Gesture Recognition

Published 2014-10-27 | (Compatible with SDK 4.5,5.0,5.1 and 2013,2014 models)

> This document shows how to use Gesture Recognition in Smart TV applications

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.

Introduction

Gesture Recognition enables users to control their Smart TVs with hand gestures. Smart TV's recognition engine tries to match user's hand gestures to predefined set of gesture samples. This operation is performed locally by the TV.

Gestures can be divided into two types:

Gestures emulating computer mouse - simple hand movement and grab gestures are translated into standard JavaScript mouse events, respectively: mousemove and click. Application which has mouse support implemented can take advantage of this type of gestures with no further modification.

Extended gestures - these gestures need to be handled using dedicated API.

This tutorial presents how to develop a simple application that utilizes the extended gestures capability.

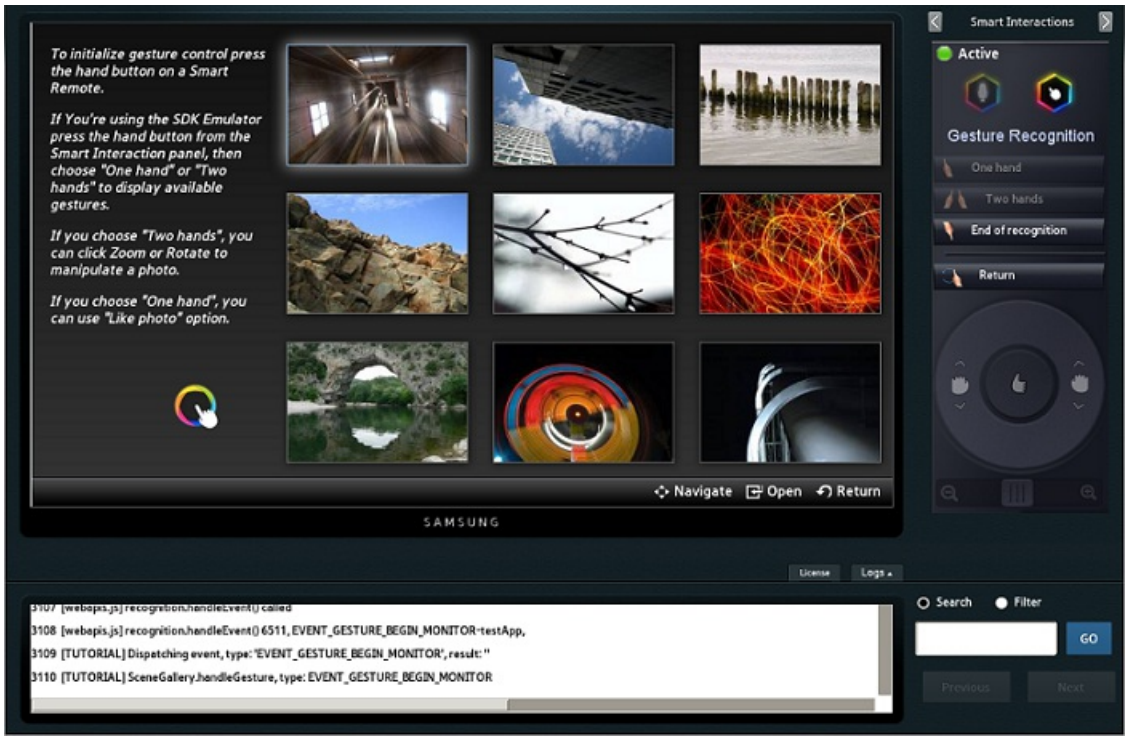

Figure 1. Initialized tutorial application on the virtual machine based version of the Emulator

Required files and settings

To run the application that you will develop according to this tutorial, please download the **tutorial application source code** and extract it into the Samsung Smart TV SDK application folder.

Gesture features require the Web Device API library. There are two ways to access it:

1. Use a standalone version of the library.
   File: /index.html
   <script type="text/javascript" src="$MANAGER_WIDGET/Common/webapi/1.0/webapis.js"></script>
2. Use the Application Framework 2.0 loader.
   File: /index.html
   <script type="text/javascript" src="$MANAGER_WIDGET/Common/af/2.0.0/loader.js"></script>

After this step, the Recognition API will be available under the webapis.recognition namespace.

Additionally, application configuration needs to be modified to enable mouse support. Ensure that the following line is present in application's config.xml file.

File: /config.xml

<mouse>y</mouse>

Checking for Recognition support

Before performing any action related to Gesture Recognition, you should check if this feature is supported and enabled on the device your application is running on. Use IsRecognitionSupported and IsGestureRecognitionEnabled functions for this purpose.

You can find them in the Setup scene's handleShow method, in /app/scenes/Setup.js.

File: /app/scenes/Setup.js

```
// checks if device supports voice and gesture recognition

if (webapis.recognition.IsRecognitionSupported() !== true) {
    $("#error").text("ERROR: Gesture recognition not supported");
    return;
}

// checks if gesture recognition is enabled

if (webapis.recognition.IsGestureRecognitionEnabled() !== true) {
    $("#error").text("ERROR: Gesture recognition is not enabled");
    return;
}
```

Subscribing to events

Use the SubscribeExEvent function to subscribe for Gesture Recognition events.

File: /app/scenes/Setup.js

```
var subscribeResult = webapis.recognition.SubscribeExEvent(
    webapis.recognition.PL_RECOGNITION_TYPE_GESTURE, "testApp",
    GestureDispatcher.handleGesture);
```

The SubscribeExEvent function requires three parameters:

- The recognition type (in this case it's PL_RECOGNITION_TYPE_GESTURE)
- Subscription name, which can be later used for unsubscribing the handler (here it's testApp)
- A callback function which fires when a recognition event occurs (here it's GestureDispatcher.handleGesture)

Unsubscribing from events

When the application is closed, you should unsubscribe from gesture events with the UnsubscribeExEvent function. If your application uses the Application Framework, unsubscribe in the onDestroy function which can be found in /app/init.js. Otherwise, you can unsubscribe when a window.onunload event occurs.

File: /app/init.js

```
webapis.recognition.UnsubscribeExEvent(webapis.recognition.PL_RECOGNITION_TYPE_GESTURE, "testApp");
```

The UnsubscribeExEvent function takes two parameters: an event type and a subscription name. The parameters must be equal to the ones that were used when calling the SubscribeExEvent function.

Handling events

When a recognition event occurs, the callback function that was given during event subscription is invoked. This function receives an event object as a parameter.

There can be only one gesture callback registered. Because of that the GestureDispatcher helper object was introduced in the application. It routes received gesture events to the currently focused scene.

File: /app/scenes/Setup.js

```
var GestureDispatcher = {
    currentScene: null, /**< Current scene object, set it when a scene is focused */

    /**
     * Passes received gesture event to currently focused scene.
     * The scene object needs to have handleGesture() method.
     * @param e Gesture event object
     */
    handleGesture: function(e) {
        log("Dispatching event, type: '" + e.eventtype + "', result: '" + e.result + "'");

        if (GestureDispatcher.currentScene !== null) {
            GestureDispatcher.currentScene.handleGesture(e);
        }
    }
};
```

Each gesture event object contains multiple properties, including eventtype and result. Based on this information, gesture event can be distinguished and handled. This can be done similarly like in the PhotoView scene. The switch statement on e.eventtype allows the application to determine which action should be performed.

File: /app/scenes/PhotoView.js

```
this.handleGesture = function(e) {
    log("ScenePhotoView.handleGesture, type: " + e.eventtype);

    switch (e.eventtype) {
    case "EVENT_GESTURE_2HAND_ZOOM":
        // calculate zoomRatio ratio based on the accumulated distance

        zoomDistance += parseInt(e.result, 10);
        // clamp zoom distance value to some arbitrary range
        zoomDistance = Math.max(-SCREEN_W * 0.75, Math.min(SCREEN_W * 1.25, zoomDistance));
        // current zoom ratio
        var zoomRatio = (SCREEN_W + zoomDistance) / SCREEN_W;

        Grid.zoom(zoomRatio);
        break;

    case "EVENT_GESTURE_2HAND_ROTATE":
        // use accumulated rotation angle
        rotationAngle -= parseInt(e.result, 10);
        Grid.rotation(rotationAngle);
        break;

    case "EVENT_GESTURE_LIKE":
        Grid.likePhoto();
        break;
    }
};
```

Possible gesture event types include:

EVENT_GESTURE_BEGIN_MONITOR
triggered when user’s first hand is detected

EVENT_GESTURE_SECONDARY_DETECT
triggered when user’s second hand is detected

EVENT_GESTURE_SECONDARY_LOST
triggered when user’s second hand is out of camera view

EVENT_GESTURE_2HAND_ZOOM
triggered when “zoom in/out” gesture is recognized; this event has a result property set which holds the change in distance between two hands since the last event of this type
EVENT_GESTURE_2HAND_ROTATE
triggered when rotate gesture is recognized; this event has a result property set which holds the change in rotation angle (in degrees) since the last event of this type
EVENT_GESTURE_LIKE
triggered when thumb-up gesture is recognized

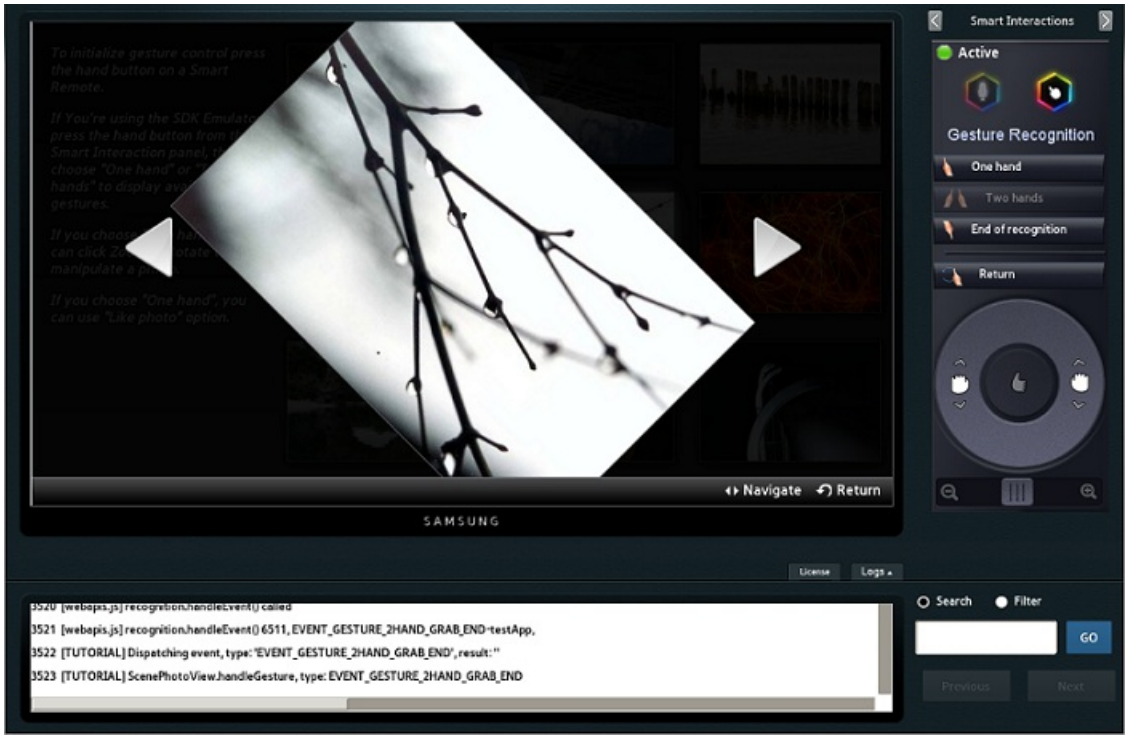

Figure 2. Usage of rotate gesture in the application

Handling mouse events

As already mentioned, the basic hand gestures are translated to standard mouse events. It is desired for the application to support mouse events such as click. This way, the application can be controlled both by a computer mouse connected to the TV and by means of gesture recognition.

For more information on adding mouse support refer to Mouse Event Handling.

Setting up the gesture helpbar

The gesture helpbar is a screen overlay which shows graphical elements to the user, informing him about available and supported recognition features, e.g. different gesture commands.

> Note
>
> The layout of the screen overlay may vary depending on the version of the device or Smart TV SDK Emulator used in development process.

SetGestureHelpbarInfo function is used to set up the gesture helpbar and its items. Gesture helpbar configuration must be passed as a string. Following code fragments illustrate the format of the configuration and what items can be used.

> Tip
>
> You can use the JSON.stringify method to convert a JavaScript object into a string.

File: /app/scenes/Gallery.js

```
this.handleShow = function () {
    log("SceneGallery.handleShow()");

    var helpBarInfo = {
        helpbarType: "HELPBAR_TYPE_GESTURE_CUSTOMIZE",
        itemList: [
            { itemType: "HELPBAR_GESTURE_ITEM_RETURN", itemText: "Go back" },
            { itemType: "HELPBAR_GESTURE_ITEM_POINTING_GESTURE", itemText: "Choose picture" }
        ]
    };

    webapis.recognition.SetGestureHelpbarInfo(JSON.stringify(helpBarInfo));

    [...]
};
```

File: /app/scenes/PhotoView.js

```
this.handleShow = function () {
    var helpBarInfo = {
        helpbarType: "HELPBAR_TYPE_GESTURE_CUSTOMIZE",
        itemList: [
            { itemType: "HELPBAR_GESTURE_ITEM_POSE_LIKE", itemText: "Like it" },
            { itemType: "HELPBAR_GESTURE_ITEM_ZOOM_OUT_IN", itemText: "Zoom in / out" },
            { itemType: "HELPBAR_GESTURE_ITEM_ROTATION", itemText: "Rotate photo" },
            { itemType: "HELPBAR_GESTURE_ITEM_RETURN", itemText: "Go back" }
        ]
    };

    webapis.recognition.SetGestureHelpbarInfo(JSON.stringify(helpBarInfo));

    [...]
};
```

> **Important**
>
> To receive two hand gesture events (if supported by the TV) you need to at add HELPBAR_GESTURE_ITEM_ZOOM_OUT_IN and/or HELPBAR_GESTURE_ITEM_ROTATION helpbar item.