. The application has a Scene with a label, which displays the text according to the function which is triggered. When the ENTER button is pressed, the label displays the text of the `readFile()` function. When the DOWN key is pressed, the label displays the `modelID` of the `getInfo()` function. When the LEFT key is pressed, the `setData` function sets the value of the TV environment variable name 'myName'. When the RIGHT key is pressed the value that is just set is displayed in the label.

Step1: - Just as in the previous examples add a Scene and a Label and export it to the Samsung Smart TV SDK.

Step2: - Similar to adding a JavaScript file, add any random text file to the project folder MyApps. Include any text in the file that has been just added.

Step3: - Now in `Scene1.js` add the following code for the `readFile()` under the ENTER `keyCode` of the `handleKeyDown`.

```
case $.sfKey.ENTER:
    var str = $.sf.readFile('new.txt');
    //$('#svecLabel_50Z0').sfLabel({text:" " + str, width:'385px'});
    $('#svecLabel_50Z0').html(str);
    // See the difference between both and note them!    ".html" does not give you blank spaces, whereas, normal case gives
    you EOL boxes
    break;
```

Now the output looks as follows when the ENTER key of the remote is pressed. It is shown in

Figure 6.

Step4: - Now in order to see the working of `getInfo()` add the following code in the `DOWN` `keyCode` of the `handleKeyDown()`. When the user presses `DOWN` key of the remote, the value obtained by the `getInfo` is displayed in the label. The output of which has been shown in Figure 7.

```
case $.sfKey.DOWN:  
    var str = $.sf.getInfo('modelid');  
    $('#svecLabel_50Z0').html("model ID is " + " " +str);  
    break;
```



Fig 6 Emulator showing the output of `readfile()`

Step5: - To experiment how `setData()` and `getData()` works add the following code in the `LEFT` and `RIGHT` cases of the `keyCode`. When the `LEFT` key is pressed the value is set and when the `RIGHT` key is pressed the value that is set is displayed. Figure 8 shows the same.

Apps Framework API

```
case $.sfKey.LEFT:
    $.sf.setData('myname', 'Jack');
    break;
case $.sfKey.RIGHT:
    var mystr = $.sf.getData('myname');
    $('#svecLabel_50Z0').html('This is the value from getData(). Name of the config \'myName\' is ' + " " + mystr);
    break;
```



Fig 7 Emulator showing the output of getInfo()



Fig 8 Emulator showing the output of `getData()` and `setData()`

1.1.8.plugin()

Syntax	<code>\$.sf.plugin ('name')</code> <i>name</i> (String) <i>TV, TVMW, NNAVI, AUDIO, APPCOMMON, FRONTPANEL, IMAGEVIEWER, PLAYER</i>
Description	The API framework function <code>plugin()</code> is used to create an instance of a plugin.

To understand more about plugins you are encouraged to read the Device API documentation. In the naive way, the programmer has to insert an html object and write JavaScript function to access and modify the properties of the created HTML object. This usually requires, inserting and creating an object of a HTML element.

However, `plugin ()` function makes this job easier, by creating an object to the plugin. If it doesn't exist, it creates the plugin and gives a handle, to access the properties of the plugin. The example illustrates the use of `plugin ()` function.

Example for plugin () function

Apps Framework API

To understand the plugin() function well, create a scene with 3 buttons and a label. Each button corresponds to the name of a Plugin() function. And the label displays the value returned by the plugin property.

Step1: -Create a scene with 3 buttons and a label. Export the design to the Samsung Smart TV SDK.

Step2: - Write code to navigate between the 3 buttons as explained in the Nesting Scenes example program. It's not shown here again.

Step3: - Now write code for each button as shown below to create an object of the plugin and access its properties. To know more about the different plugins available and its object's properties, refer to the DeviceAPI documentation. Add the following code in the ENTER key case of the handleKeyDown function.

```
case $.sfKey.ENTER:
    if(this.index == 1) // refers the button index number (in this case button 1)
    {
        var tvObj = $.sf.plugin('Audio');
        var vol = tvObj.GetVolume();
        var mute = tvObj.GetSystemMute();
        $('#svecLabel_UZIQ').html("System Volume" + " " + vol + ", " + "System Mute?" + " " + mute );
    }
    if(this.index == 2)
    {
        var scObj = $.sf.plugin('AppCommon');
        var ans = scObj.RegisterAllKey( );
        $('#svecLabel_UZIQ').html("Is all key registered?" + ans );
    }
    if(this.index == 3)
    {
        var winObj = $.sf.plugin('NNAvi');
        var res = winObj.GetModelCode();
        $('#svecLabel_UZIQ').html("ModelNumber is " + " " + res );
    }
    break;
```

The emulator output of the code is shown.

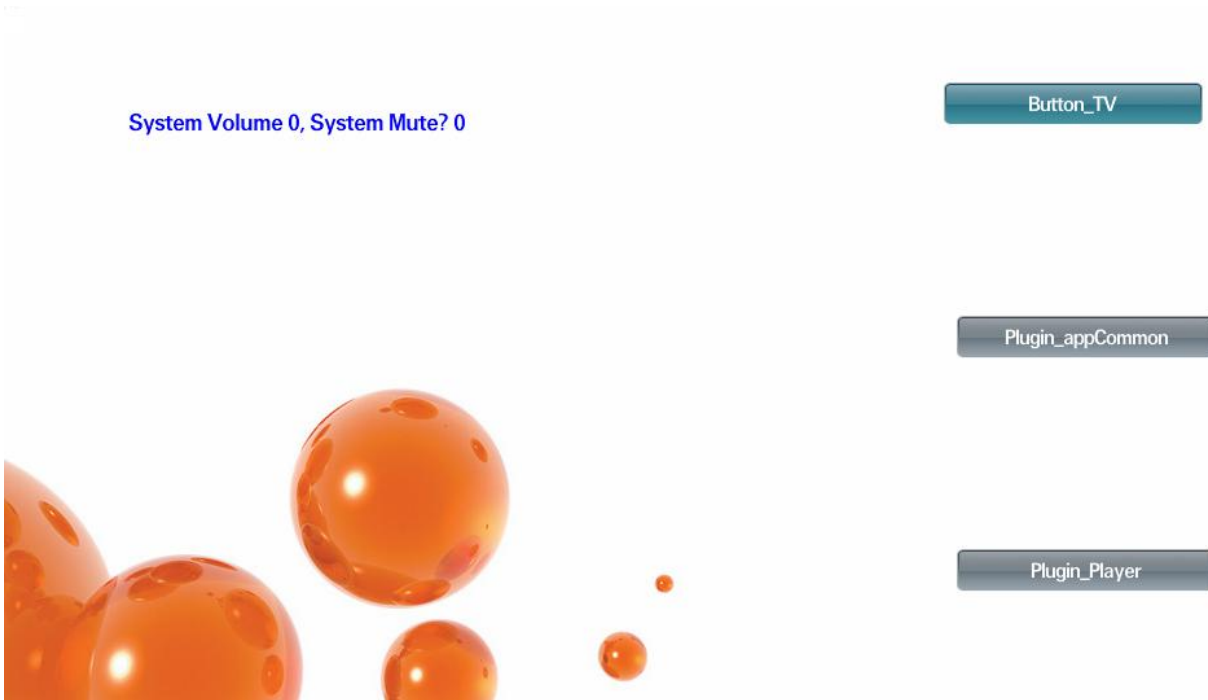


Fig 9 Emulator showing the output of plugin()

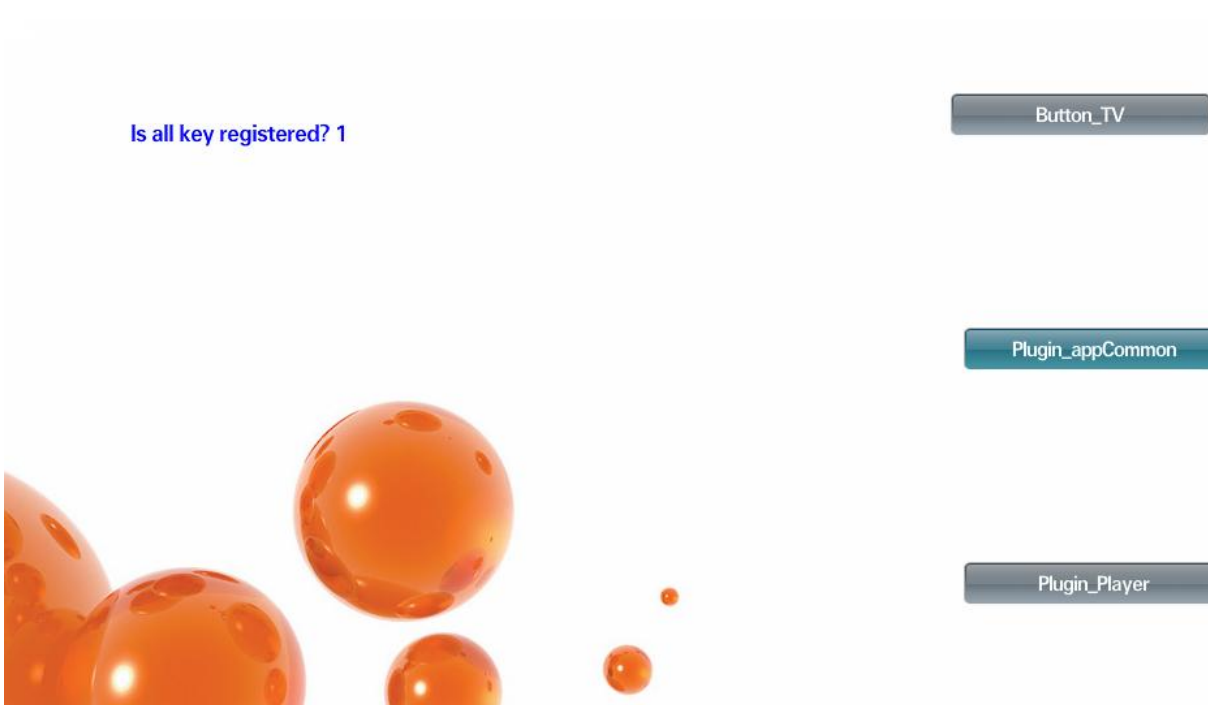


Fig 10 Emulator showing the output of plugin()

This ends the illustration for the Framework API functions. You are ready to experiment it all by yourself.

1.2. Key Event Function

1.2.1. *Block()*

This section describes a Key Event Function that would be very useful while creating an Samsung Smart TV application. `Block()` function is a key event function that is used to block the default action of the RETURN and the EXIT key. As known already, RETURN key takes the application to the Contents Home/Application Manager and EXIT key takes it to the TV screen. Note that the `block()` function is used to block only the RETURN and EXIT keys and cannot be used to block any other key.

Quite often the application of `block()` function is very useful. For instance, consider that an application is displaying a popup box. The application exits to the application manager or TV screen when the RETURN or EXIT keys are pressed. However, if the default action is not required to happen, until the user has selected a response for the popup, block the default action of the RETURN and EXIT keys when the popup is displayed. In the example that follows one such illustration is given. To see how it works, do as follows.

Example

Step1: - Create two scenes in the WYSIWYG area with a label in each scene with appropriate text (just to know which scene you are in). And add a popup box in the Scene2.

Step2:- Now export the Scenes to the Samsung Smart TV SDK. Write a program to switch between Scenes just as in the tutorial shown for `show()`, `focus()` and `hide()` functions. Now there is an application that can switch between scenes.

Step3:- Now in Scene2, write a code to display the popup box. Displaying a popup box has already been explained in tutorial of `get()` function. Now that a popup is displayed, add the code to block the RETURN and EXIT keys when the popup is displayed.

Step4: - When `block()` function is given within the cases of RETURN and EXIT keys, it always blocks the action of the RETURN and EXIT keys. To test how this works, add the following code in Scene1

```
case $.sfKey.RETURN:
    $.sfKey.block();
    break;
case $.sfKey.EXIT:
    $.sfKey.block();
    break;
```

Now run the application in Emulator with Contents Home (Emulator -> Run emulator with Contents Home) . Within the application, while in Scene1, if the RETURN or EXIT keys are pressed, it is seen that the application does not exit to the Contents Home/Application Manager or the TV.

Step5: - Now in Scene2, block the RETURN and EXIT keys only when the popup is displayed using some special condition.

```
function SceneScene2(options) {
    this.options = options;
    this.bool = 1;
}
case $.sfKey.RIGHT:
    this.bool = this.bool+1;
    $('#svecPopup_5KT5').sfPopup('show');
    break;
case $.sfKey.RETURN:
    if(this.bool==2)
    { $.sfKey.block();
    }
    else { }
    break;
case $.sfKey.EXIT:
    if(this.bool==2)
    { $.sfKey.block();
    }
    else { }
```

Apps Framework API

```
break;
```

```
SceneScene2.prototype.popupcallback = function (data) {  
  this.bool = 1;  
  alert('SceneScene2.popupcallback() : '+data);
```

The above code changes the value of bool to 2 whenever the popup is shown and changes it to 1 whenever the popup is closed. Based on the value of bool the RETURN and EXIT keys are blocked.

It can be tried again by running the application in the Emulator with Contents Home/ Application manager. Now open the application and goto Scene2 and display the popup box. When the RETURN or EXIT key in the emulator is pressed, the emulator does not exit to the content manager. However, when the Popup box is closed, it does exit to the Application manager while in Scene2.

This completes the illustration of the Key Event Functions. You are now free to experiment it all by yourself.