Voice Recognition

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

> This document shows how to use Voice 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

Voice Recognition enables users to control their Smart TVs with voice commands. Two modes of recognition are supported: embedded mode and server mode. In the first mode, the TV tries to match voice samples to one of predefined commands. This operation is performed locally by the TV. In the second mode, voice samples are sent to an external server which converts them into text and sends the results back to the TV. In this tutorial you will learn how to develop an application which lets the user control it with voice.

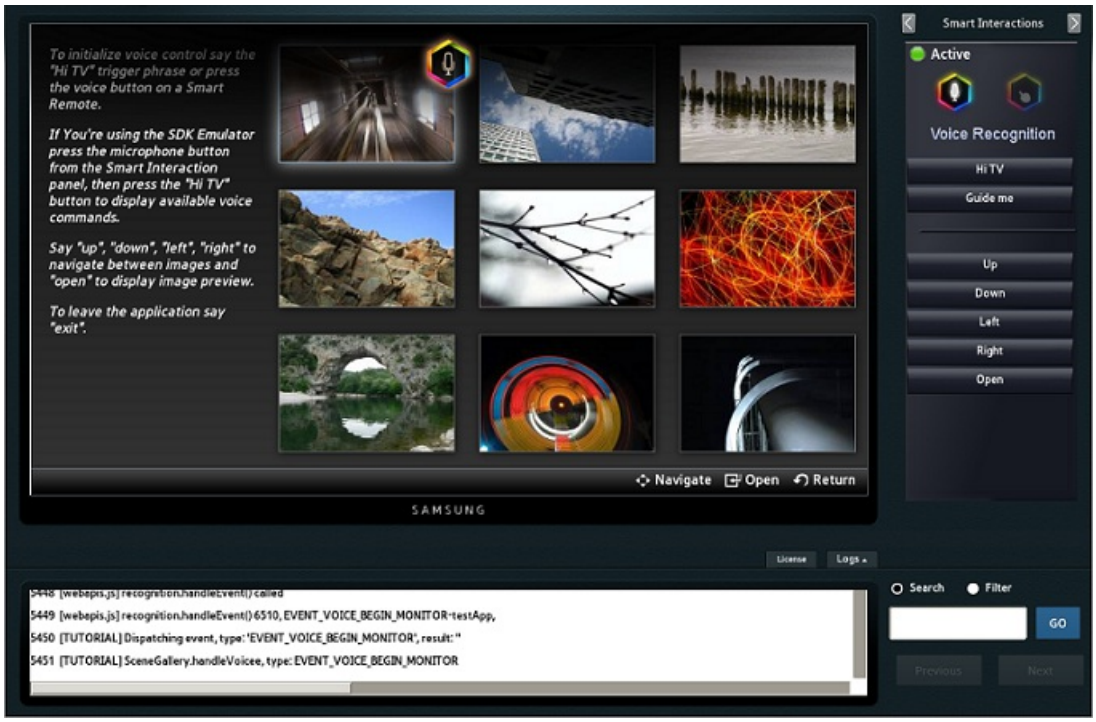

Figure 1. Initialized tutorial application on the virtual machine based version of the Emulator

Source Files

> Note
>
> The files needed for the sample application are **here**.

Required files

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.

Recognition features require API library. There are two ways to access it:

Use a standalone version of the library.

```
<script type="text/javascript" src="$MANAGER_WIDGET/Common/webapi/1.0/webapis.js"></script>
```

Use the library below.

```
<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.

Checking for recognition support

Before performing any action related to Voice Recognition, you should check if this feature is supported by the device your application is running on. Use the IsRecognitionSupported function for this purpose. You should also check if voice recognition is enabled on this device, using IsVoiceRecognitionEnabled function.

You can find the code responsible for those actions 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) {
    document.getElementById("error").innerHTML = "ERROR: Voice recognition not supported";
    return;
}

// checks if voice recognition is enabled

if (webapis.recognition.IsVoiceRecognitionEnabled() !== true) {
    document.getElementById("error").innerHTML = "ERROR: Voice recognition is not enabled";
    return;
}
```

Subscribing to events

Use the SubscribeExEvent function to subscribe to Voice Recognition events.

File: /app/scenes/Setup.js

```
var subscribeResult = webapis.recognition.SubscribeExEvent(
    webapis.recognition.PL_RECOGNITION_TYPE_VOICE, "testApp",
    VoiceDispatcher.handleVoice);
```

The SubscribeExEvent function requires three parameters:

- The recognition type (in this case it's PL_RECOGNITION_TYPE_VOICE)
- Subscription name, which can be later used for unsubscribing the handler
- A callback function which is being invoked when a recognition event occurs

Unsubscribing from events

When the application is closed, you should unsubscribe from voice 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 file. Otherwise you can unsubscribe when a window.onunload event occurs.

File: /app/init.js

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

The UnsubscribeExEvent function takes two parameters: an event type and a subscription identifier. 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 set up during event subscription is invoked. The function receives an event object as a parameter. There can be only one voice event callback registered. Because of that the VoiceDispatcher object was introduced in the application. It routes received voice events to the currently focused scene.

File: /app/scenes/Setup.js

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

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

        if (VoiceDispatcher.currentScene !== null) {
            VoiceDispatcher.currentScene.handleVoice(e);
        }
    }
};
```

Voice event object passed by the VoiceDispatcher to the scene's handler contains properties specifying the type of the event and the recognition result. Basing on this information, appropriate action can be performed.

File: /app/scenes/PhotoView.js

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

    switch (e.result.toLowerCase()) {
        case "left": Grid.left(); break;
        case "right": Grid.right(); break;
        case "exit": sf.core.exit(); break;
        case "close":
            sf.scene.hide("PhotoView");
            sf.scene.show("Gallery");
            sf.scene.focus("Gallery");
            sf.key.preventDefault();
            break;
        case "describe":
            (...)
            isDescriptionOn = true;
            break;
        case "return":
            if (isDescriptionOn) {
                (...)
                isDescriptionOn = false;
            }
            break;
        default:
            if (isDescriptionOn) {
                // Recognize any text description of a photo in gallery.
                Grid.setDescription(e.result);
                (...)
                isDescriptionOn = false;
            }
            break;
    }
};
(...)
```

In property event.result there is an information about the recognized text. In the preceding code snippet you can see a command handler for both embedded mode set by user (left, right, exit, close, describe) and server mode (return).

In this application, there is a possibility to set a text description for each photo in a gallery. If event.result is not recognized as keyword (left, right, exit, close, describe or return), we treat it as a description, if the description mode is on (its state is held in isDescriptionOn variable).

Setting up the voice helpbar

The voice helpbar shows available voice commands and/or a guide text at the top of the screen. It acts as a guide for the user prompting which voice commands are currently supported by the application. Helpbar contents can be changed by the application at any time.

SetVoiceHelpbarInfo function is used to set up the voice helpbar type and its items. Voice helpbar configuration must be passed as a stringified JSON object. Following code snippet illustrates the format and possible properties of the configuration object.

> Important
>
> Recognition mode is determined by the type of the voice helpbar.
>
> Tip
>
> You can use the JSON.stringify method to convert a JavaScript object into a string.

Embedded mode helpbar

In embedded recognition mode, the TV tries to match voice samples to one of the commands registered with the helpbar. This operation is performed locally on the TV.

File: /app/scenes/PhotoView.js

```
helpBarInfo = {
  helpbarType: "HELPBAR_TYPE_VOICE_CUSTOMIZE",
  helpbarItemsList: [
    { itemText: "Describe", commandList: [{command: "Describe"}]},
    { itemText: "Left", commandList: [{command: "Left"}]},
    { itemText: "Right", commandList: [{command: "Right"}]},
    { itemText: "Close", commandList: [{command: "Close"}]}
  ]
};
webapis.recognition.SetVoiceHelpbarInfo(JSON.stringify(helpBarInfo));
```

Server mode helpbar

In server mode, the TV sends recorded voice samples to an external server which performs recognition. When the operation is completed, the server returns the recognized string. In addition to the guide text, the server mode helpbar can also display one of the predefined items such as OK, Cancel or Return:

HELPBAR_TYPE_VOICE_SERVER_GUIDE
default helpbar for server recognition mode
HELPBAR_TYPE_VOICE_SERVER_GUIDE_RETURN
helpbar for server recognition mode with special Return command emulation
HELPBAR_TYPE_VOICE_SERVER_GUIDE_OK
helpbar for server recognition mode with special OK command emulation
HELPBAR_TYPE_VOICE_SERVER_GUIDE_CANCEL
helpbar for server recognition mode with special CANCEL command emulation

File: /app/scenes/PhotoView.js

```
helpBarInfoDescribe = {
   helpbarType: "HELPBAR_TYPE_VOICE_SERVER_GUIDE_RETURN",
   guideText: "Say the word or phrase you wish to add as the image description"
}
webapis.recognition.SetVoiceHelpbarInfo(JSON.stringify(helpBarInfoDescribe));
```

> Tip
>
> In order to test voice recognition in the server mode using the SDK, use the input field shown on Figure 2 to enter the phrase to be recognized.

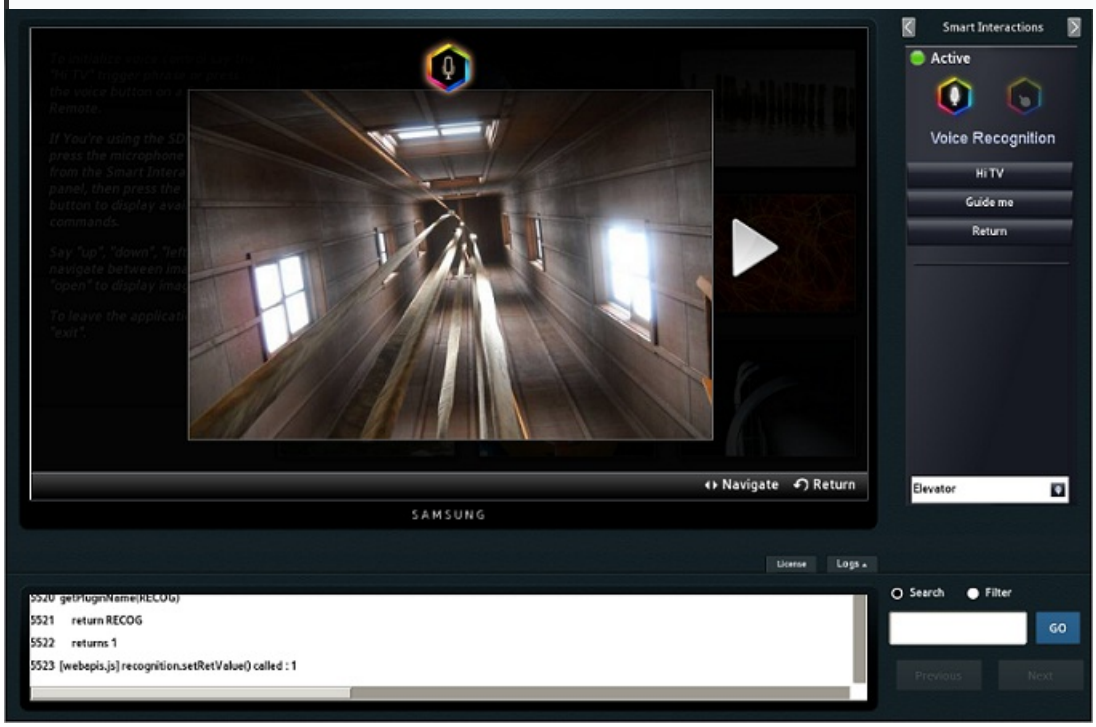

Figure 2. Smart Interactions panel with text input for simulating server recognition mode

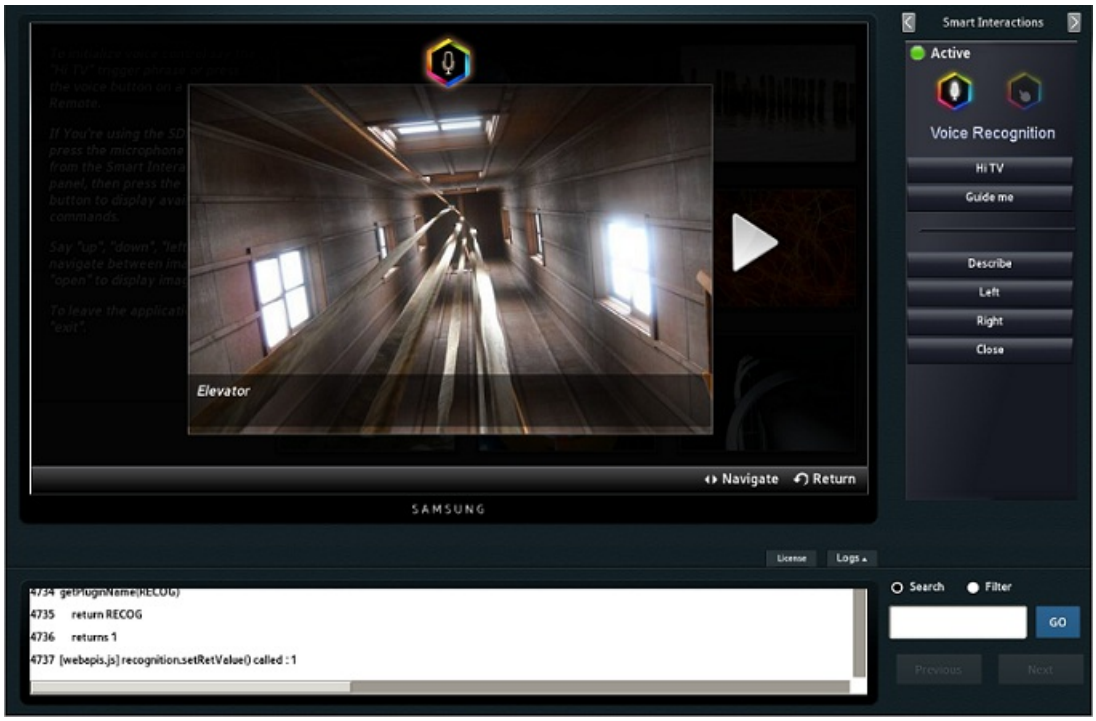

Figure 3. Application in the state just after submitting the phrase for server recognition mode