Coding Your JavaScript Application

Published 2014-10-28 | (Compatible with SDK 2.5,3.5,4.5,5.0,5.1 and 2011,2012,2013,2014 models)

> Learn how to create basic Smart TV app using JavaScript, CSS and HTML.

Contents

Example Application

- **Writing the config.xml file**
- **Writing the index.html file**
- **Writing JavaScript file**
- **Writing a CSS file**
- **Making the Application Respond to the Remote Control**

A Smart TV application contains at least the following elements:

index.html file

Serves as the access point of the application.

config.xml file

An XML file in the root of the application structure that holds information about setting up the application.

JavaScript files

Allow a preview of the application and control the behavior of the application.

CSS files

Define the look and feel of the application.

Image files

The images used by the application.

For more information about creating a JavaScript project type application, see JavaScript.

Example Application

The the following example explains the implementation of an application that displays text written in CSS and responds to remote control button events in terms of:

- Writing the config.xml file
- Writing the index.html file
- Writing JavaScript file
- Writing a CSS file
- Making the Application Respond to the Remote Control

For a full set of example code, see Coding Your JavaScript Application: Sample Code.

The figure below illustrates the file structure and appearance of the application.

Figure. File structure of example application **Figure**. Example application

Writing the config.xml file

The config.xml file contains information about the application's execution, updates, operating environment settings and so on. Depending on the information, the Application Manager controls the version of the application, sets the environment in which the application is run, and creates and manages user accounts. The config.xml file must be located in the directory in which the application is installed, and contain the tags listed in the table below.

The config.xml file is called first. The <ver> tag value indicateds whether to update the application. The <ThumbIcon> tag defines the images shown in the thumbnail.

```
<?xml version="1.0"encoding="UTF-8"?>
<widget>
  <ThumbIcon>Resource/image/icon/picasa_106.png</ThumbIcon>
  <BigThumbIcon>Resource/image/icon/picasa_115.png</BigThumbIcon>
  <ListIcon>Resource/image/icon/picasa_85.png</ListIcon>
  <BigListIcon>Resource/image/icon/picasa_95.png</BigListIcon>
  <category></category>
  <autoUpdate>n</autoUpdate>
  <cpname>MyCP</cpname>
  <cpauthjs></cpauthjs>
  <login>y</login>
  <ver>0.930</ver>
  <mgrver>1.000</mgrver>
  <fullwidget>y</fullwidget>
  <srcctl>y</srcctl>
  <ticker>n</ticker>
  <childlock>n</childlock>
  <audiomute>n</audiomute>
  <videomute>n</videomute>
  <dcont>y</dcont>
  <network>y</network>
  <widgetname>HelloWorld</widgetname>
  <description>Welcome!</description>
  <width>960</width>
  <height>540</height>
  <author>
    <name>Samsung Electronics Co. Ltd.</name>
    <email></email>
    <link>http://www.sec.co.kr/</link>
    <organization>Samsung Electronics Co. Ltd.</organization>
  </author>
</widget>
```

Tag information

Element	Description	Value
<widget>	Indicates that the information is relevant to the application.	-
<ThumbIcon>	An icon image displayed in the Application Manager. It is used in case of no focus and its size is 106 x 86 pixel.	File path
<BigThumbIcon>	An icon image displayed in the Application Manager. It is used in case the focus is placed on an image and its size is 115 x 95 pixel.	File path
<ListIcon>	An icon image displayed in the Application Manager. The size is 85 x 70 pixel.	File path
<BigListIcon>	An icon image displayed in the Application Manager. The size is 95 x 78 pixel.	File path
<category>	The category to distinguish applications. Available items are video, sports, game, lifestyle, information, education.	String
<autoUpdate>	Decides whether to synchronize with the hub site. For an application that does not need synchronization, select ‘n’.	y | n
<apptype>	Shows information on contents type. 11: HTML + Javascript + Flash Player Object 12: Adobe SWF (Ver. Flash Lite 3.1) 13: Adobe SWF (Ver. Flash 10.1) 14: Lua Script	Number

Element	Description	Value
<contents>	File path and name at the initial execution of contents Only the following application types need contents tag: 12: Adobe SWF (Ver. Flash Lite 3.1) 13: Adobe SWF (Ver. Flash 10.1) 14: Lua Script	File Path
<channelType>	Channel-bound Service Type (optional)	root | child
<channelRoot>	Confirms the relations with root-child clarifying root application ID. Optional, only used when the channel-bound service type is child) When connected to more than one root, roots are distinguished by '::'.	Application ID
<channelName>	Channel information to be executed for channel-bound service (Distinguish each channel using '::' for example: AAA::BBB::CCC) (Optional, only used when channel-bound service type is the root)	String
<channelDisplay>	Decides whether the installed channel-bound service is displayed on the first main screen or not. If you want to hide the service in the first main screen, select 'n'.	y | n
<previewjs>	**Deleted**	String
<cpname>	Enter the application provider in this tag.	String
<cpauthjs>	The name of the JavaScript file that allows you to confirm account information of application providers. This file has to be written in a defined format.	String
<cplogo>	**Deleted**	File path
<prelcon>	**Deleted**	File path
<login>	Decides whether a service is available for login or not. Select 'y', to enter ID and password in the Integrated Sign-in site of the Application Manager for login. Validity verification should be preceded in JavaScript file corresponding to <cpauthjs> tag value.	y | n
<ver>	Indicates the application version. The server computer updates the corresponding application depending on the version information.	x.xxx
<mgrver>	Indicates the Application Manager version that is required to run an application having the config.xml file.	x.xxx
<fullwidget>	Indicates whether the application is a full-screen or a single-wide one. Display type affects the audio policy of the application when it's run. See Handling Remote Control Key Events for details.	y | n
<srcctl>	If 'y' is selected, the TV source automatically switches from the current TV channel or external input to the internal media player, and back again when the application is completed. If source conversion is needed, set this tag to y. (ex. YouTube application).	y | n
<childlock>	Determines whether to use the childLock function. This function enables the user to lock an application.	y | n
<audiomute>	Turns on or mutes the audio. If 'y' is selected, TV broadcasting sound is muted when entering the application. Select 'y' if the application occupies the full screen, and select 'n' if the application occupies a part of the screen.	y | n
<videomute>	Turns on or turns off the video. If 'y' is selected, TV broadcasting is not displayed on the screen when entering the application.	y | n
<dcont>	Sets the Disable dynamic contrast function. Dynamic contrast is the function to adjust TV contrast ratio and brighten TV screen by darkening the dark screen and lightening the light screen. The screen might get lighter or darker when application is on with Dynamic contrast. Selecting 'y' turns off the Dynamic contrast, and selecting 'n' turns on the Dynamic contrast. If the application occupies the full screen, select 'y' to remove sparkling. If the application occupies a part of the screen, select 'n'.	y | n
<movie>	Applications playing video files can cause problems as stated below: 1. If a video file is played on a device connected to the HDMI port, such as a DVD Player, sounds can get mixed, when executing an application converting sources (for example, YouTube). 2. Sparkling can happen at the entry of the application, due to the difference of frame rate between the TV image and video file. Such problems can be avoided by selecting 'y' - the HDMI device is stopped, or the frame rate is fixed.	y | n

Element	Description	Value
<widgetname>	Enter name of the application.	String
<description>	Enter a brief description of the application.	String
<width> <height>	Enter the screen area that the application will occupy. It is recommended to enter 960 * 540 pixels, the digital TV specification.	Number
<author>	Enter the name of the author.	string
<network>	This tag is used to check the network while operating an application. If the tag value is 'y' and the network test result is 'fail', entry for the application can be blocked with a message indicating the failure. Without any set value, default is 'y'.	y | n
<hubsite>	This tag is used to check whether the hubsite has been authorized or not while operating an application. If the tag value is 'y', and the hubsite has not been authorized, entry for the application can be blocked with a message indicating the failure. Without any set value, default is 'n'.	y | n
<pushNotice>	Indicates whether the application provides Push Notification Service. Without any set value, default is 'n'.	y | n
<pushControl>	This tag is reserved for former Push Notification Service. Without any set value, default is 'n'.	y | n
<pushUerbinding>	Indicates whether Push Notification Service is provided for a specific user. Without any set value, default is 'n'.	y | n
<flashplayer>	This tag is enabled for applications that use embedded Flash player objects, or a stand-alone Flash player.	y | n

Writing the index.html file

The index.html file is the access point of the application.

A sample HTML code snippet for the index.html file is shown below. It includes the Main.js file under the JavaScript folder and calls the Main.onLoad() function when the document loaded.

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Hello World!!</title>
    <script type="text/javascript" src="JavaScript/Main.js"></script>
  </head>
  <body onload="Main.onLoad();">
    <div>
      Welcome to Samsung application world!
    </div>
  </body>
</html>
```

Writing JavaScript file

When the HTML document is loaded, the onLoad() function of the Main object is called, as the Main.onLoad() function in the onload property is registered in the <body> tag.

Create a Main object and add the onload function:

```
var Main = {   // Main object };

Main.onLoad = function () {    // called by body's onload event
  alert("Main.onLoad()");
  /**
   * JavaScript code Here!
   */
}
```

Once all the above steps are complete, a debug message of Main.onLoad() is displayed. Note that the “Welcome to Samsung application world!” that was entered does not yet appear on the TV screen. You must use a common module, a library containing essential functions provided by the Application Manager. Add the following code to <head> of index.html file for making the common module available for use by JavaScript:

```
<script type="text/javascript" src="$MANAGER_WIDGET/Common/API/Widget.js"></script>
```

Declare the common module as a global variable for the Main.js and call the sendReadyEvent() function. This makes the Application Manager display an application on the screen:

```
var Main = {    // Main object};

var widgetAPI = new Common.API.Widget();    // Creates Common module

Main.onLoad = function () {     // called by body's onload event
    alert("Main.onLoad()");
    widgetAPI.sendReadyEvent(); // Sends ready message to Application Manager
    /**
     * JavaScript code Here!
     */
}
```

Run the application. The “Welcome to Samsung application world!” message from the index.html file now appears on the screen. To change how the application looks, use the CSS.

Writing a CSS file

Add the following string to the <head> field in the index.html file:

```
<link rel="stylesheet" type="text/css" href="CSS/Main.css"/>
```

Assign an ID to element in the index.html file.

```
<div id="welcome">Welcome to Samsung application world!</div>
```

Create a Main.css file in the CSS folder and enter the code snippet shown in the box below to specify the style of the welcome element:

```
body {
    margin: 0;
    padding: 0;
    background-color: transparent;
}

#welcome {
    position: absolute;
    left: 50px;
    top: 50px;
    width: 500px;
    height: 50px;

    background-color: #AFAFAF;
    color: #99FFFF;
    font-size: 30px;
    text-align: center;
}
```

Making the Application Respond to the Remote Control

Users can change the words your application displays on the screen by pressing any of the five buttons located in the center of the remote control. A ‘keydown’ event occurs when a button on the remote control is pressed. Add <a> element in the index.html file and register a function to be executed when that event occurs in the onkeydown property. Place focus on <a> and press the remote control key - the function registered previously is executed. Add <a> which executes the Main.keyDown() method when a ‘keydown’ event occurs:

```
<body onload="Main.onLoad();">
    <div id="welcome">
        Welcome to Samsung application world!
    </div>
    <a href='javascript:void(0);' id='anchor' onkeydown='Main.keyDown();'></a>
</body>
```

Write a keyDown() method for getting the key code value when pressing the remote control key:

```
Main.keyDown = function(){     // Key handler
    var keyCode = event.keyCode;
    alert("Main Key code : " + keyCode);
}
```

In a function processed by keys such as keydown(), each key has its own key code value. The Application Manager provides a common module containing key code values to distinguish keys. Add the code shown below to <head> element in the index.html file for using the TVKeyValue common module:

```
<script type="text/javascript" src="$MANAGER_WIDGET/Common/API/TVKeyValue.js"></script>
```

Modify the Main.js file to change the contents of welcome div tag for creating common module objects, classifying keys in the keydown() method, and defining actions for each key.

```
var Main = {    // Main object};

var widgetAPI = new Common.API.Widget();        // Creates Common module
var tvKey = new Common.API.TVKeyValue();

Main.onLoad = function() {            // called by body's onload event
    alert("Main.onLoad()");
    widgetAPI.sendReadyEvent();        // Sends ready message to Application Manager
    document.getElementById("anchor").focus();    // Sets focus on Anchor for handling key inputs
                                        // from the remote control
    /**
     * JavaScript code Here!
     */
}

Main.keyDown = function(){            // Key handler
    var keyCode = event.keyCode;
    alert("Main Key code : " + keyCode);

    switch (keyCode) {
        case tvKey.KEY_LEFT:
            alert("left");
            document.getElementById("welcome").innerHTML = "Nice to meet you.";
            /**
             * Code for Left key event!
             */
            break;
        case tvKey.KEY_RIGHT:
            alert("right");
            document.getElementById("welcome").innerHTML = "I'm so happy.";
            break;
        case tvKey.KEY_UP:
            alert("up");
            document.getElementById("welcome").innerHTML = "I Love you.";
            break;
        case tvKey.KEY_DOWN:
            alert("down");
            document.getElementById("welcome").innerHTML = "Good job.";
            break;
        case tvKey.KEY_ENTER:
            alert("enter");
            break;
        case tvKey.KEY_RETURN:
            break;
    }
}
```

The value of the welcome div tag changes when the up, down, left, or right button is pressed.