How To Use IME (Input Method Editor)

Published 2014-10-27 | (Compatible with SDK 2.5,3.5,4.5,5.0,5.1 and 2011,2012,2013,2014 models)

> The following tutorial demonstrates how to enable text input: how to create a basic application, and how to add the IME feature to the application for entering text in the input boxes, how to control the IME functions provided by the IME common module.

Contents

About IME

Samsung Smart TV provides a common module that enables text input in applications. This way, Smart TV users can enter text via a remote control. The module utilizes the Input Method Editor (IME).

> See also
> Mouse Event Handling
> Describes how to handling mouse event in application.

About this tutorial

This tutorial demonstrates adding text input capability to an application by using the Input Method Editor (IME) feature of the Application Manager. This feature allows the input of text characters using only the numeric keys available on the remote control. In this tutorial, a sample application is created with a text input box and a password input box, and user input with a virtual keyboard display is enabled. This tutorial focuses on adding IME support to a page layout. Code for 2 sample applications is provided. One of the sample applications is described in this document. The second sample application (index_simple.html, ime_sample.js) is provided as a simple example.

This tutorial consists of the following tasks:

Application with Text Input Capability
Demonstrates how to create a basic application, and how to add the IME feature to the application for entering text in the

input boxes.

Switching focus with IME

Demonstrates how the code is used to enable switching the focus from one HTML input box to another, so that the user can enter text in each of them.

Notifications and control functions with IME

Demonstrates how to implement the optional notifications available with the IME feature, by displaying various status messages.

Prerequisites

To create applications that run on a TV screen, you need:

Samsung TV connected to the Internet

SDK or a text editor for creating HTML, JavaScript and CSS files (using Samsung Smart TV SDK is recommended)

Development Environment

Use Samsung Smart TV SDK to create the application. You can also use the emulator provided with the SDK to debug and test the application before uploading it in your TV. Later, you can run the application on a TV; see Testing Your Application on a TV. Note that applications may perform better on the TV than on the emulator.

You can find general instructions for creating applications in Implementing Your Application Code.

Source Files

> Note
>
> The files needed for the sample application are **here**. **You should use the IME source files from Samsung Common Files, not the including the sample codes.**

Copy all the files from the supplied application source archive into the SDK application folder, except those in the JavaScript folder.

Directory	Description
CSS	Contains the CSS files.
Javascript	Contains the JavaScript files.

Design

Most of the complexity of international character entry is implemented in the IME feature of the Application Manager. This tutorial creates a JavaScript file (ime_sample.js), an HTML file (index.html), and a CSS file (Main.css).

Application with Text Input Capability

This task consists of the following parts:

1. Creating the Basic Application
2. Using the IME for Basic Text Entering

Creating the Basic Application

To create the basic application:

1. Start the SDK for Samsung TV Applications.
2. Create a new application using the following config.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<widget>
  <previewjs>PreviewIME</previewjs>
  <type>user</type>
  <cpname>James Grant</cpname>
  <cplogo></cplogo>
  <cpauthjs></cpauthjs>
  <ver></ver>
  <mgrver></mgrver>
  <fullwidget>y</fullwidget>
  <srcctl>n</srcctl>
  <ticker>n</ticker>
  <childlock>n</childlock>
  <audiomute>n</audiomute>
  <videomute>n</videomute>
  <dcont>y</dcont>
  <widgetname>IME Tutorial</widgetname>
  <description>Example of how to use the IME feature</description>
  <width>960</width>
  <height>540</height>
  <author>
    <name>Samsung Electronics Co. Ltd.</name>
    <email></email>
    <link>http://www.sec.com</link>
    <organization>Samsung Electronics Co. Ltd.</organization>
  </author>
</widget>
```

The following settings are used:

<fullwidget>y</fullwidget>

Makes the application run in full screen mode. This affects what keys are registered by default.

<type>user</type>

Enables the user application feature for testing on a real TV. This tag has no effect on the emulator.

3. Add ime_sample.js, with the following code:

```
function func_onLoad() {
  alert("func_onLoad begin...");
  widgetAPI = new Common.API.Widget();
  keyc = new Common.API.TVKeyValue();
  var pluginAPI = new Common.API.Plugin();
  widgetAPI.sendReadyEvent();
}

function func_onUnload () {
  alert("onUnload ======================================================");
}
```

4. Start the SDK emulator. If the alert() : func_onLoad begin... message appears in the log manager, the application was successfully created. The provided layout, consisting of several text boxes, appears on the screen. This can be viewed when testing your application on a TV.

Using the IME for Basic Text Entering

To add the IME feature to the application for entering text in the input boxes,

1. Add the following code to ime_sample.js:

```
function func_onLoad() {
    document.getElementById("searchText1").setAttribute("maxlength", 5);
 _g_ime.init("en","2_35_259_12","USA","","us");
 ime = new IMEShell(id, imeReady, 'en');
}
```

This class creates an IMEShell object for the specified HTML input element. Note that the IME feature supports HTML <input> elements only; other kinds of elements are not supported. The callback function imeReady() is called when the IME object has been fully created. Creation of an IME object is asynchronous, and no IME methods should be called until the callback has been received. In the callback, the Main object is notified of this HTML input element being ready. In the code above, the third parameter to IMEShell() is used to manually specify that the IME should work in the English language. This is done for the correct operation on the emulator. If this parameter is missed out, the IME library automatically detects the language used on the device. However, this only works on a real Smart TV device.

2. Add the following code to the definition of the var Main object.

```
elementIds: ["plainText", "passwordText", "maxText"],
inputs: [null, null, null],
ready: [false, false, false]
```

3. Add the following code in the func_onLoad function to create the input objects, and register the key handling for the IME to work correctly.

```
pluginAPI.registIMEKey();
```

Registration is done for all the keys used by the IME. Each key can also be registered individually. Registering the number keys disables the usual TV behaviour of these keys, which is changing the TV channel. To restore this behaviour when the IME is no longer needed, call the unregistIMEKey function to unregister all the keys, or unregistKey function to unregister each key individually. For more details of key registration functions, see Handling Remote Control Key Events.

4. Add the following code to complete the ime_sample.js file.

```
widgetAPI = new Common.API.Widget(),
Keyc = new Common.API.TVKeyValue();

ime = new IMEShell("searchText1", ime_init_text, this);
if (! ime){
    alert("object for IMEShell create failed", 3);
}

ime2 = new IMEShell("searchText2", ime_init_passwd, this);
if (! ime2){
    alert("object for IMEShell create failed", 3);
}

ime3 = new IMEShell("searchText3", ime_init_id, this);
if (! ime3){
    alert("object for IMEShell create failed", 3);
}
```

An input object is created for each of the HTML input elements used, and the readiness for use of each input object is tracked. When all the input objects are ready, the focus is set to the input element with the ID plainText. All the input objects have to be ready before setting the initial focus, as the IME object is needed to receive a focus event. When the IME object receives the focus event, it is displayed and characters can be typed into the first input box.

The capability to change the focus to a different input box while running is added. For now, change the input box being used by changing the code at the end of ime_sample.js, to set focus on a different element ID.

The IME displays and enters the characters used in the world region of the TV device. For example, in the United Kingdom, it enters English characters. In Korea, it enters characters from the Hangul alphabet.

Note the difference in behaviour with each of the input boxes. The **plainText** input box and the **maxText** input box are both of

text type. Key input using the IME with these input boxes is in the style of a mobile telephone - each numeric key press cycles between the different letters on the key. If the user waits for a short time, or presses a different numeric key, the letter currently displayed is accepted and the cursor position moves on to the next letter. It is also possible to move the cursor position left and right, and erase the letters. On-screen help is displayed in the current language of the TV set.

In the **passwordText** input box, of the password type, the letters are not visible. For security, they are replaced with the '*' character. This is the expected behaviour of an HTML input box with this type. Because each letter entry is hidden, it is not possible to use mobile-phone style text entry with this type of a text box. In this case, each numeric key corresponds to a single letter, and the user has to select different pages to access all the possible letters and symbols. For security, there is no animation to show which key has been pressed for password entry. On-screen help is also displayed.

Switching focus with IME

1. Change the Input constructor as below to accept new parameters.

```
var Input = function (id, previousId, nextId) {
...
```

2. Add the following code in the Input constructor:

```
var previousElement = document.getElementById(previousId),
  nextElement = document.getElementById(nextId);
```

3. Add the following line inside the imeReady() function.

```
installFocusKeyCallbacks();
```

4. Add the following new function inside the Input constructor.

```
var installFocusKeyCallbacks = function () {
  imeobj.setKeyFunc(tvKey.KEY_UP, function (keyCode) {
    previousElement.focus();
    return false;
  });
  imeobj.setKeyFunc(tvKey.KEY_DOWN, function (keyCode) {
    nextElement.focus();
    return false;
  });
  imeobj.setKeyFunc(tvKey.KEY_RETURN, function (keyCode) {
    widgetAPI.blockNavigation();
    return false;
  });
  imeobj.setKeyFunc(tvKey.KEY_EXIT, function (keyCode) {
    widgetAPI.blockNavigation();
    return false;
  });
}
```

   Some callback functions that the IME object calls when certain keys are pressed are added. Handlers for the **UP** and **DOWN** keys are added since these keys are used for changing the focus. The key code is passed as a parameter. The IME object is notified of all key presses and this key callback mechanism is offered for the convenience of the developer. Functionality other than just changing the focus can be added in the callback as needed. In this application, false is returned as the IME need not take any action for this key - it switches to a new IME instance.

5. To pass the correct element IDs for the next and previous element to the Input constructor, update the function createInputObjects() as follows:

```
function createInputObjects() {
    var index,
        previousIndex,
        nextIndex;

    for (index in this.elementIds) {
        previousIndex = index - 1;
        if (previousIndex < 0) {
            previousIndex = Main.inputs.length - 1;
        }
        nextIndex = (index + 1) % Main.inputs.length;
        Main.inputs[index] = new Input(this.elementIds[index],
                this.elementIds[previousIndex], this.elementIds[nextIndex]);
    }
}
```

The index of the next and previous element in the elementIds array is calculated (taking care to wrap around correctly at the first and last element) and the correct element IDs are passed to the Input constructor.

The focused input object can be changed by pressing the **UP** or **DOWN** down keys. The display of the IME changes depending on the box selected - the password box has a different display from the others.

On the input box of the tutorial application, **RETURN** and **EXIT** are blocked. Those events should be handled exclusively.

Notifications and control functions with IME

Notifications

To implement notifications by callback functions for events other than certain key presses:

1. Add the following line inside the function imeReady():
   ```
   installStatusCallbacks();
   ```
2. Add the following function definitions inside the Input constructor:
   ```
   var installStatusCallbacks = function () {
       imeobj.setKeySetFunc('qwerty');
       imeobj.setAnyKeyFunc(onAnyKey);
       imeobj.setMaxLengthFunc(onMaxLength);
       imeobj.setPrevEventOnLeftFunc(onLeft);
       imeobj.setOnCompleteFunc(onComplete);
       imeobj.setEnterFunc(onEnter);
       imeobj.setKeyFunc(tvKey.KEY_INFO, onInfoKey);
       imeobj.setKeypadChangeFunc('12key', move12KeyPosition);
       imeobj.setKeypadChangeFunc('qwerty', moveqwertyPosition);
   };
   ```

The following IME functions are called to set the callbacks:

setAnyKeyFunc

the specified callback function is called each time any key is pressed.

setMaxLengthFunc

the specified callback function is called when the maximum length of the HTML input element is reached.

setPrevEventOnLeftFunc

the specified callback function is called when the left key is pressed, and the cursor position is at the start of the input box. A message is displayed when this callback is triggered.

setOnCompleteFunc

the specified callback function is called when the entry of one character is completed. For this application, only a message is displayed.

setEnterFunc

the specified callback function is called when the **ENTER** key is pressed on the input box. A message on the HTML page is displayed when this callback is triggered.

setKeyFunc

the specified callback function is called when the specified key is pressed. This callback is implemented in Switching focus with IME to change the focus. A new callback for the **INFO** key has been added, as an example of key handling being customised for the requirements of a particular application. A message is displayed as response to this callback. To allow IME to take action if needed, true is returned.

Control Functions

This section describes controlling the IME functions provided by the IME common module.

To set a specific string

> Note
>
> To set the contents of an input box to a specific string, do not use code like the following. Character entry with IME is complex (especially for the characters used in some Asian countries) and the IME object is not able to keep track of text added to an input object directly. The results of using the code below can be unpredictable.

```
document.getElementById("plainText").value = "Hello world";
```

To set the input box to a specific string compatible with IME, add the following line in the function onInfoKey before the return statement.

```
imeobj.setString("Hello world");
```

IME replaces the text in the input box with the specified string. After this, further input can continue without problems.

To set start keypad layer of IME

By default, the IME keypad start layer is QWERTY. The IME has two keypad layers - OSK QWERTY and OSK 12key. The options are qwerty and 12key strings.

```
imeobj.setKeySetFunc('qwerty');
```

or

```
imeobj.setKeySetFunc('12key');
```

To change the position of the IME display

By default, the IME window is positioned near the top of the display, and right horizontally. However, each different application may require it to be displayed in a different position, depending on the layout of the current screen. The position of the IME display can be changed by calling a member function of the IME object.

Add the following code at the start of the installStatusCallbacks function.

```
imeobj.setKeypadPos(428, 80);
imeobj.setQWERTYPos(428, 80);
```

This changes the position of the IME further to the right on the display.

To set change keypad callback function of IME

This callback function is called when the keypad changes to 12key or qwerty. The position of the HTML Inputbox div is changed when the IME keypad change event occurs. The options are qwerty and 12key strings and a callback function. To set a callback function that is called when keypad changes to 12key or qwerty, add the following code at the start of installStatusCallbacks function:

```
imeobj.setKeypadChangeFunc('12key', move12KeyPosition);
imeobj.setKeypadChangeFunc('qwerty', moveqwertyPosition);

function move12KeyPosition(arg) {
    if (curWidget.height == '720') {
        document.getElementById("search").style.right = 384;
        document.getElementById("search").style.top = 106;
    } else {
        document.getElementById("search").style.right = 286;
        document.getElementById("search").style.top =  82;
    }
}

function moveqwertyPosition(arg) {
    if (curWidget.height == '720') {
        document.getElementById("search").style.right = 714;
        document.getElementById("search").style.top =  106;
    } else {
        document.getElementById("search").style.right = 534;
        document.getElementById("search").style.top =  82;
    }
}
```