# IDE Help Guide

Published 2014-10-28 | (Compatible with SDK 4.5,5.0,5.1 and 2013,2014 models)

> This document is the user guide for the Samsung Smart TV SDK IDE, which helps Samsung Smart TV Application developers better understand and use the IDE.

Contents



**** This class will not be supported in 2015.**

*All functionalities of AppsFramework are more improved, integrating with CAPH. Therefore Appsframework is not supported since 2015 Smart TV. To use functions of Appsframework, refer to **here**.*

# Prerequisites

The required environments are listed as follows:

For the operating system:

Windows XP / 7 (recommended) / 8

Linux (Ubuntu Recommended)

Mac OS X

For the JDK:

Oracle JDK 1.6 or above

For the Eclipse:

Eclipse 4.2 (Juno) or newer (Samsung SDK 4.5 or 5.0, This can be downloaded from **https://developer.samsung.com/smarttv/develop/extension-libraries/smart-view-sdk/download.html**)

For Run and Debug App:

Visual Box.

SDK Emulator Image for Virtual Box. This can be downloaded from **http://www.samsungdforum.com/Devtools/SdkArchive**.

Chrome browser

# Interfaces description

The following picture shows the User Interface of the Samsung Smart TV SDK IDE.

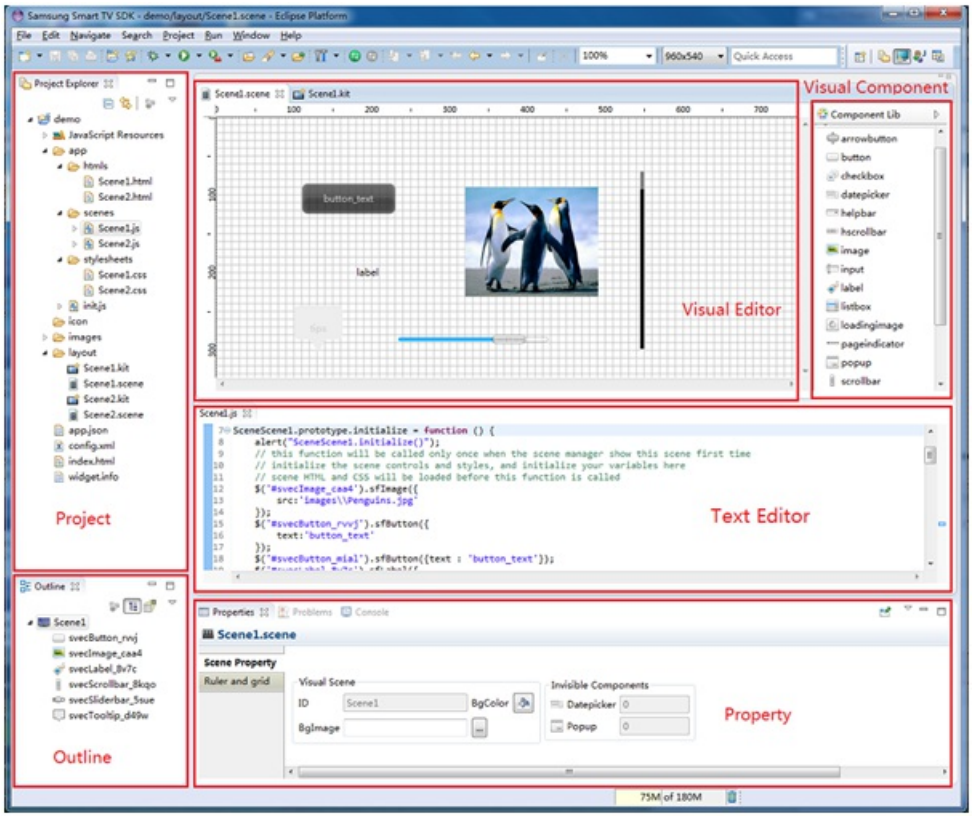



**Figure:** Eclipse SDK Editor Framework

The next picture shows the User Interface of the Visual Kit feature.

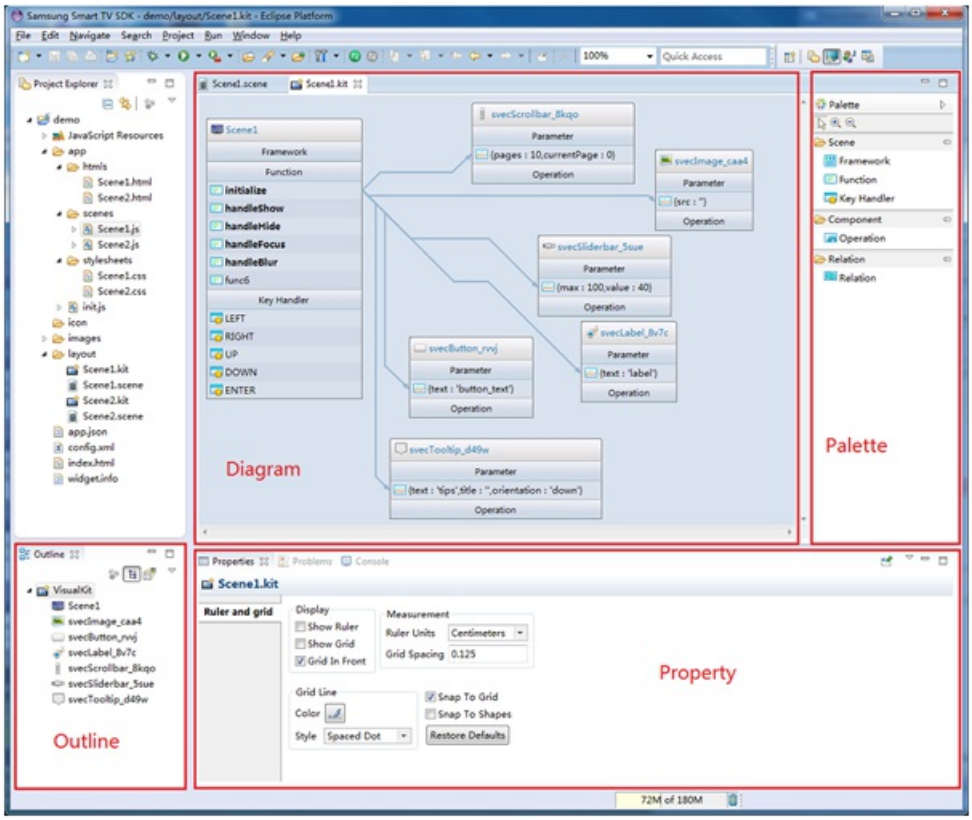



**Figure:** Eclipse Visual Kit Framework

# Tutorial application instructions

This session introduce a sample app to demonstrate the creation processes by using Visual Editor to draw the UI and using Visual Kit to build the relationship.

This App consists of two scenes, Scene1 and Scene2 with logical relation. Once this App is launched, Scene1 can switch to Scene2 by pressing the ENTER and Scene2 can turn back to Scene1 by pressing the RETURN.

Scene1 has a container, which contains two pictures: day and night. When the App receives the LEFT or RIGHT key event from a remote control, the container will switch among these two pictures.

## Create an AF 2.0 Project

1. After the Eclipse platform is launched, select the menu item File ▸ New ▸ Project ▸ Samsung Smart TV Apps.

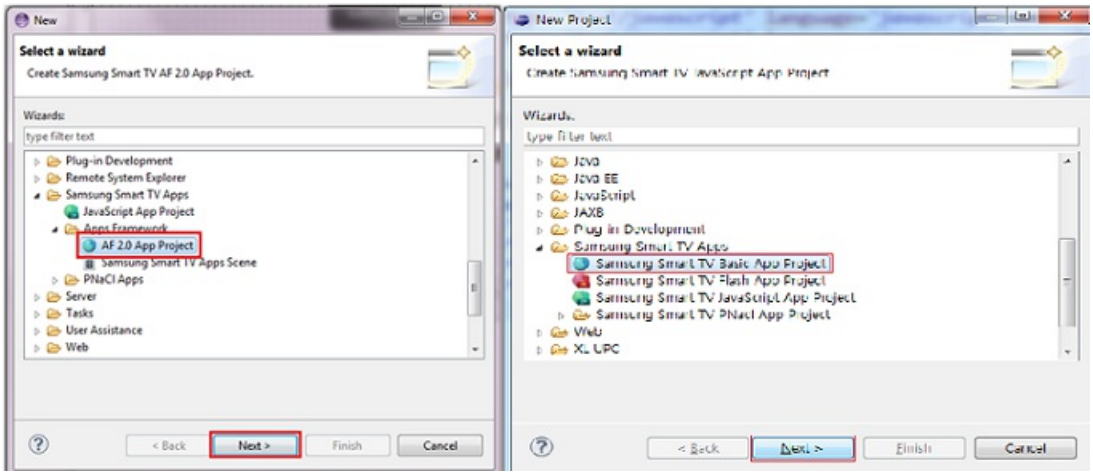



**Figure:** AF 2.0 Project Wizard Page 1 (Left : SDK 5.0, Right : SDK 4.5)

2. Select Apps Framework ▸ AF 2.0 App Project and then click Next to start the wizard.

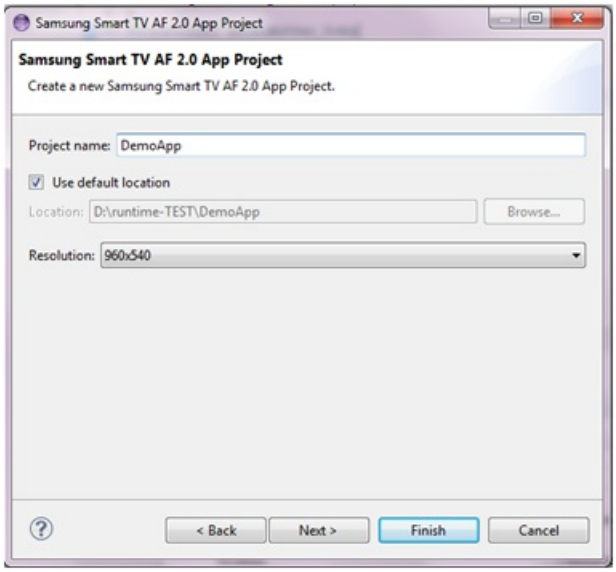



**Figure:** AF 2.0 Project Wizard Page 2

On this page: Type **DemoApp** in the Project name field. Select 960*540 in the Resolution field.

Then click Finish button.

3. In the Project Explorer, expand the **DemoApp** project

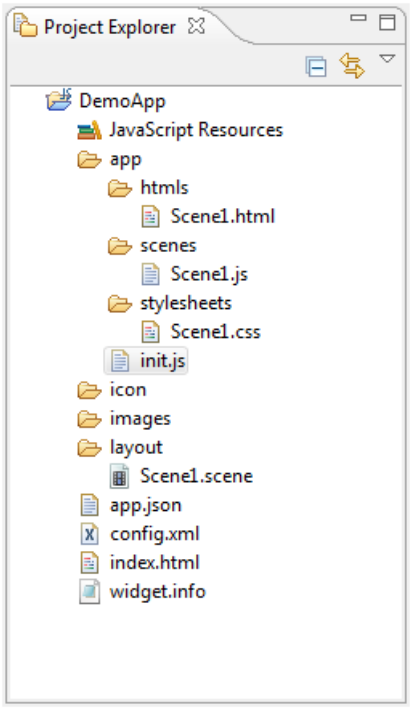



**Figure:** DemoApp project

4. Open Scene1.scene file with Scene Editor or double click Scene1.scene to open the scene.
5. Now, drag-and-drop or double click the component from **Component Lib** to create components for your App.

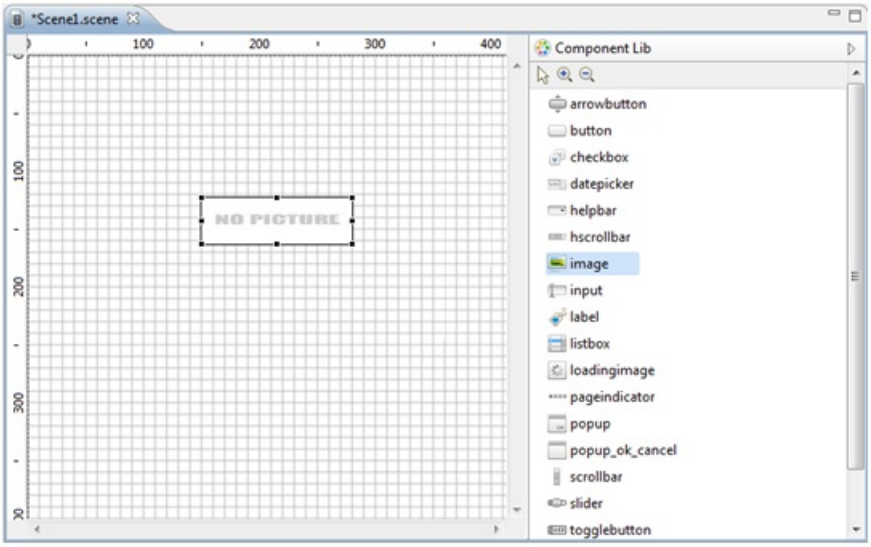



**Figure:** Create the components

## Create and Design Scenes

1. Add two buttons and one image components to Scene1 which is created by default when create an AF 2.0 app project.

   Drag an image component and two buttons to the scene from Palette View.

   Adjust the components' location and other properties, and the scene is shown as below.

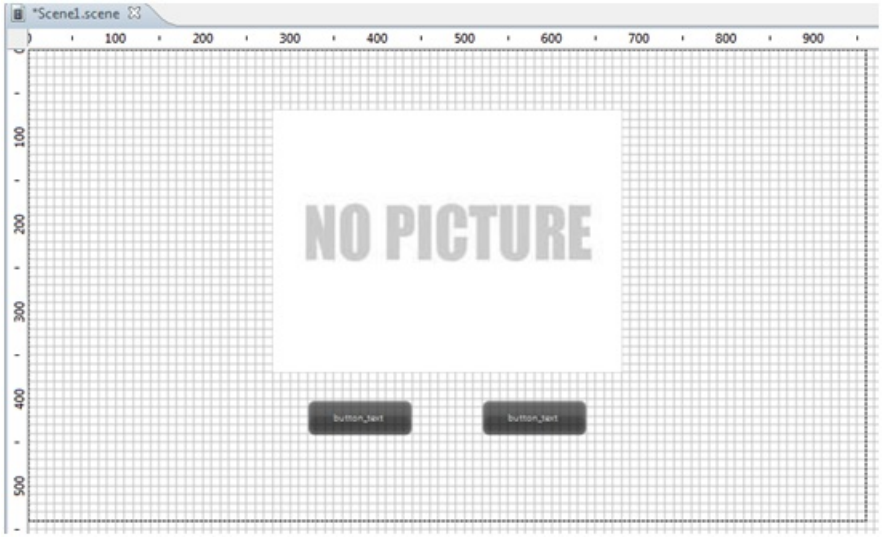



**Figure:** The Scene1

2. Add a new scene

   Choose the project tree or layout scene folder, popup the context menu. Choose New ▸ Samsung Smart TV Apps Scene to open the New Visual Scene Diagram wizard.

   In New Visual Scene Diagram wizard, rename the scene name, and click Finish button to add a new scene.

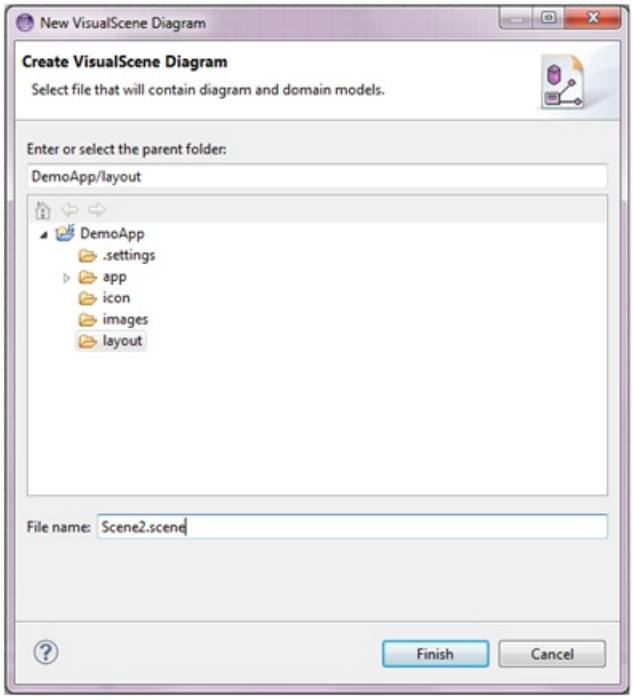



**Figure:** New Visual Scene Diagram wizard

3. Add an image, label and helpbar component for the Scene2:

   Drag an image, label and helpbar component to the scene from Palette View.

   Adjust the components' location and other property, the scene is shown as below.

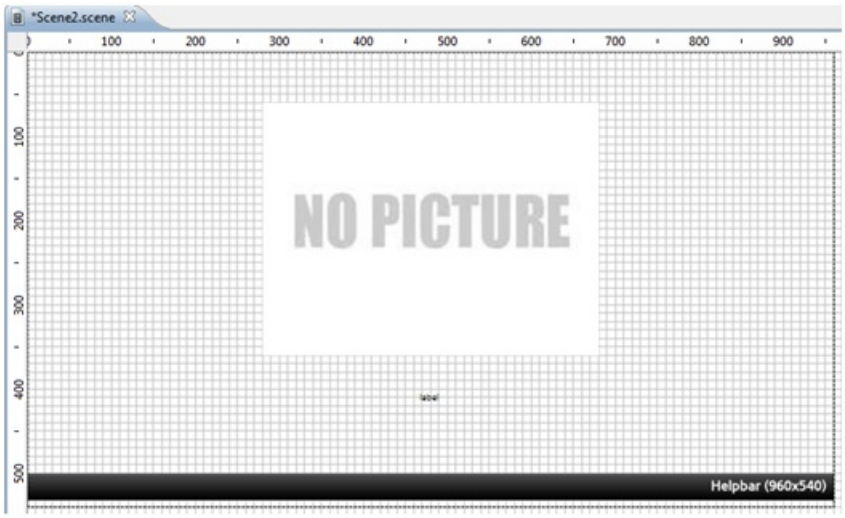



**Figure:** The Scene2

## Operations on Visual Kits

1. Create a Visual Kits. Choose the layout scene folder, popup the context menu. Choose New ▸ Samsung Smart TV Apps Kits to create the Visual Kits of the two scenes.

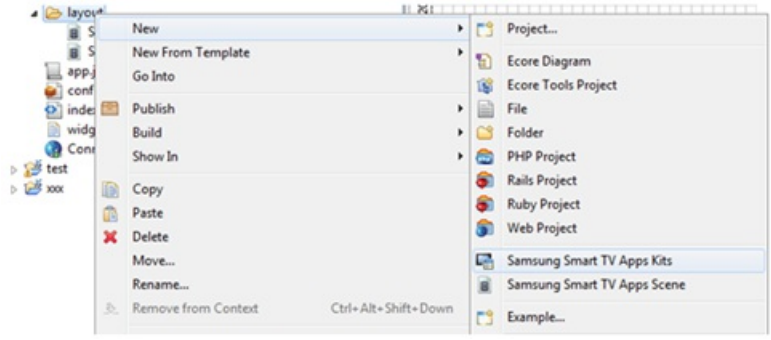



**Figure:** Create a Visual Kits

2. Add RETURN key in Scene2 and add Framework function in both scenes. The Framework functions are show, hide, and focus. Find the Palette and choose the Framework or Key Handler and click Framework or Key Handler block in scene to

add Framework or Key Handler. The image is shown as below.

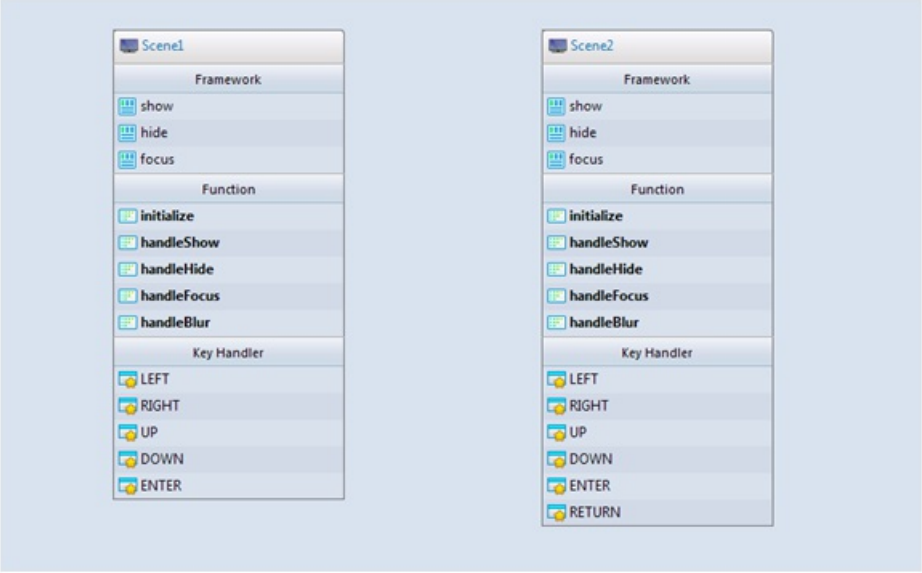



**Figure:** Show Visual Kits

3. Add the response code to ENTER and RETURN key. Add relation to two scenes to switch scene. Find the Palette and choose the Relation and click on block in a scene and drag to the needed block. For example, in Scene1 add relation between Enter key and Framework of hide, and add relation between ENTER in Scene1 and Framework of show, and focus in Scene2. Do the same operation of RETURN in Scene2. The image is shown as below.

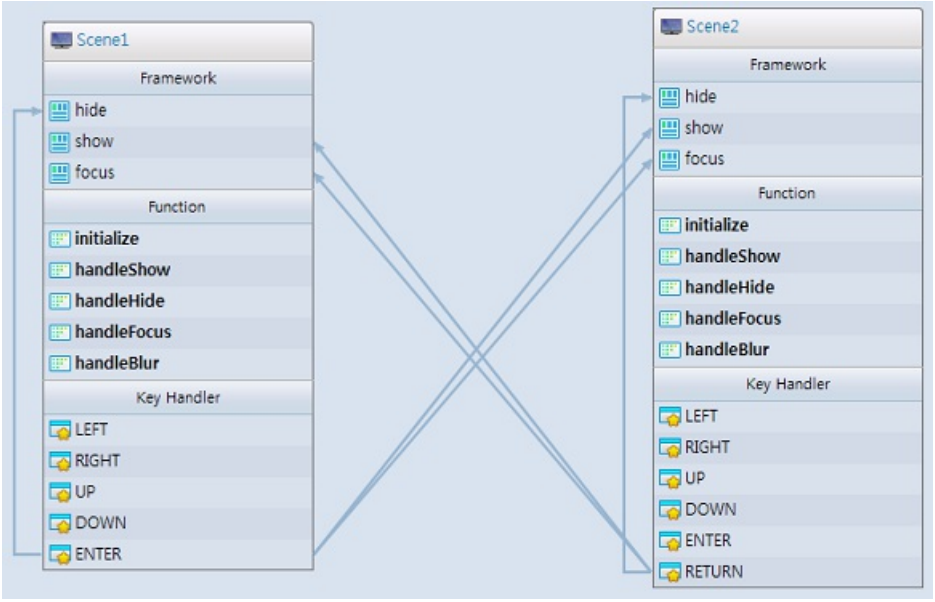



**Figure:** Add Relations on Visual Kits

The generated codes of the two scenes are shown as below.

```
SceneScene1.prototype.handleKeyDown = function (keyCode) {
    alert("SceneScene1.handleKeyDown(" + keyCode + ")");
    // TODO : write an key event handler when this scene get focued
    switch (keyCode) {
        case sf.key.ENTER:
            sf.scene.hide('Scene1');
            sf.scene.show('Scene2');
            sf.scene.focus('Scene2');
            break;
        ...
    }
}
SceneScene2.prototype.handleKeyDown = function (keyCode) {
    alert("SceneScene2.handleKeyDown(" + keyCode + ")");
    // TODO : write an key event handler when this scene get focued
    switch (keyCode) {
        case sf.key.RETURN:
            sf.scene.hide('Scene2');
            sf.scene.show('Scene1');
            sf.scene.focus('Scene1');
            break;
        ...
    }
}
```

Now, the smart TV will switch from the loading scene to the main scene automatically.

## Operations on Visual Kit

1. Create a Visual Kit. Choose a scene in layout scene folder, popup the context menu. Choose New ▸ Samsung Smart TV Apps Kit to create the Visual Kit of this scene. Do the same operation to the other scene.

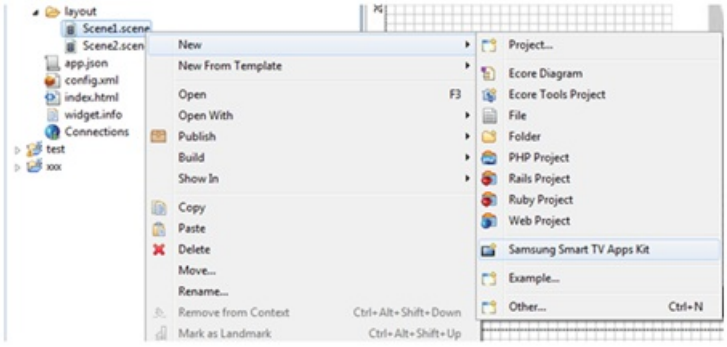



**Figure:** Create a Visual Kit

2. Add image path and change the button's text and so on. They can be changed in property view when click the needed block of component. And add a parameter to the image component and set the other image's path. Add needed operations to the button and image component. The image is shown as below.

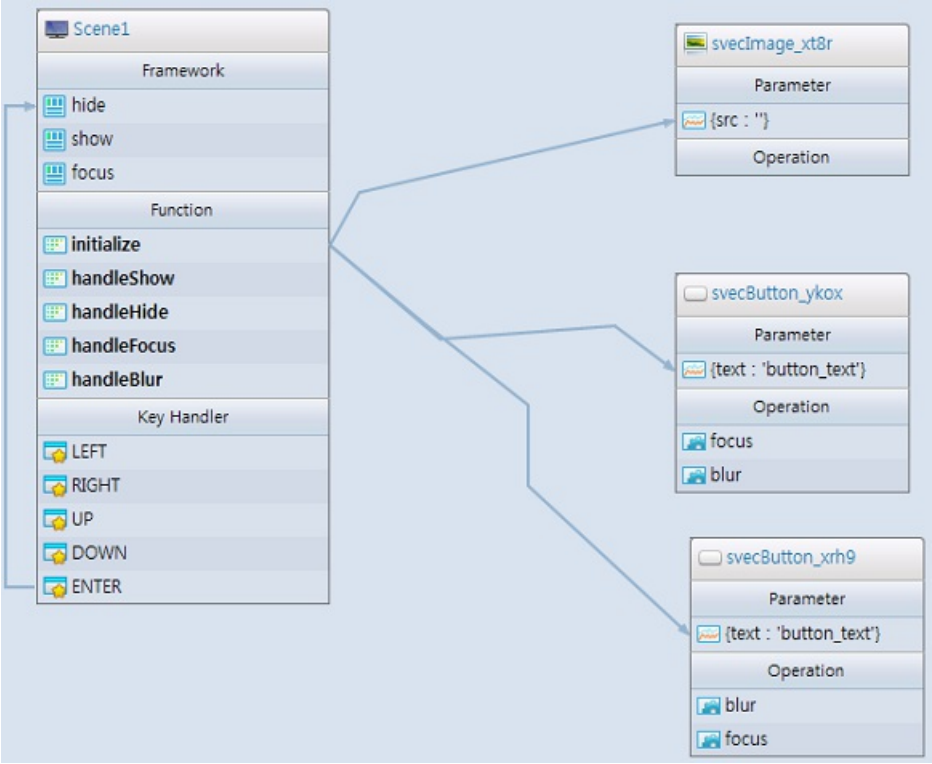



**Figure:** Visual Kit of Scene1 step 1

3. Then add Relations between LEFT, RIGHT keys and three components to add response to these two keys in Scene1. And add Relations between handleShow function and components in Scene2. The image is shown as below.

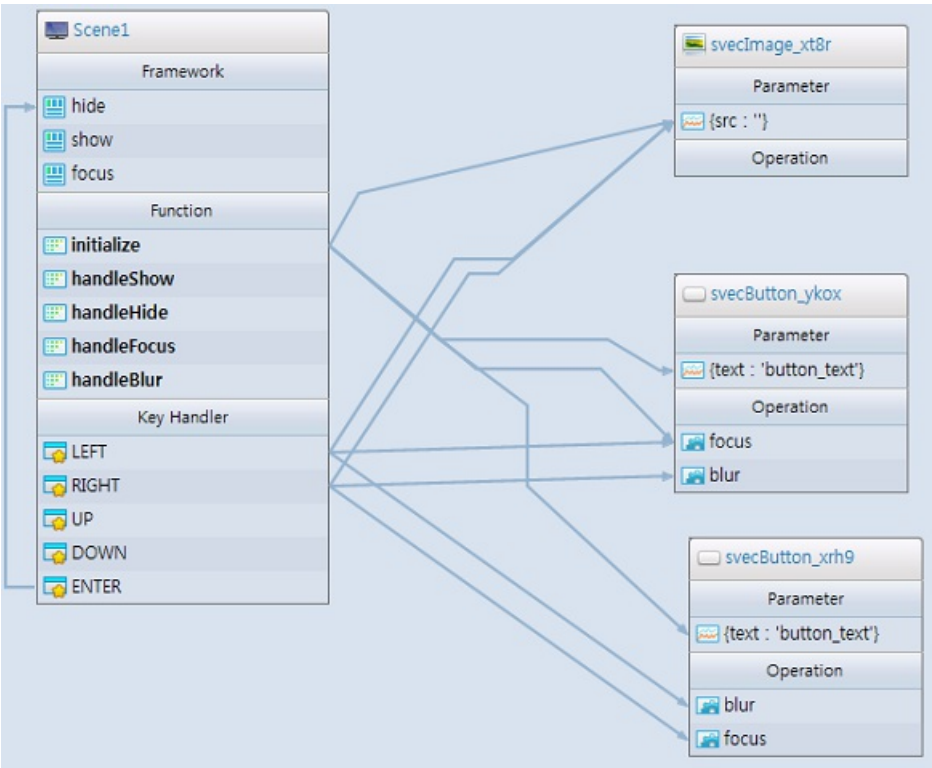



**Figure:** Visual Kit of Scene1 step 2

The generated codes: Scene1.

```
SceneScene1.prototype.handleKeyDown = function (keyCode) {
    alert("SceneScene1.handleKeyDown(" + keyCode + ")");
    // TODO : write an key event handler when this scene get focused
    switch (keyCode) {
        case sf.key.LEFT:
            $('#svecButton_5i7kfg86vl6xg').sfButton('blur');
            $('#svecButton_5i7kfg86toaxf').sfButton('focus');
            $('#svecImage_5i7kfg86dd7xe').sfImage({src:'images/day.png'});
            break;
        case sf.key.RIGHT:
            $('#svecButton_5i7kfg86toaxf').sfButton('blur');
            $('#svecButton_5i7kfg86vl6xg').sfButton('focus');
            $('#svecImage_5i7kfg86dd7xe').sfImage({src:'images/night.png'});
            break;
        case sf.key.UP:
            break;
        case sf.key.DOWN:
            break;
        case sf.key.ENTER:
            sf.scene.hide('Scene1');
            sf.scene.show('Scene2');
            sf.scene.focus('Scene2');
            break;
        default:
            alert("handle default key event, key code(" + keyCode + ")");
            break;
    }
};
```

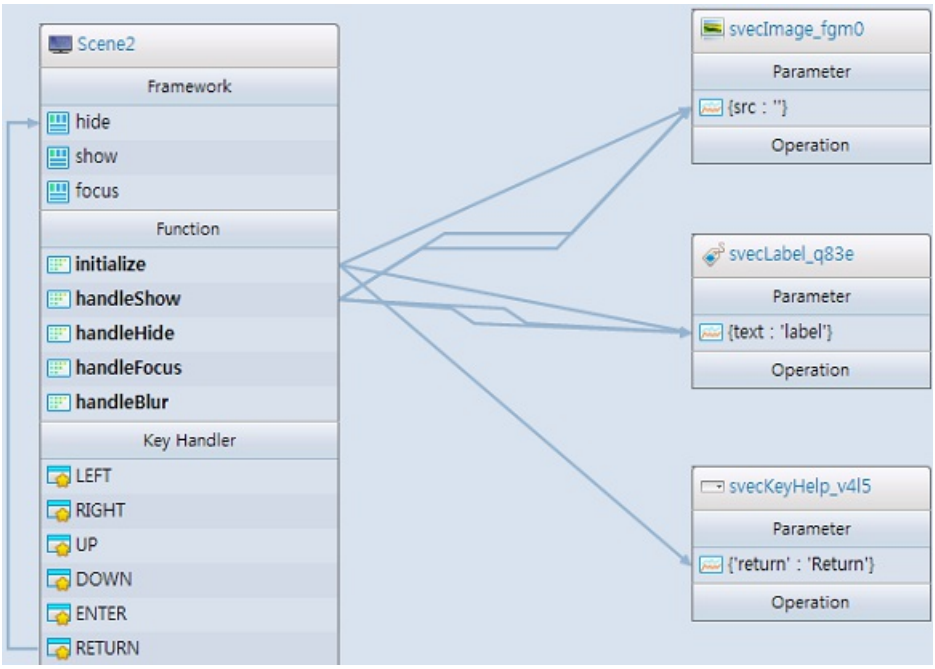



**Figure:** Visual Kit of Scene2

The generated codes: Scene2.

```
SceneScene2.prototype.handleShow  = function (data) {
    alert("SceneScene1.handleKeyDown()");
    $('#svecLabel_5i7kfg7p5soxc').sfLable(text:'Day');
    $('#sveclmage_5i7kfg7gho2xa').sflmage({src:'images/day.png'});
    $('#svecLabel_5i7kfg7p5soxc').sfLable(text:'Night');
    $('#sveclmage_5i7kfg7gho2xa').sflmage({src:'images/night.png'});
};
```

4. Add some logic code that cannot do by VisualKit in Scene1. Do the similar to the Scene2. The code is shown as below.

The codes in Scene1

```
SceneScene1.prototype.handleKeyDown = function (keyCode) {
    alert("SceneScene1.handleKeyDown(" + keyCode + ")");
    // TODO : write an key event handler when this scene get focued
    switch (keyCode) {
        case sf.key.LEFT:
            $('#svecButton_5i7kfg86vl6xg').sfButton('blur');
            $('#svecButton_5i7kfg86toaxf').sfButton('focus');
            $('#sveclmage_5i7kfg86dd7xe').sflmage({src:'images/day.png'});
            imageIndex = 0;
            break;
        case sf.key.RIGHT:
            $('#svecButton_5i7kfg86toaxf').sfButton('blur');
            $('#svecButton_5i7kfg86vl6xg').sfButton('focus');
            $('#sveclmage_5i7kfg86dd7xe').sflmage({src:'images/night.png'});
            imageIndex = 1;
            break;
        case sf.key.UP:
            break;
        case sf.key.DOWN:
            break;
        case sf.key.ENTER:
            sf.scene.hide('Scene1');
            sf.scene.show('Scene2',imageIndex);
            sf.scene.focus('Scene2');
            break;
        default:
            alert("handle default key event, key code(" + keyCode + ")");
            break;
    }
};
```

The codes in Scene2

```
SceneScene2.prototype.handleShow  = function (data) {
    alert("SceneScene1.handleKeyDown()");
    if (data == 0){
        $('#svecLabel_5i7kfg7p5soxc').sfLable(text:'Day');
        $('#sveclmage_5i7kfg7gho2xa').sflmage({src:'images/day.png'});
    } else if (data == 1){
        $('#svecLabel_5i7kfg7p5soxc').sfLable(text:'Night');
        $('#sveclmage_5i7kfg7gho2xa').sflmage({src:'images/night.png'});
    }
};
```

# App Framework 2.0 project development

The App Framework 2.0 is a Samsung provided framework which obeys the operation standards for Samsung Smart TV. The Smart TV IDE provides a visualization editing tools (such as Visual Editor) to helps developer to achieve a WYSIWYG (What You See Is What You Get) design. User can drag and drop some components to enrich their application. Currently, it has lots of integrated components to support WYSIWYG design.

Smart TV IDE also provides a high-level design method for developers, the Visual Kit. The visual kit diagram (*.kit) file shows the selected scene, components and the relations between them. Each *.kit file is responsible for one Scene. Therefore the *.kit file uses the same name as the *.scene file. The Visual Kits diagram (*.kits) file is the root for all the Kit files within the project. The Kits diagram contains all the scenes and their relationship information. By editing the visual kit/kits diagram, users can build logic relation for their apps.

## Project Overview

To create an AF 2.0 app project, user can click File in menu bar, select New ▸ Other in context menu. Choose the Samsung Smart TV Apps ▸ Apps Framework and expand it. Then select the AF 2.0 App Project to crate it. Samsung Smart TV AF 2.0 app project is composed of some resource folders, source code files and some configuration files. When finish creating an AF 2.0 app project, Samsung Smart TV SDK will generate the following resources:

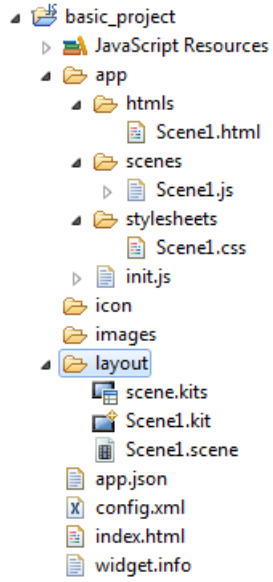



**Figure:** AF 2.0 app Project Overview

The detailed descriptions of the resources in the AF 2.0 app project are as below:

| File (Folder) | Description |
|---|---|
| app | The folder contains widget all source code files. |
| app/htmls | The folder contains widget .html source code files. |
| app/htmls/Scene1.html | The file is created by default when create a new AF 2.0 app project. |
| app/scenes | The folder contains widget .js source code files. |
| app/scenes/Scene1.js | The file is created by default when create a new AF 2.0 app project. |
| app/stylesheets | The folder contains widget .css source code files. |
| app/stylesheets/Scene1.css | The file is created by default when create a new AF 2.0 app project.. |
| app/init.js | The file includes initialization (onStart) and de-initialization (onDestroy) functions. onStart is entry point of application. |
| icon | The folder contains all icon files of the widget. |
| images | The folder is to store AF 2.0 app project components images. |
| layout | The folder is to store AF 2.0 app project scene file. |
| layout/Scene1.scene | The file is created by default when creating a new AF 2.0 app project to design widget. |

| File (Folder) | Description |
|---|---|
| layout/Scene1.kit | The file is created by users based on a already created *.scene file to build logic relation of scene and components. |
| layout/scene.kits | The file is created by users based on all the *.scene files to build logic relation of scenes. |
| app.json | This file is descriptor for Apps Framework. |
| config.xml | This file is descriptor of Smart TV application. |
| index.html | This is the main page of application. All elements from Apps Framework are added to this file. |
| widget.info | This file is to configuration widget information. |

Samsung Smart TV IDE also provides three types of template. For each template, there are three themes with different colors available for users to choose. These templates show the fundamental approaches to develop Smart TV applications by using App Framework 2.0 such as components, scene manager, and event handlers. If developers want to use the template, click the Use AF 2.0 app project template checkbox and select one template during the creation process. Otherwise directly click the Finish button to create an empty AF 2.0 app project.

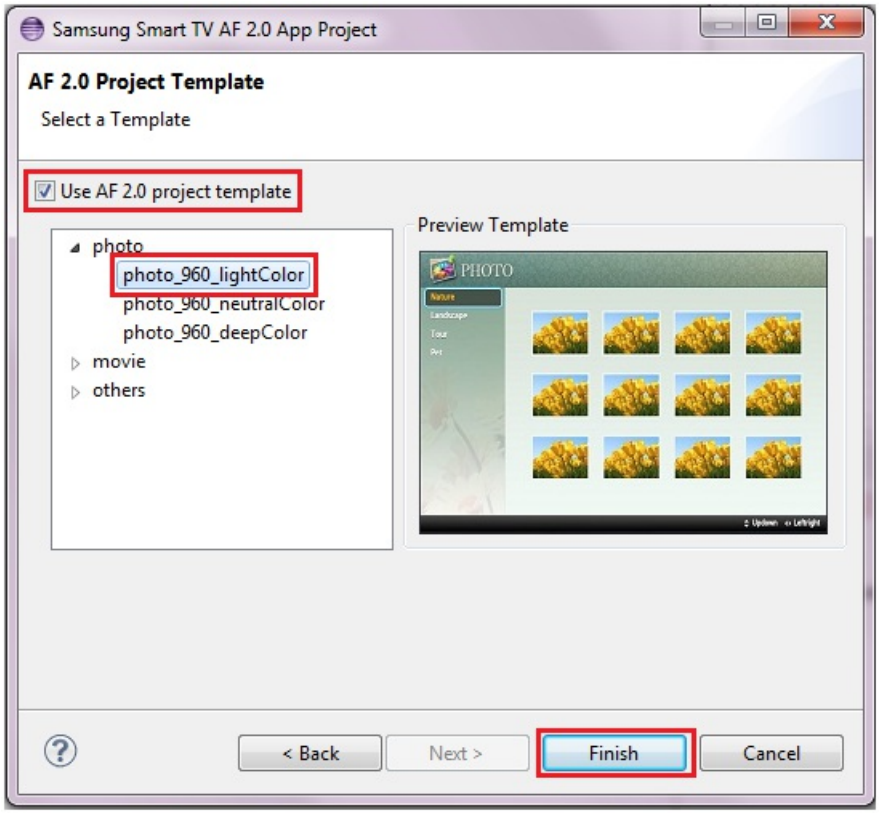
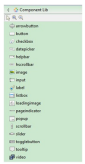



**Figure:** AF 2.0 app Template Project

## Palette View

Visual Editor: Component **Lib View** provides the following palette for creating Samsung Smart TV applications.

| Overview | Style | Description |
|---|---|---|
|  | arrow button | The arrow button is the most basic component used in every design as we know. It is used to handle all the events that should happen on the click of a button. The arrow button component that is provided by the Samsung TV SDK can be extended horizontally and the text property of the arrow button can be modified. |
| | button | The button is the most basic component used in every design as we know. It is used to handle all the events that should happen on the click of a button. The button component that is provided by the Samsung TV SDK can be extended horizontally and the text property of the button can be modified. |
| | checkbox | The checkbox is a component used to typically select options. In Samsung TV SDK, it can be used to select a single option or multiple options depending on the developer's requirement. The size of the checkbox cannot be extended horizontally and vertically. |
| | | |

| Overview | Style | Description |
|---|---|---|
| | datepicker | Samsung TV SDK provides the date picker component. It makes the task for entering the dates and times in the application easier. |
| | helpbar | Helpbar gives information about the function of each button on the remote control. |
| | hscrollbar | This component is typically used when the size of the page exceeds the size of the screen. You can set the size of the bar by using the size of the entire page, to the size of the screen. |
| | image | It is used to place an image in the design. The size of the image can be specified. |
| | input | The input component is used to get text input from the user. The component can be added using the Samsung TV SDK. It is quite different from that of the other components. It has two parts. The text box and the number pad component. The number pad can be used to type in text into the text box. |
| | label | It is used to display text on the screen. |
| | listbox | List box is used to manage the data items in the form of a list. You can set the number of items in the list and add an event handler for each entry in the list. |
| | loading image | Loading image component can be used when a task is currently loading or getting ready to be displayed on the screen. The usage of this component is absolutely a developer's choice and left up to his style of coding. However, good care has to be taken when the component is used in the right time and only for a specified amount of time. For example, in a video stream application, this component can be used along with the timer functions in JavaScript only when there are pauses in streaming. |
| | page indicator | Page indicator component is used to display a series of dots to indicate which page is currently being viewed. |
| | popup | Samsung TV SDK provides two types of popup components. They are single button popup and two buttons popup. When a popup is added to the design editor, the popup does not appear on the Samsung TV SDK. |
| | scrollbar | This component is typically used when the size of the page exceeds the size of the screen. You can set the size of the bar by using the size of the entire page, to the size of the screen. |
| | slider | This component is used to allow changing a numeric value by touching and dragging a slide handle. |
| | toggle button | The toggle button is a component that can toggle between two states. |
| | tooltip | The tooltip is a simple popup widget that encloses its content and displays a small arrow associating the content with a node. |
| | video | The Samsung TV SDK provides the Video Component to insert Videos into the Design. Often, while designing multi-media applications, developer would need such components more than the basic components. Hence Samsung TV SDK provides a very good platform for developing multi-media applications. |

Visual Kit: The Visual Kit and Kits files are design tools aiming to provide an easy way for developers to create their applications. Once a *.kit or a *.kits file is opened, a palette appears at the right side of the Visual Kit diagram which contains the major functions of visual kit. By selecting the elements from the palette and inserting it into the active Scene or Component part in Visual Kit diagram, the program can generate the code automatically into the *.js file. The code generation scenario will be listed in the Synchronize Section.

| Overview | Style | Description |
|---|---|---|
|  | Framework | Developer can select **Framework** from palette and insert it into Scene object in the left diagram. After that, a new element will be added in the Framework section. The name of the newly added element is depends on App Framework (Ver2.0). Framework element can only be inserted into Scene object. Adding a Framework element will not cause any source code change. |
| | Function | Developer can select **Function** from palette and insert in into Scene object in the left diagram. After that, a new element will be created in the Function section. The name of the newly added element is formatted as func + number, for example func8, the number is the total count of function elements in that Scene object. If the formatted name exists yet, count number will plus one. In text editor, a piece of JavaScript code who defined a new function with the formatted name will generate. |
|  | Key Handler | Developer can select **Key Handler** from palette and insert it into Scene object in the left diagram. After that, a new element will be created in the Key Handler section. The name of newly added element can be found on those keys on TV remote control. In text editor, a piece of JavaScript code of key handle case will be generated. |

| Overview | Style | Description |
|---|---|---|
| | Operation | Developer can select **Operation** from palette and insert it into Component object in the left diagram. After that, a new element will created in the Operation section. The name of newly created element can be found in Component guide. Adding an Operation element will not cause any source code change. |
| | Relation | Developer can select **Relation** from palette then drag and drop it from element to element. After that, a new line with arrow will be created between the two elements. At the same time, one line of source code would be generated. |

## Property View

Developers can use the property view to change the property value. The change will be reflected in other places where the change is used. If modifying the property value in source code, the changes will also be updated in the property view.

Sometimes, the synchronization will be done when user trigger the synchronization action (Sync all: Sync current: ) via toolbar. In addition, if developers change some properties of an item in the property view, the properties will be checked whether the value is available for that item or not. If the value is not suitable, warning information will be displayed on the top of the property view, and the previous value of the item will be remained.

### Visual Editor

### Component Property

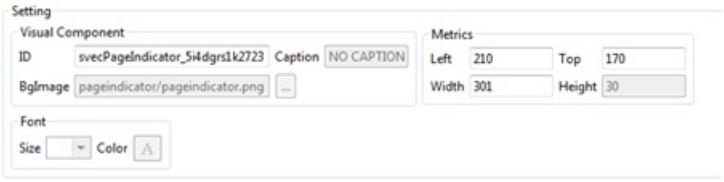



**Figure:** Component Property

| Property | Description |
|---|---|
| ID svecPageIndicator_5i4dgrs1k2723 | Show the component's **ID**, if modified, it would be synchronous related to code. |
| Caption NO CAPTION | Show the component's **caption**, if modified, it would be synchronous related to code. |
| BgImage pageindicator/pageindicator.png ... | Show the component's **background image** path, if modified, it would be synchronous related to code. |
| Left 210 Top 170 Width 301 Height 30 | Show the component's metric, if modified, it would be synchronous related to code. |
| Size Color A | Show the component's font, if modified, it would be synchronous related to code. |

### Component Parameter

| Property | Value |
|---|---|
| timePicker | false |
| format | yyyy-MM-dd |
| callback | function |

**Figure:** Component Parameter

Description: Show the component's JavaScript parameters. The code is loaded when create the component via Visual Editor.

### Scene Property

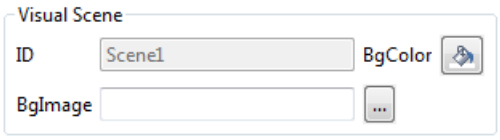



**Figure:** Scene Property

| Property | Description |
|---|---|
| ID Scene1 | Show the scene's **ID**, if modified, it would be synchronous related to code. |
| BgColor | Show the scene's **background color**, if modified, it would be synchronous related to code. |
| BgImage ... | Show the scene's **background image**, if modified, it would be synchronous related to code. |
| Invisible Components | Show the invisible components in diagram, and their numbers. |

### Ruler and Grid

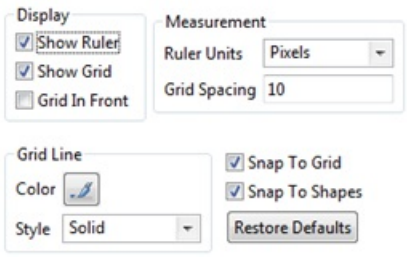



**Figure:** Ruler and Grid

| Property | Description |
|---|---|
| Display<br>Show Ruler<br>Show Grid<br>Grid In Front | **Show Ruler** - Display a ruler along the left and top edges of the diagram and provides access to ruler guides. Guides are added by clicking anywhere in the ruler and are removed by pressing Delete when active.<br><br>**Show Grid** - Display the grids and used for alignment.<br><br>**Grid in Front** - Show the grid in front of the visual component. |
| Measurement<br>Ruler Units Pixels<br>Grid Spacing 10 | **Ruler Units** - Set the units of the ruler.<br><br>**Grid Spacing** - Set the space of the grid. |
| Grid Line<br>Color<br>Style Solid | **Color** - Set the color of the grid line.<br><br>**Style** - Set the style of the grid line. |
| Snap To Shapes | **Snap To Shape** - This is a useful feature for alignment. A "laser line" will be appeared at the edges or the center of the node which assist users to arrange the components in the canvas. |
| Snap To Grid | **Snap To Grid** - This is another useful feature for alignment. This feature achieve an automatic snapping function when users move one component close to another one. |
| Restore Defaults | **Restore Defaults** - Reload the settings of the ruler and grid tab. |

### Visual Kit

All the elements in Visual Kit diagram remain their own property view page. The property views are different from each other. Users can modify the code in *.js file via editing the value in property view. Meanwhile, the code changes in the *.js file also can be illustrated on the Visual Kit diagram by executing the synchronization for the Kit file.

### Scene Property

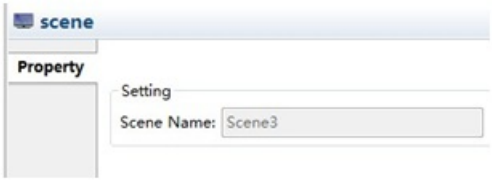



**Figure:** Scene Property

Description: The property page for the scene contains only one setting, the scene name. This scene name option uses the file name as a default setting which cannot be replaced.

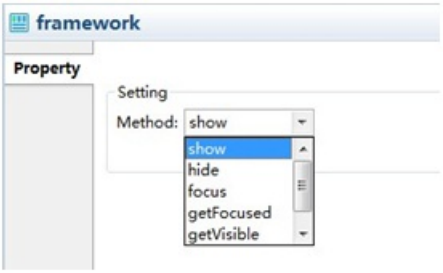



**Figure:** Framework Property

Description: The value of the framework is selectable from a combo box. There are six choices available in the framework property: show, hide, focus, getFocused, getVisible, getState.

### Function Property

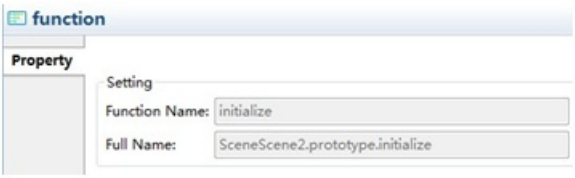



**Figure:** : Function Property

Description: There are two types of function. One type of function is generated initially by the system which cannot be deleted or renamed from the Visual Kit diagram. There are initialize, handleShow, handleHide, handleFocus and handleBlur. The property view shows the function name and the full name in *.js file. The other functions created by users are using the same property view. The only difference is the user created functions are supporting rename action.

### Key handler Property

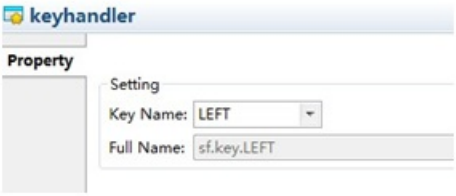



**Figure:**: Key handler Property

Description: Each Key Handler element represents a button in a remote controller. Therefore the number and the name of the Key Handler element are structured in the Visual Kit editor. There are 39 options available including 5 initially created for users to implement the Key Handler elements. All the options of the key name are listed in a combo box in property page. The full name of the key in *.js file is also shown in property page.

### Component Property

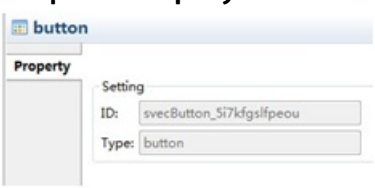



**Figure:**: Component Property

Description: The component property page presents two pieces of information. The first one is the id code of the component which obeys the information from Scene file and is generated by the system. The second one is the type of the component which is defined during the creation process. Both of these two settings cannot be modified.

### Parameter Property

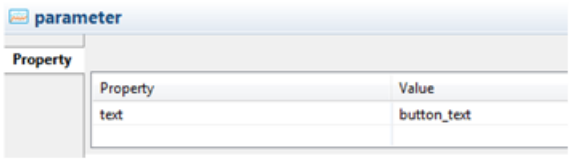



**Figure:**: Parameter Property

Description: Majority of the components have the parameter option. It shows the default property and value of current both of Property and Value can't modified by users. Figure Parameter Property shows the property view of a Button component and its default value is button_text.

### Operation Property

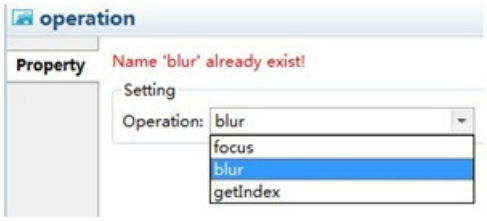



**Figure:**: Operation Property

Description: Most of the components provided by the system lib have operation elements. The Operation option describes the basic actions of the component that can be activated in the application. Each type of component has its own behaviors established in the system lib. Therefore the operation's value only can be picked and only one time picked from a particular combo box.

### Relation Property

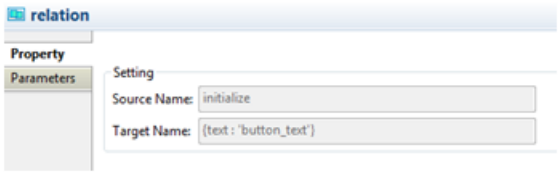



**Figure:**: Relation Property

Description: The relation items also contain their own property view. There are two tabs in relation property; They are Property and Parameters. The content in tab Property view shows two elements. The first one is the Source Name and the second one is the Target Name. Both of them can't be modified. Relation tab Parameters shows the Name and Value of Relation parameter. The Value can be modified by users.

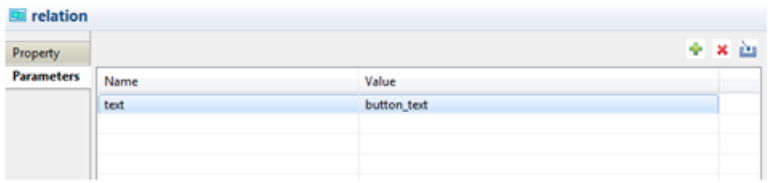



**Figure:**: Relation Parameters

On the top right of the Parameters view are three toolbars, Add, Delete, and Insert. They can be also trigger by right click. When click Add, a dialog will popup. When add parameter to the relation which target is Parameter, the dialog is as shown below. User can add Name by itself or click the comb box to select a suggested item. When relation target is Function, Framework or Option, the Name is fixed and user can only input Value. The Insert is the same as Add operation except that user must select an existed property line to trigger it. Select an exist property and click Delete to delete the selected property.

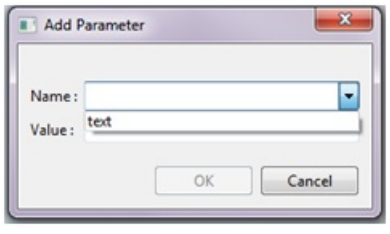



**Figure:**: Relation Add Parameters (Relation to Parameter)

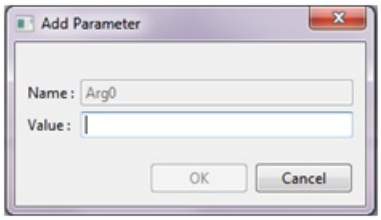



**Figure:**: Relation Add Parameters (Relation to Function, Framework, Operation)

## Context Menu

**Visual Editor**: The **Context Menu** is a popup menu when user right clicks on the Visual Editor. This menu provides some common used functions of the Visual Editor.

The screen shot of the context menu is shown below:

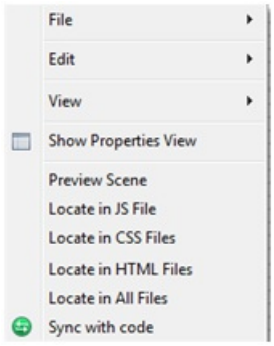
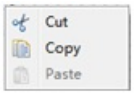
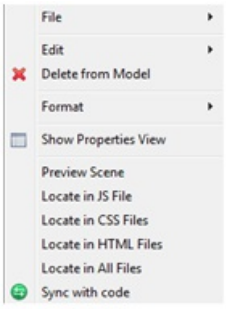
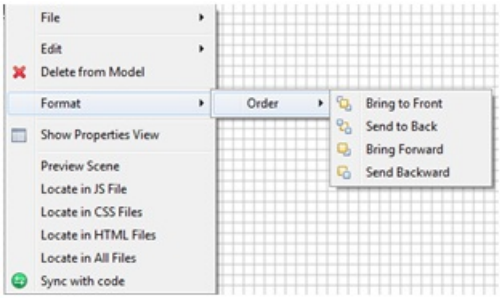



**Figure:** VE Context Menu

The details of **Context Menu** are listed as follows:

| Menu Name | Description |
|---|---|
| File | It contains the following submenu:<br>  <br><br>There are two operations available here.<br>The Save As Image File... operation can save the current scene's appearance as an image file.<br>The Print... operation allows user to print a picture of the current scene's appearance. |
| Edit | It contains the following submenu:<br>  <br><br>It provides cut, copy and paste operations for selected components.<br>Cut can cut the selected components from current scene.<br>Copy can copy the selected components from current scene.<br>Paste can paste the components which are copied or cut. |
| View | This operation is only provided for Visual Scene. It contains the following submenu:<br>  <br><br>Grid please refers to Property View<br>Rulers please refer to Property View.<br>Page Breaks can make a page break on the scene.<br>Recalculate Page Breaks can recalculate page breaks.<br>Snap to Grid please refer to Property View. |
| | |

| Menu Name | Description |
|---|---|
| Delete from Model | This operation will appear when component is selected in Visual Scene.<br>  <br>By clicking it, the selected component will be deleted. |
| Show Properties View | This operation makes Property View to be active. |
| Preview Scene | Preview the current scene in the Emulator. |
| Format | The hierarchy of its menu is showed as follows:<br>  <br>It provides functions to change the relative z-order of the selected components.<br>Bring component to the front<br>Send component to the back<br>Bring component to the forward<br>Send component to the backward |
| Locate in JS File | This operation will open the related JavaScript file and locate the selected component in JavaScript file. |
| Locate in CSS File | This operation will open the related CSS file and locate the selected component in CSS file. |
| Locate in HTML File | This operation will open the related HTML file and locate the selected component in HTML file. |
| Locate in All Files | This operation will open the related JavaScript, CSS and HTML files and locate the selected component in these files. |
| Sync with code | Synchronize the Visual Scene file with the code from JavaScript, CSS and HTML files. For more details, please refer to: Synchronize with Code. |

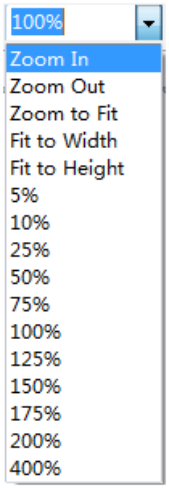
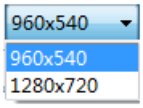
**Visual Kit**: The **Context Menu** is a popup menu when user right clicks on the Visual Kit diagram. This menu provides some common used functions of the Visual Kit. Depend on the requirement for each elements, the content of the context menu is different.


**Figure:**: VK Context Menu

| Menu Name | Description |
|---|---|
| Locate in JS File | This operation will open the related JavaScript file. |
| Sync with code | Synchronize the Visual Scene file with the code from JavaScript, CSS and HTML files. For more details, please refer to: Synchronize. |
| Show Properties View | This operation opens the Property View page. |

## Toolbar

The toolbar provides a convenient way to do some actions such as debug, package, sync with code, sync with all codes, change resolution, zoom in/out and so on.

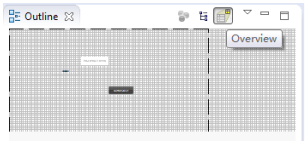
| Icon | Command Name | Description |
|---|---|---|
|  | Samsung Tools | Execute the Run PNaCl App in Chrome command if click on the tool icon, or open a drop-down menu if click on the triangular arrow. |
|  | Run PNaCl App in Chrome | Pop-up a dialog, developers can start the PNaCl Server and run the PNaCl App with Chrome browser from this dialog. |
|  | TV Log On/Off | Turn on/off the TV log. |
|  | Debug App | Debug a selected product into Samsung Smart TV Emulator. See Run and Debug for more. |
|  | Run App | Run a selected product into Samsung Smart TV Emulator. See Run and Debug for more. |
|  | Sync with code | Synchronize the current scene's JavaScript/HTML/CSS codes. |
|  | Sync all with code | Synchronize current project's all scenes's JavaScript/HTML/CSS codes. |
| | Zoom In/Out | Provides zoom in/out operations. The following selections are available:<br>  |
| | Change Resolution | Provides change project's resolution operations. The following resolutions are provided<br>  |

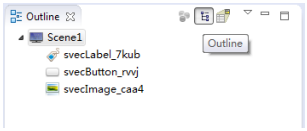
## Outline

The **Outline** provides a convenient way to do some actions such as list all components in current scene/kit (including special components), select component, preview scene, thumbnail current scene and so on.

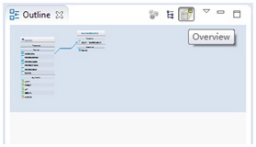
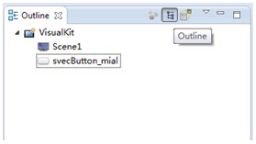
The detailed functions of the **Outline** are as below.

### Visual Editor

| Outline | Description |
|---|---|
|  | This overview option shows a thumbnail of the current active kit.<br><br>If the display scale of the current active kit is high, users can move the display interface by clicked or dragged the viewing zone from one place to another in the overview. |
| | |

| Outline | Description |
| --- | --- |
|  | The outline view shows the structure of current active scene using tree model, and provides the following operations:<br>1. right click the root node, it shows **Preview Scene** operations<br>2. click sub node, the component will be selected in current scene, all the opened related codes will be highlight in JavaScript, CSS and HTML editors, and the Properties view will be changed to the component, too. |

**Visual Kit**

| Outline | Description |
| --- | --- |
|  | This overview option shows a thumbnail of the current active kit.<br>If the display scale of the current active kit is high, users can move the display interface by clicked or dragged the viewing zone from one place to another in the overview. |
|  | The outline view shows the structure of current actived scene using tree model, and provides the following operations:<br>click sub node, the component will be selected in current kit, all the opened related codes will be highlight in JavaScript, CSS and HTML editors, and the Properties view will be changed to the component, too. |

## Synchronize with Code

Visual Editor and Visual Kit **Synchronize with code** function provide a convenient way to reflect code modification to Visual Scene/Kit/Kits. Now, our SDK gives the follow ways to do synchronize:

Through Toolbar

Through project resource and layout folder resource

Through scene file's Context Menu

Here gives an example of adding a button from code.

1. Add components' js codes in *.js file.

```
$('#svecButton_5i4dgcah6mqf').sfButton({
   text: 'button_text'
});
```

2. components' css codes in *.css file

```
#SceneScene1 #svecButton_5i4dgcah6mqf {
   position: absolute;
   left: 410px;
   top: 150px;
   width: 121px;
}
```

3. Add components' html codes in *.html file.

```
<div id="svecButton_5i4dgcah6mqf"></div>
```

When finishing synchronization, a new button component will be created on the scene.

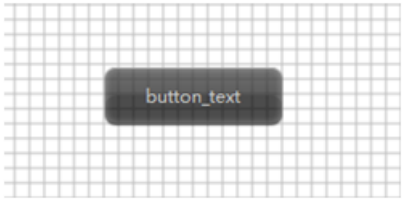



**Figure:** Synchronize result

# Samsung API Content Assist

Samsung API Content Assist, which has been integrated into **Samsung Smart TV SDK JavaScript Editor**, provides context sensitive Samsung Device API content completion upon user request when developing Samsung Smart TV apps.

Popup windows are used to propose suggestions for developers to choose from to complete a phrase.

Open a JavaScript file with **Samsung Smart TV SDK JavaScript Editor**, and then the code completion feature can be triggered by entering the objects defined in Samsung device API and call Content Assist (Alt + /, by default, or ”.”).

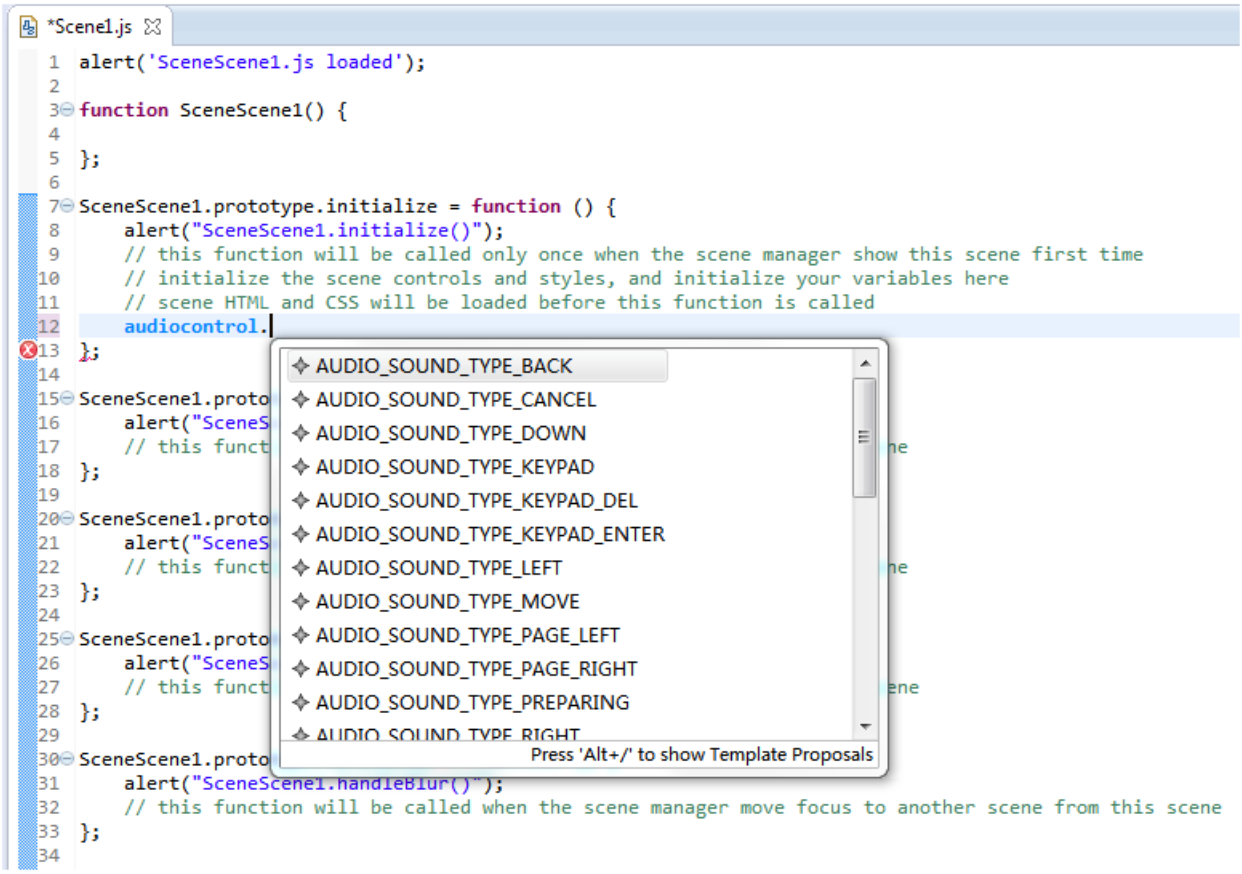



**Figure:** Content Assist of Samsung API Object

1. The developer’s previous input will be parsed, and certain related proposals will be presented in a popup window right after the ”.” character.
2. The proposals will be listed alphabetically for searching convenience.
3. Developer can select an appropriate proposal from the popup window using the up and down key, and apply the selected one by pressing Entry key.

# Run and Debug App (SDK 5.0)

Samsung Smart TV SDK not only support launching the apps into Emulator for functionality test, but also provides debugging function to developer. Samsung Smart TV SDK uses Web Inspector as its debugging tool which allows developers to view the page source, network status, script debugging, profiling and more.

After the preparation of the developing tools and installing the SDK Emulator Image into Visual Box, user should check the Visual Box setting and choose one Emulator for testing. Open the Samsung Smart TV Emulator preference page by Window ▸ Preferences ▸ Samsung Smart TV ▸ Emulator.

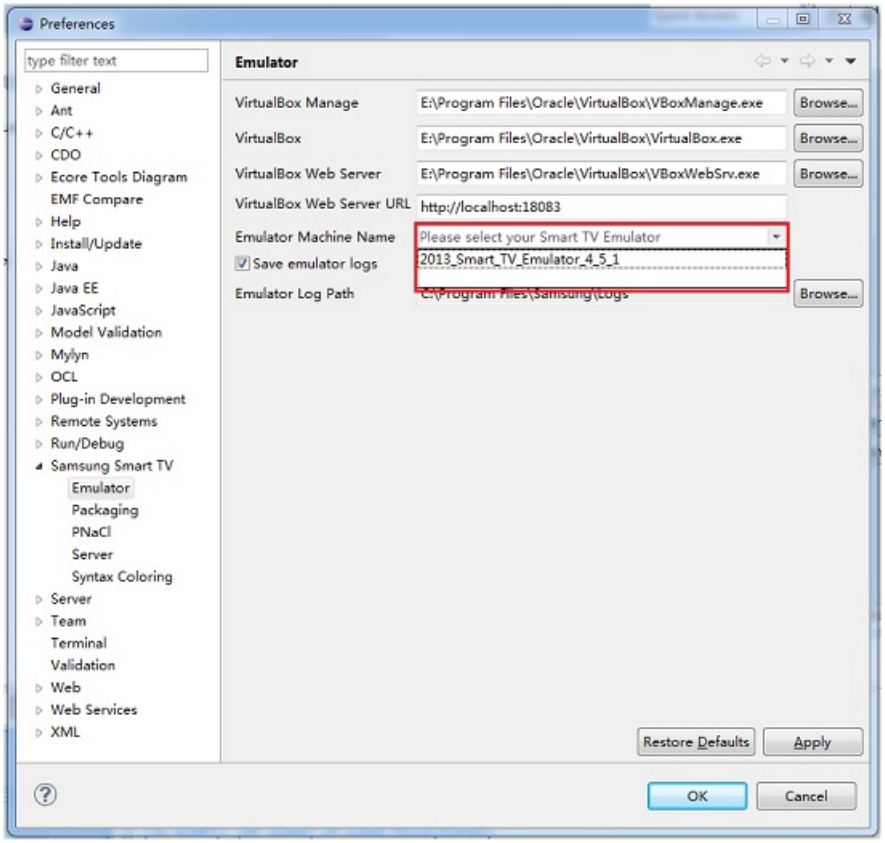



**Figure:** Emulator Preference Page

User can modify the Preferences to change some properties of the Emulator. If required to change the port of Virtual Box Web Server, user can edit the Virtual Box Web Server URL property. For example, changing the value to "http://localhost:9999", the Virtual Box Web Server will use the 9999 port rather than the default value 18083.

## Run and Debug

Developers have two ways to run or debug their apps:

1. Select one app project, and then click the Run or Debug button in toolbar.

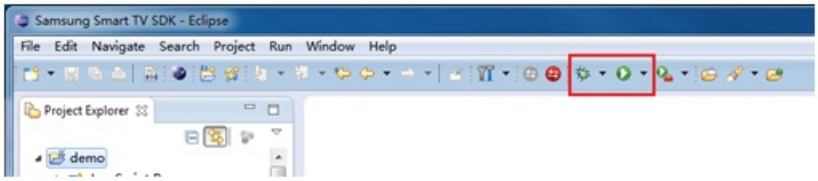



**Figure:** Run or Debug from Toolbar

2. Select one app project, right click it and within the pop-up project menu, find the Run As or Debug As button.

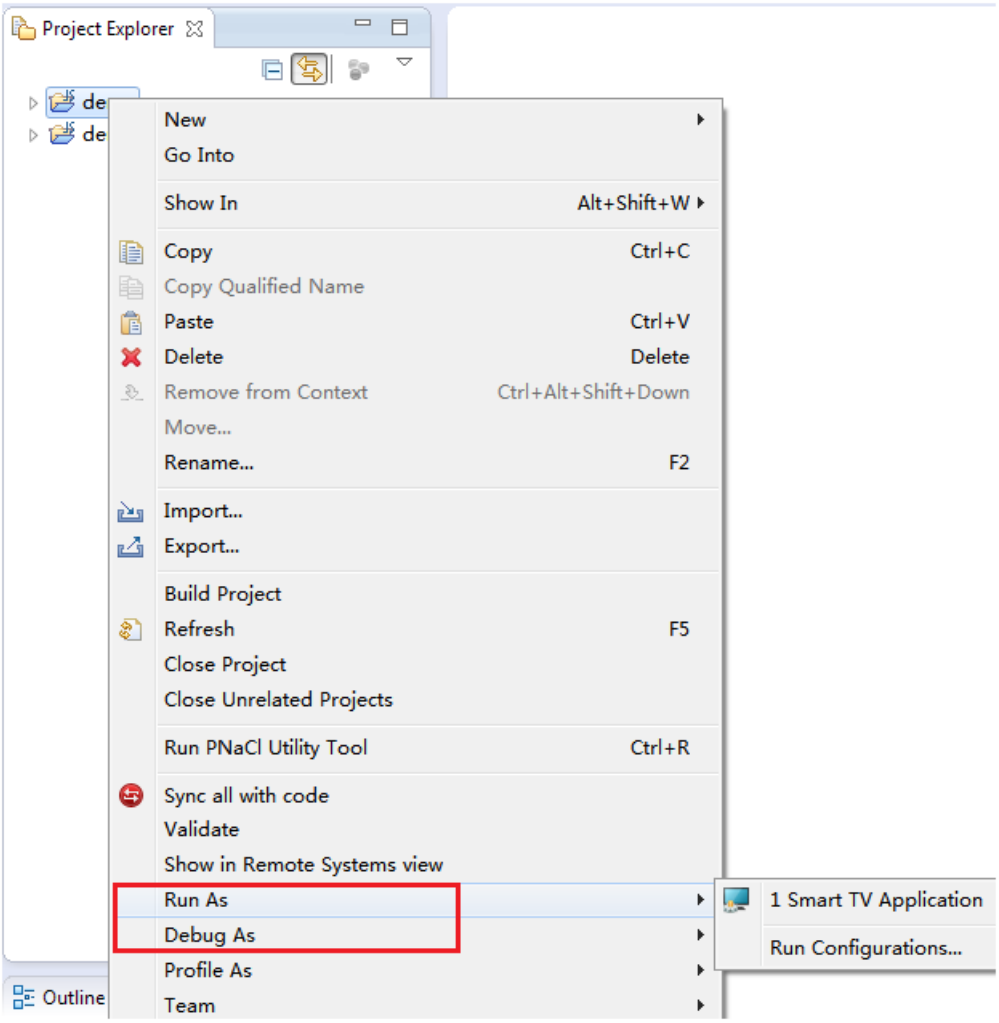



**Figure:** Run or Debug from Popup Menu

Google Chrome is a prerequisite for debugging apps, because the Web Inspector is integrated into it. Once executing the debugging function, the Web Inspector will wait until Emulator launching finished. If the emulator runs correct, the web inspector will be started soon. Developer can use it to debug JavaScript code, view page source and more.

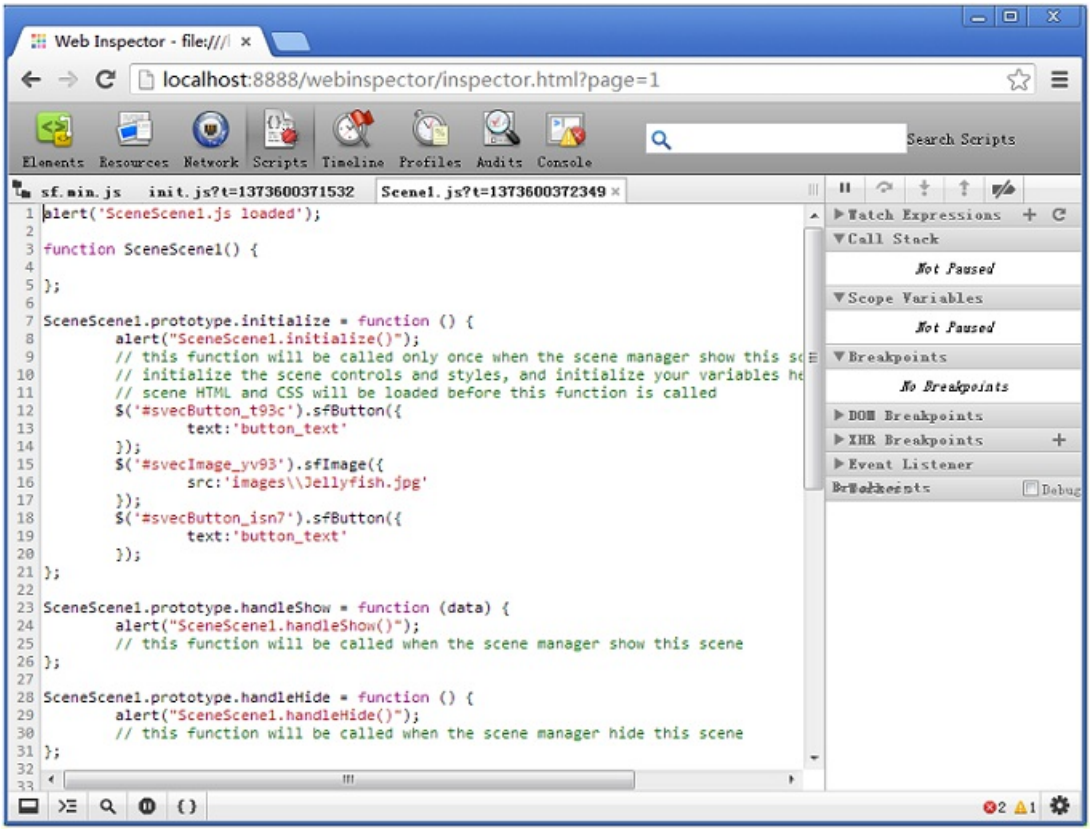



**Figure:** Web Inspector

Except the two methods above, right click on one scene and select Preview Scene can also run the current project. By the way, the Preview mode is only for preview and all the remote control events are inactive.

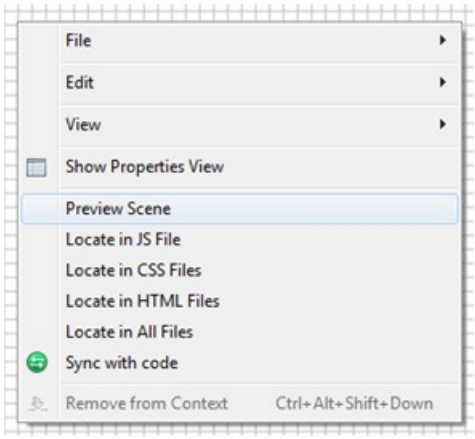



**Figure:** Preview Scene

# Embedded Log Viewer

Embedded Log Viewer is a viewer in Samsung Smart TV SDK that can see log message which is delivered from the Emulator. When emulator is launched and connected to IDE, the log viewer will be shown the IDE editor.

Developers can open Log Viewer by the path Window ▸ Show View ▸ Console as shown in the below image.

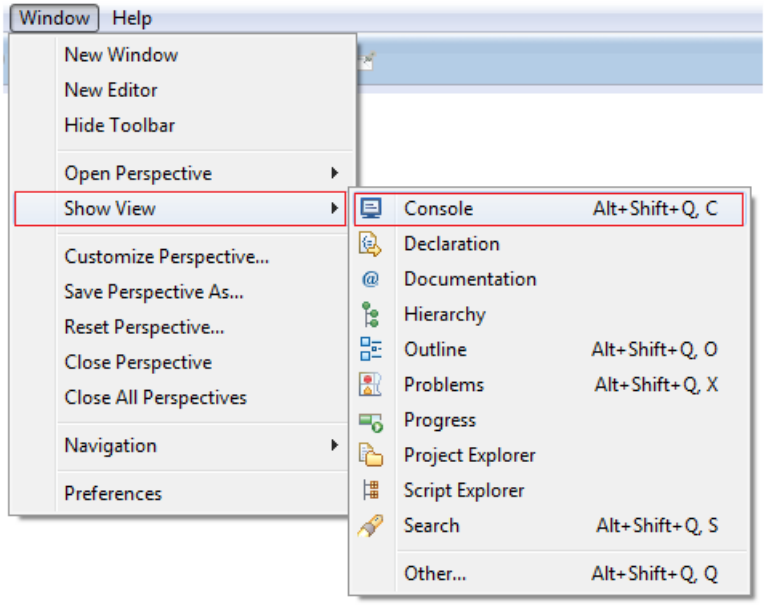



**Figure:** Menu path

When developers launch an App in emulator and Emulator responses to the IDE, the log message will be shown in the embedded console view. There are some features about console view.

Hyperlink can be shown if do as follow steps:

1. Edit js file (such as Scene1.js in basic project). Add alert message in the js file (see red rectangle). The alert message needs to follow these rules:

   Start with file://;

   Then follows an absolute file path, such as D:/dev/runtime-New_configuration/basic/app/init.js

   The file specified by the absolute path must exist in current project.

2. Run current project in Samsung Smart TV emulator.

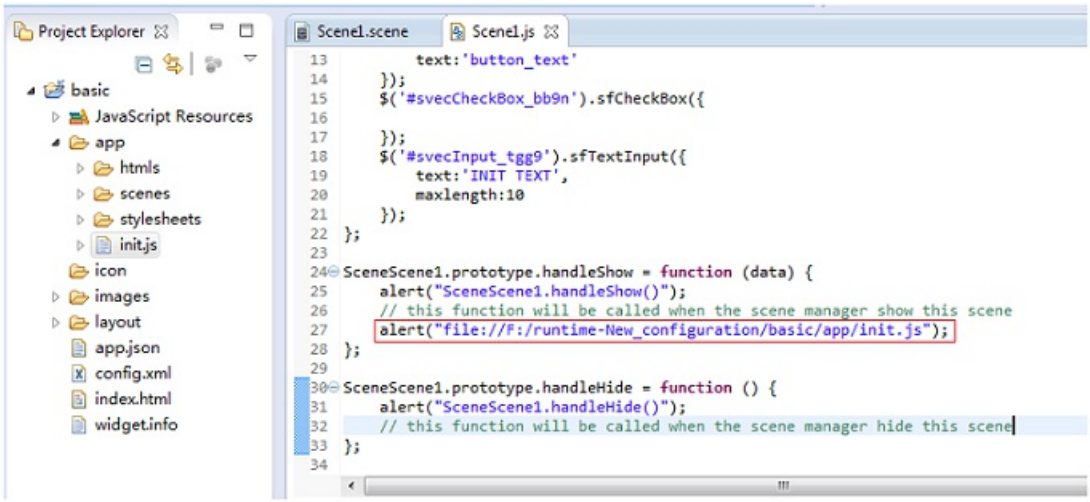



**Figure:** Samsung Smart TV Emulator Log View

If the hyperlink pointed file is in the current project and the file's absolute path is correct, this hyperlink is clickable, and the relevant file can be displayed.

The log viewer also contains a filter function to process the message contents from Emulator. The filter buttons is shown in the console view with four levels to choose:

All

show all log messages responded from emulator.

Alert

show alert messages which starts with "[JS ALERT]".

Error

show error messages.

Keyword

allow user to type in a keyword and show the log messages which contains the keyword.

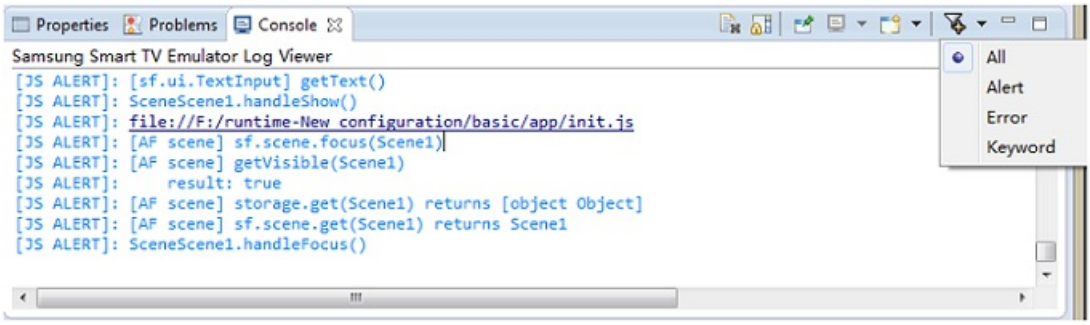



**Figure:** Log View - four levels

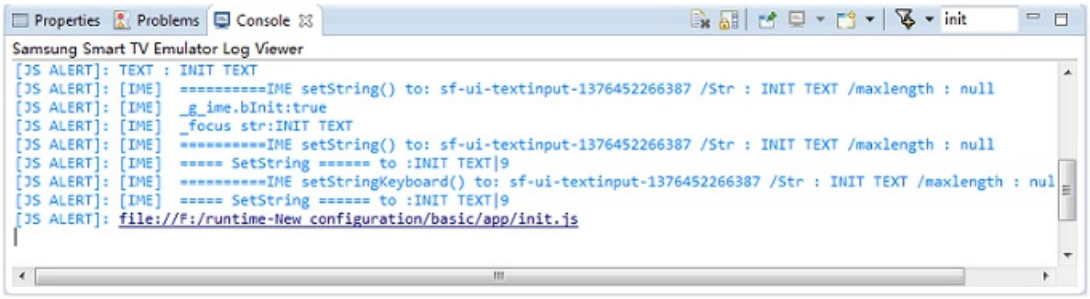



**Figure:** Filter log messages by keyword

The log message received from Emulator can also be logged into a local file, user.home/Samsung/Logs/emulator.log, automatically when the checkbox Save emulator logs is selected. The default value of the Save emulator logs is unselected which can be changed from Window ▸ Preferences ▸ Samsung Smart TV ▸ Emulator.

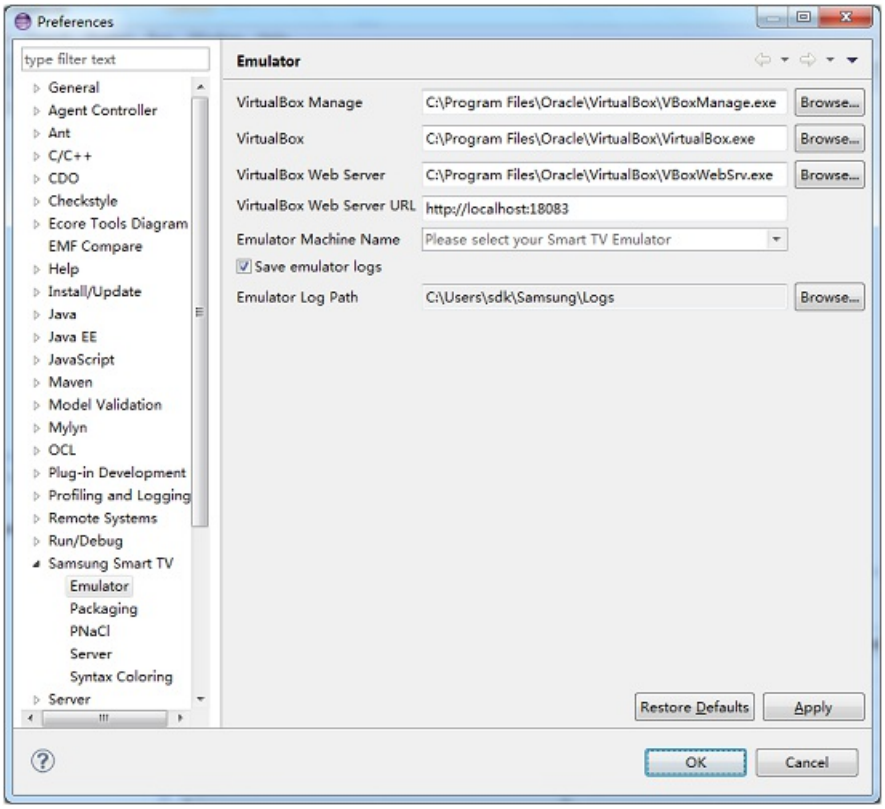



**Figure:** Log saving

The user can also change the emulator log path by clicking Browser... when the checkbox Save emulator logs is selected.

# Samsung API Syntax Highlight

With Samsung API Syntax Highlight integrated in the Samsung Smart TV SDK JavaScript Editor, the Samsung API keywords can easily be distinguished from others. There are three kinds of Samsung API keywords defined in Samsung Device API, and each keyword type is highlighted differently.

Samsung Application Interface.
Samsung Methods.
Samsung Parameter Variables.

By default, the colors for Samsung API keywords are as follows:

| Keyword Type | Color Settings |
| --- | --- |
| Samsung Application Interface | RGB (0, 128, 255) |
| Samsung Methods | RGB (255, 100, 100) |
| Samsung Parameter Variables | RGB (63, 63, 191) |

Scene1.js

```
1  alert('SceneScene1.js loaded');
2
3  function SceneScene1() {
4
5  };
6
7  SceneScene1.prototype.initialize = function () {
8      alert("SceneScene1.initialize()");
9      // this function will be called only once when the scene manager show this scene first time
10     // initialize the scene controls and styles, and initialize your variables here
11     // scene HTML and CSS will be loaded before this function is called
12     audiocontrol.setVolume(AUDIO_SOUND_TYPE_DOWN);
13 };
14
15 SceneScene1.prototype.handleShow = function (data) {
16     alert("SceneScene1.handleShow()");
17     // this function will be called when the scene manager show this scene
18 };
19
20 SceneScene1.prototype.handleHide = function () {
21     alert("SceneScene1.handleHide()");
22     // this function will be called when the scene manager hide this scene
23 };
24
```

**Figure:** Default Samsung API Keywords Colors

To customize the highlighting styles of the Samsung API keywords for the code, refer to the following steps:

Click Window ▸ Preferences (Eclipse ▸ Preferences, in Mac OS X).
Select Samsung Smart TV ▸ Syntax Coloring and it shows the current highlighting style of each Samsung API keyword type.
Click the colored button on the right side, and select the color that you want the text of the keyword to appear in, and click OK or Apply to save your changes.
Or click Restore Defaults if you want to restore the color settings of Samsung API keywords to their default values, and then click click OK or Apply to save your changes.

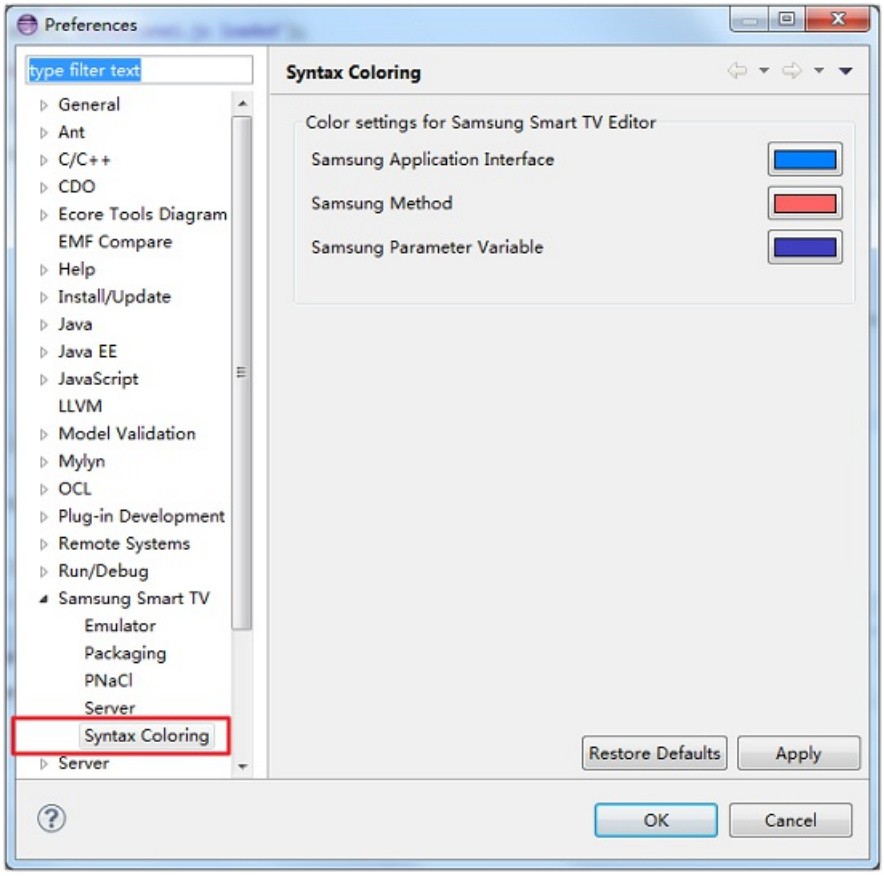



**Figure:** Samsung API keywords Preference Page

# Import and Export

With Import and Export integrated in the Samsung Smart TV SDK, Smart TV apps projects can be easily imported into your workspace and exported in compressed files for distribution and running or debugging on real Samsung Smart TVs.

## Import

To import Samsung Smart TV apps projects, refer to the following steps:

Click File ▸ Import.

In the Import dialog, select Samsung Smart TV Apps folder.

There are two options for Samsung Smart TV apps projects importation: Samsung Smart TV Native Apps Project for PNaCl apps and Samsung Smart TV Web Apps Project for AF 2.0 app and JavaScript apps.

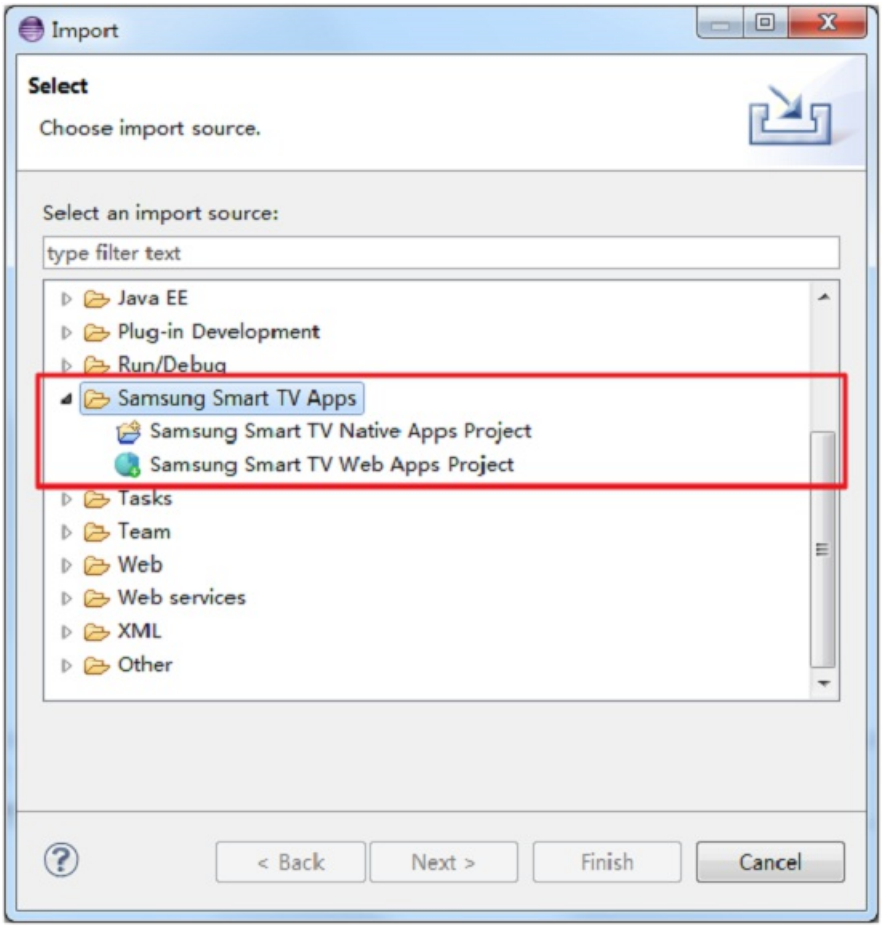



**Figure:** Import Samsung Smart TV Apps Projects

## Export

To export Samsung Smart TV apps projects, refer to the following steps:

1. Before exporting projects, certain Packaging settings should be configured.

   Click Window ▸ Preferences (Eclipse ▸ Preferences, in Mac OS X) and select Samsung Smart TV -> Packaging. The Registered Area List on the right side lists all the target distribution areas of your Smart TV apps, which can be newly added, deleted or modified through the commands on the side. And the Package file output folder specifies the destination where the packaged apps would be.

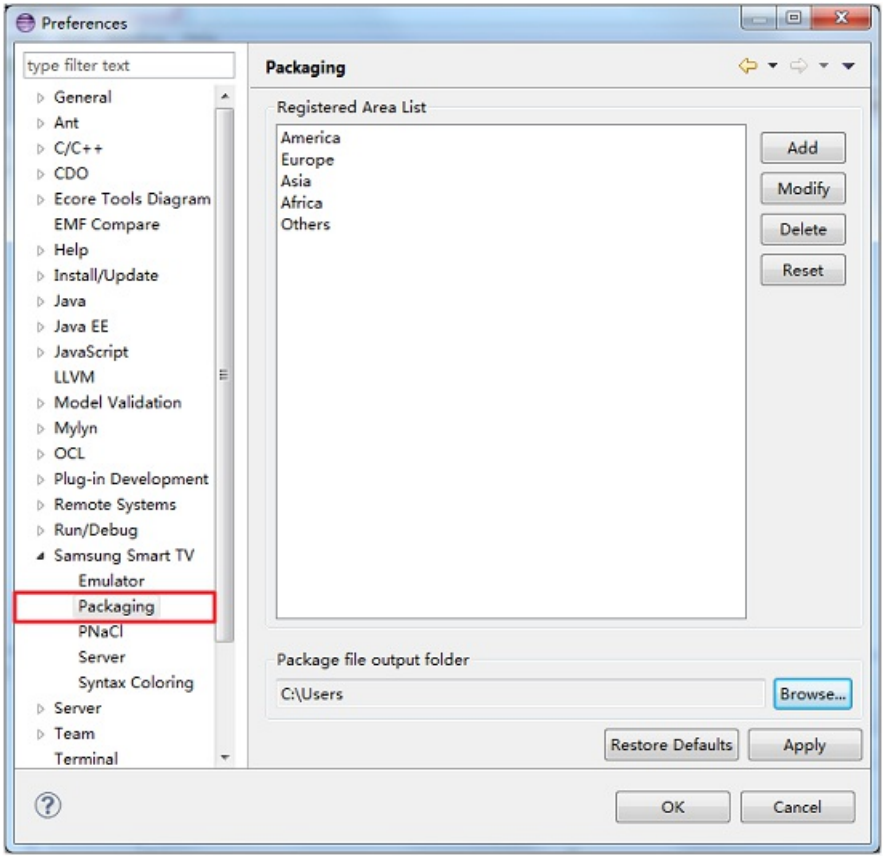



**Figure:** App Packaging Settings

2. If you want to run your apps on real Samsung Smart TV, **Server** should be configured. (Optional)

   Click Window ▸ Preferences (Eclipse ▸ Preferences, in Mac OS X) and select Samsung Smart TV ▸ Server. Then, set the IP address and resource path of the server.

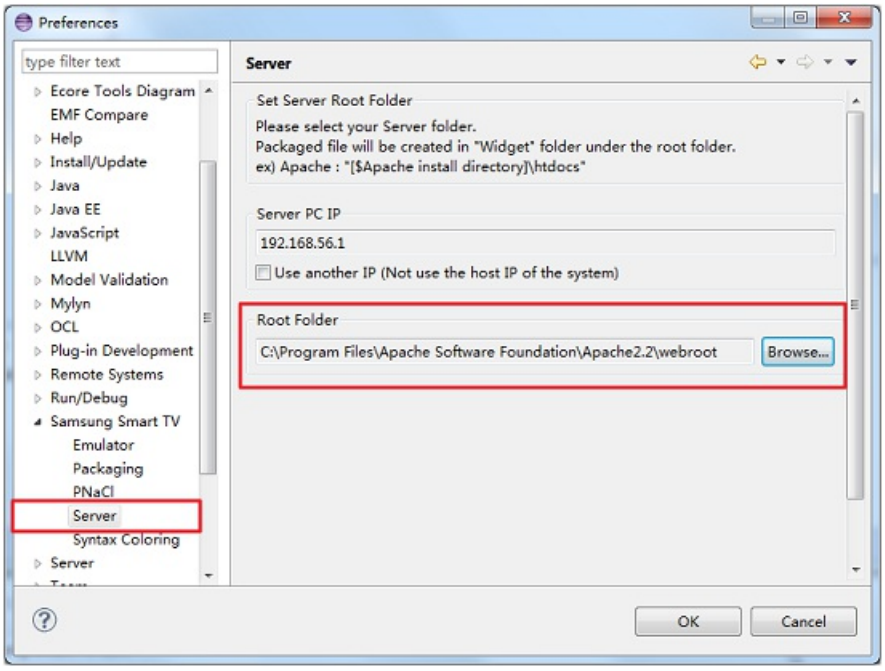



**Figure:** Server Settings

1. Click File ▸ Export, and select Samsung Smart TV Apps ▸ Packag`e file in :menuselection:`Export dialog.

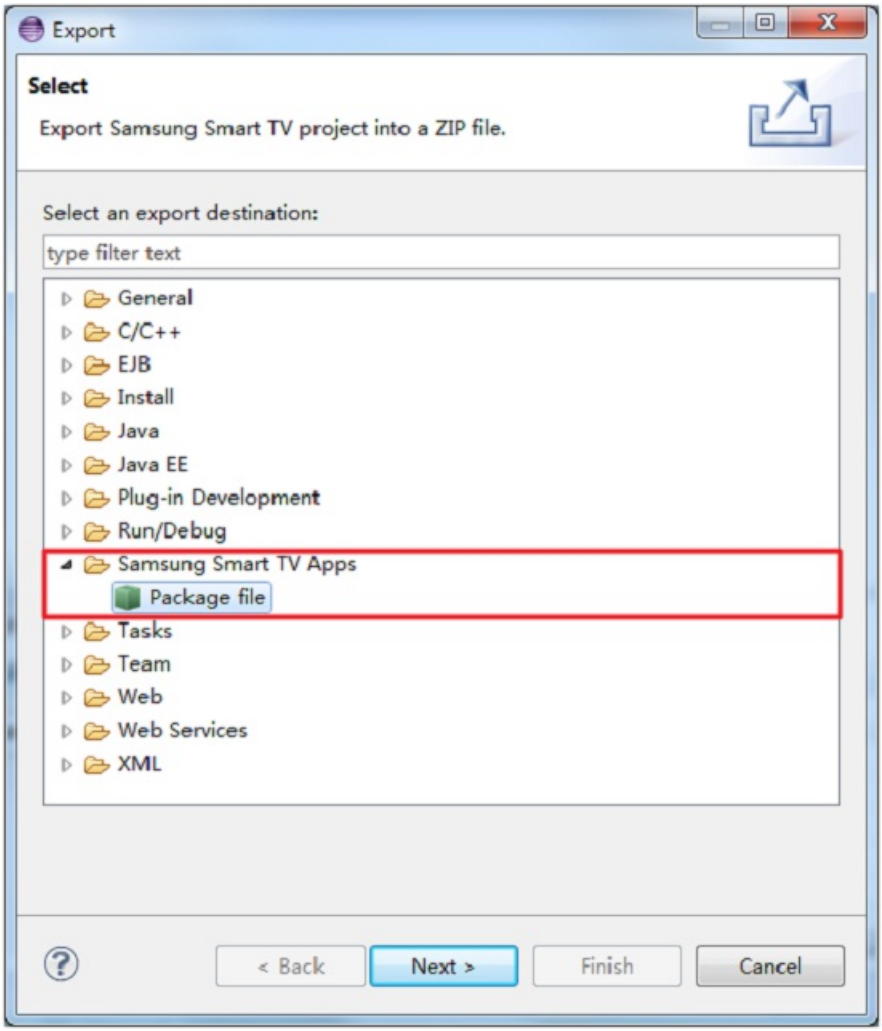



**Figure:** Export Samsung Smart TV Apps Projects

2. Configure essential packaging information, such as app version, distribution area and available languages, in ZIP Export dialog. Your app on the server would be updated if the checkbox Update the packaged files on the server is checked.

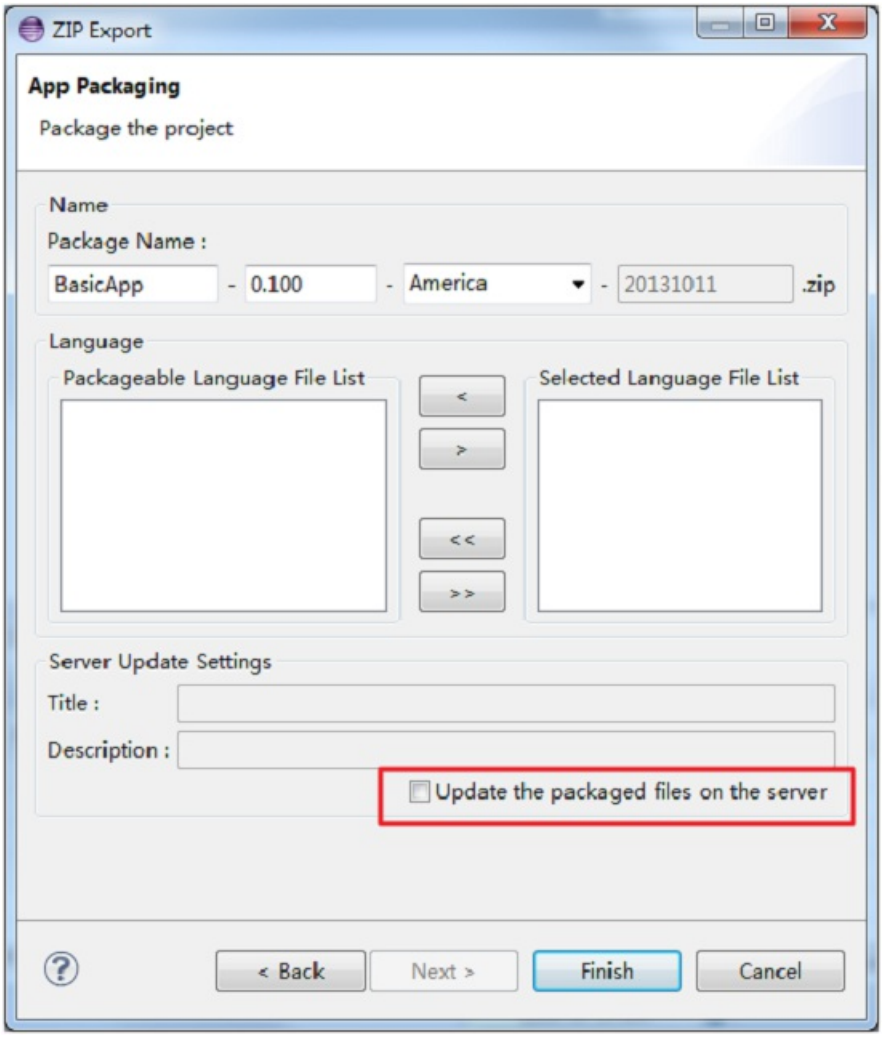



**Figure:** Configure essential packaging information

# Context Help

Context Help function is the Eclipse self-contained function which is also implemented in Samsung Smart TV IDE. By selecting one of the UI components and pressing the F1 button, a new Help View will be opened with a paragraph of description and some hyperlinks to the relevant articles. Within the Smart TV IDE, several views such as Visual Scene/Kit Editor, Properties and Outline view, integrate this context help function.

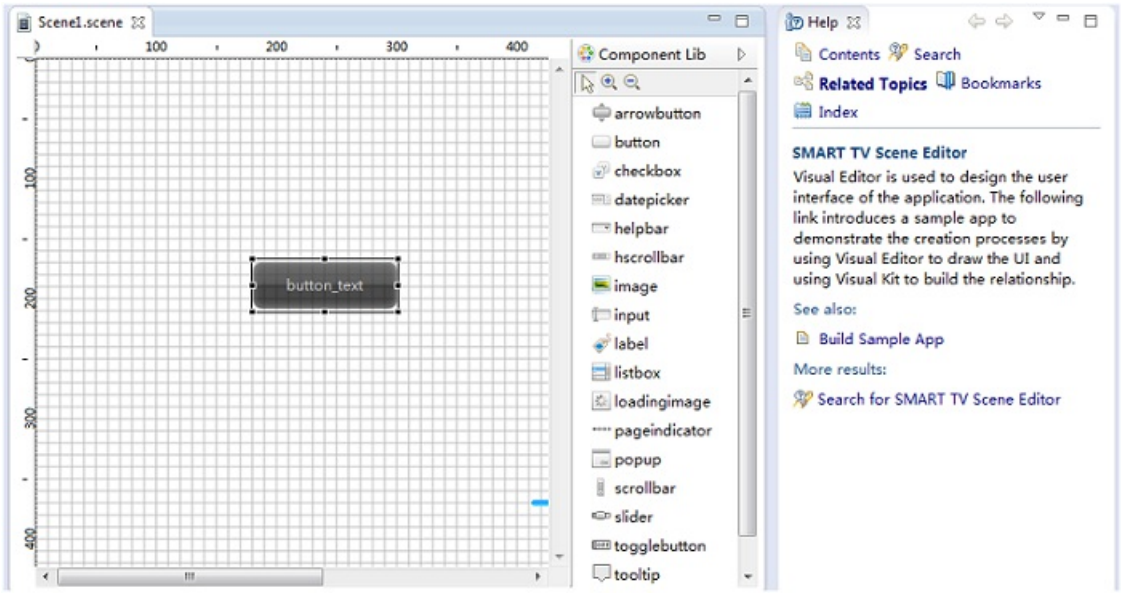



**Figure:** Context Help about Visual Scene

# Configuration File

With the help of Samsung Smart TV SDK XML Editor, user can edit certain fields such as resolution, app name, and icon preview with great convenience.

1. Select <Project Name>config.xml file, and then right click the selected xml file and select Open With Samsung Smart TV SDK XML Editor in the context menu.
2. Navigate to the configuration page in the first tab, and editing certain fields.

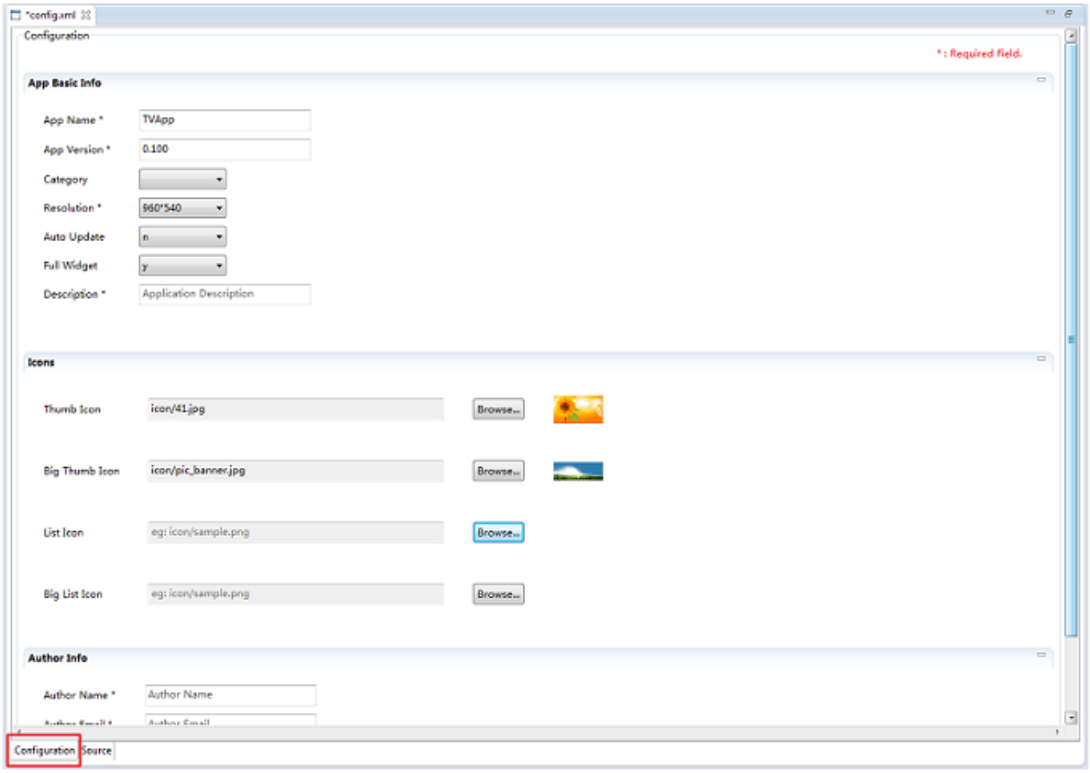



**Figure:** Visual Editing of Config.xml