# IME Tutorial

Version 1.03

Samsung Smart TV

## Purpose of Document

This document is a tutorial that is aimed at widget developers for Samsung Smart TV, and will show how to use the Input Method Editor feature of the Samsung widget service. This feature is needed to allow input of text characters using only the numeric keys available on the remote control. It is recommended that you read Widget Development Guide for Samsung Smart TV (hereinafter referred to as "the Guide") first. This document provides references to the Guide for your understanding.

## Target Readers

This document is aimed at programmers who have used web development languages such as HTML, CSS and JavaScript, and will be even more helpful for those who have web development experience. This document has been written on the assumption that readers have already read the Guide.

# 1. Overview

This tutorial will show the steps needed to add text input capability to a widget, using the IME feature of the widget manager. This will be done by creating a simple widget with a text input box and a password input box, and enabling user input with a virtual keyboard display. This widget will also demonstrate the optional notifications available from the IME feature, by displaying various status messages.

## 1.1. Sample code

Code for two sample widgets is provided with this document. One sample widget is the one described in this document. A second sample widget (index_simple.html, ime_sample.js) is provided as a simple example.

# 2. Introduction

In order to develop this IME example widget, developers are required to have background knowledge such as HTML, JavaScript and CSS. There will be no explanation about HTML, JavaScript or CSS provided in this document. Developing widgets suited for TVs is different from developing widgets on PCs in several aspects. You can get more information on this from the Guide.

Together, we're going to write code for each development stage and you can see how each part of the code is completed.

## 2.1. Development environment

You are going to use the Samsung TV Widget SDK ("SDK") made by Samsung to create your widget. With use of the emulator provided with the SDK, you can operate your widget before actually putting it in your TV.

It is also possible to run the widget on a real Samsung Smart TV device (for example, a TV) using the "User Widget" feature. For details of this process please see the document "User Widget".

## 2.2. Files needed for the widget

This tutorial starts with the HTML already created, along with a CSS file that takes care of formatting and positioning. This tutorial will focus on how to add IME support to a page layout that has already been created – it is assumed that the reader is familiar with creating HTML page layouts.

Please begin by copying all files from the supplied widget source archive into the SDK widget folder, except for those in the Javascript folder.

# 3. Overall structure

In this section, we're going to examine the structure of the video widget that we will create.

## 3.1. Design

The design of the widget will be very simple – because almost all of the complexity of international character entry is implemented already, in the IME feature of the widget manager. We will create a single JavaScript file (Main.js), a single HTML file (index.html) and a single CSS file (Main.css).

## 3.2. Directory structure

| Directory | Description |
|---|---|
| CSS | CSS files. |
| Javascript | Javascript files |

# 4. Creating the Basic Widget

## 4.1. Initial Widget setup

Start the SDK for Samsung TV widgets. Create a new widget using the following config.xml file:

```xml
<?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>
```

Please note the following settings that we have used:

`<fullwidget>y</fullwidget>` - this means that the widget will run in full screen mode. This affects what keys are registered by default.

`<type>user</type>` - this enables the user widget feature for testing on a real TV set. This tag has no effect on the emulator.

Add Main.js in the Javascript folder, with contents as follows:

```
var widgetAPI = new Common.API.Widget();
var tvKey = new Common.API.TVKeyValue();

var Main =
{
}

Main.onLoad = function()
{
	alert("Main.onLoad()");
	widgetAPI.sendReadyEvent();
}

Main.onUnload = function()
{
	alert("Main.onUnload()");
}
```

If you have unzipped the provided files, the widget should already have an HTML index page.

Now, start the SDK emulator. If you see the message 'alert() : Main.onLoad()' in the log manager, that means you have successfully created the widget. You should be able to see the provided layout on the screen, consisting of several text boxes.

You should also be able to view this on the TV with the User Widget upload feature.

## 5. Basic Text Entry using the IME

In this section we will add the IME feature to the widget, which will enable us to enter text in the input boxes.

Add the following class definition to Main.js:

```
var Input = function(id)
{
    var imeReady = function(imeObject)
    {
        Main.ready(id);
    }

    var ime = new IMEShell(id, imeReady, 'en');
    var element = document.getElementById( id );
}
```

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 element 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, we notify the Main object that this HTML input element is ready.

In the code above, we have used the third parameter to IMEShell() to manually specify that the IME should work in the English language. This is done for correct operation on the emulator. If this parameter is missed out, the IME library will automatically detect the language used on the device. However, this only works on a real Samsung Smart TV device.

Add the following to the definition of the object var Main:

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

Now we will create the Input objects, and register to handle the keys needed for the IME to work correctly. Add the following inside the function Main.onLoad:

```
    this.createInputObjects();

    widgetAPI.registIMEKey();
```

Note that here we are registering for the widget to handle all keys used by the IME. This is a short-cut function that is useful for widgets that use the IME. The widget developer can also choose to register each key individually. Developers should be aware that by registering the number keys, they are disabling the usual TV behaviour when pressing these keys, which is to change to a TV channel. To restore this behaviour when the IME is no longer needed, the widget can call the function unregistIMEKey to un-register all the keys, or alternatively unregistKey to un-register each key individually. For more details of these key registration functions, please see the document "Develoment Guide for Samsung widget service".

And add the following code to complete Main.js:

```
var widgetAPI = new Common.API.Widget();
var tvKey = new Common.API.TVKeyValue();

Main.createInputObjects = function()
{
    for (var index in this.elementIds)
    {
        Main.ime[index] = new Input( this.elementIds[index]);
    }
}

Main.ready = function(id)
{
    var ready = true;

    for (var i in Main.elementIds)
    {
        if (Main.elementIds[i] == id)
        {
            Main.ready[i] = true;
        }

        if (Main.ready[i] == false)
        {
            ready = false;
        }
    }

    if (ready)
    {
        document.getElementById("plainText").focus();
    }
}
```

This code creates an Input object for each of the HTML input elements that we are using. It also tracks when each one is ready to be used. When all of them are ready, it sets the focus to the input element with ID "plainText". We have to wait until everything is ready before setting the initial focus – because we need the IME object to receive a focus event. When the IME object receives the focus event, it will be displayed and we can type characters into the first input box.

So far, there is no way to change the focus to a different input box while we are running. We will add this capability in the next section. For now, you can change which input box we are using by changing the code at the end of Main.js, so that it focuses to a different element ID.

The IME will display and enter the characters used in the world region of the TV set. For example, in the UK it will enter English characters. In Korea it will enter 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 type "text". 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 that are shown on that 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 letters. On-screen help is displayed in the current language of the TV set.

The "passwordText" input box behaves differently, because the type is "password". The letters are not visible in the input box, 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 style of text box. So in this case, each numeric key corresponds to only one letter, and the user has to select different pages to access all the possible letters and symbols. Also for security, there is no animation to show which key has been pressed for password entry. On-screen help is also displayed in this case.

## 6. Switching the focus

Now we will add code to switch the focus to different HTML input boxes, so that the user can enter text in each of them.

Change the Input constructor so that it accepts some new parameters:

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

Add the following in the Input constructor, after the definition of var element:

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

Add the following inside the function imeReady():

```
        installFocusKeyCallbacks();
```

And add the following new function inside the Input constructor:

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

We have added some callback functions that the IME object will call when certain keys are pressed. In this case we have added handlers for the up key and the down key, and we use them to change the focus. The key code is passed as a parameter. The IME object is already notified of all key presses, so it offers this key callback mechanism for the convenience of the developer. Of course, depending on the requirements of a particular widget, we could do more in the callback than just change the focus. But for this widget we will keep it simple. We return false because we don't want the IME to take any action for this key – we are switching to a new IME instance.

All that remains is to pass the correct element IDs for the next and previous element to the Input constructor. Update the function Main.createInputObjects() as follows:

```
Main.createInputObjects = function()
{
    for (var index in this.elementIds)
    {
        var previousIndex = index - 1;
        if (previousIndex < 0)
        {
            previousIndex = Main.inputs.length - 1;
        }
        var nextIndex = (index + 1) % Main.inputs.length;

        Main.inputs[index] = new Input( this.elementIds[index], this.elementIds[previousIndex],
this.elementIds[nextIndex] );
    }
}
```

This just calculates the index of the next and previous element in the elementIds array (taking care to wrap around correctly at the first and last element). Then it passes the correct element IDs to the Input constructor.

Now it is possible to change the focussed input object by pressing the up or down keys. The display of the IME changes depending on which box we have selected - the password box has a different display to the other two.

On input box, return and exit is blocked. So you should return and exit manually when return and exit is pressed.

# 7. Notifications and control functions

## 7.1. Notifications

So far we have installed callback functions for certain key presses. The IME can also make notifications by callback functions for other events.

Add the following inside the function imeReady():

```
        installStatusCallbacks();
```

Add the following function definitions inside the Input constructor:

```
    var installStatusCallbacks = function()
    {
        ime.setAnyKeyFunc(onAnyKey);
        ime.setMaxLengthFunc(onMaxLength);
        ime.setPrevEventOnLeftFunc(onLeft);
        ime.setOnCompleteFunc(onComplete);
        ime.setEnterFunc(onEnter);
        ime.setKeyFunc(tvKey.KEY_INFO, onInfoKey);
    }

    var onAnyKey = function(keyCode)
    {
       alert("a key pressed");
    }

    var onLeft = function()
    {
        Main.showMessage("Left key pressed at start of " + element.id);
    }

    var onComplete = function()
    {
        Main.showMessage("Letter entry completed in " + element.id + ", text is " + element.value);
    }
    var onEnter = function(string)
    {
        Main.showMessage("Enter key pressed in " + element.id + ", string is " + string);
    }

    var onInfoKey = function(keyCode)
    {
        Main.showMessage("Info key pressed in " + element.id + ", key code is " + keyCode + ", text is " +
element.value);

        return true;
    }
```

We have called the following functions of IME to set callbacks:

- setAnyKeyFunc – the specified callback function will be called each time a key is pressed, no matter what the key.
- setMaxLengthFunc – the specified callback function will be called when the maximum length of the HTML input element is reached.
- setPrevEventOnLeftFunc – the specified callback function will be called when the left key is pressed, and the cursor position is at the start of the input box. We display a message when this callback is triggered.
- setOnCompleteFunc - the specified callback function will be called when the entry of one character is completed. For this simple widget we will just display a message.
- setEnterFunc - the specified callback function will be called when the enter key is pressed on the input box. We will display a message on the HTML page when this callback is triggered.
- setKeyFunc - the specified callback function will be called when the specified key is pressed. We used this callback in the previous section to change the focus. Here we have added a new callback for the Info key, as an example of how key handling can be customised for the requirements of a particular widget. We will respond to this callback by displaying a message. We return true as we don't want to prevent the IME from taking action if it needs to.

For more details on these functions, see section 0.

## 7.2. Control functions

This section will describe how to control the IME with some more member functions. For more details on these functions, see section 0.

### 7.2.1. Set a Specific String

If a widget needs to set the contents of an input box to a specific string, code like the following should not be used:

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

Character entry with an IME can be complex (especially for the characters used in some Asian countries). Because of this complexity, the IME object is not able to keep track of text added to an input object directly. So the results of using the above code can be unpredictable.

We will add code to set the input box to a specific string, in a way that is compatible with the IME. Add the following code in the function onInfoKey, before the return statement:

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

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

### 7.2.2. Change the position of the IME display

By default, the IME window is positioned near the top of the display, and is centred horizontally. But each different widget may require this 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.

Like Keypad position, Word suggestion window (window that appears in T9 mode) can be moved in same way

Add the following code at the start of function installStatusCallbacks:

```
        ime.setKeypadPos(320, 80);
        ime.setWordBoxPos(320, 80);
```

This will change the position of the IME to a more central position on the display.

## 7.3. IME Function Reference

The following functions are all member functions of the IME object.

| ***Function*** **IMEShell** | |
|---|---|
| Create IME object | |
| **Syntax** | IMEShell(inputObjId, callbackFunc, IMELanguage) |
| **Parameter** | • inputObjId: id of the input object<br>• callbackFunc: called when the IME object has been fully created<br>• IMELanguage : language code to use (optional, default: match the device language setting) |
| **Return Value** | IME object |
| **Remarks** | None |
| **Example** | ime = IMEShell('input_obj_id', callbackFunc, 'en'); |

| ***Function*** **getInputObj** | |
|---|---|
| Get input object from IME object | |
| **Syntax** | getInputObj () |
| **Parameter** | • None |
| **Return Value** | input object from IME object |
| **Remarks** | You can use this function when you need to get the HTML input object attached to an IME object. |
| **Example** | InputObj = ime.getInputObj(); |

***Function***

## setAnyKeyFunc

Set a callback function that will be called each time a key is pressed, no matter what the key.

| | |
|---|---|
| **Syntax** | setAnyKeyFunc(func) |
| **Parameter** | • func: callback function with following parameters:<br>○ keyCode: code of the key that was pressed<br>○ No return value |
| **Return Value** | None |
| **Remarks** | The function will be called after key handle process. |
| **Example** | `var callback = function(keyCode)`<br>`{`<br>`    alert("Key pressed, code " + keyCode);`<br>`}`<br>`ime.setAnyKeyFunc(callback);` |

***Function***

## setKeyFunc

Set a callback function that will be called when when a specific key is pressed.

| | |
|---|---|
| **Syntax** | setKeyFunc(keyCode, func) |
| **Parameter** | • keyCode: code for the specific key that should trigger the callback<br>• func: callback function with the following parameters:<br>○ keyCode: the key code that was pressed<br>○ Return value:<br>▪ true – IME will also handle this key if it needs to.<br>▪ false – IME will not handle the key at all |

| **Return Value** | none |
|---|---|
| **Remarks** | Some keys are usually handled by the IME itself. For example, numeric keys are used for text input, and the left and right keys are used for moving the cursor. It is possible to register callbacks for these special keys using setKeyFunc(). The IME will call the callback function first, before handling the key itself. If the callback function returns true (or null), it will continue as normal to handle the key and take the usual action (like entering text or moving the cursor). If the callback function returns false, the IME will not take action for this key. This allows the client to block certain default behaviour of the IME (like cursor movement for example). This feature should be used with care as it prevents normal functioning of the IME. |
| **Example** | <pre>var tvKey = new Common.API.TVKeyValue();<br>var callback = function(keyCode)<br>{<br>    if (keyCode == tvKey.KEY_LEFT)<br>    {<br>        alert("Blocking left key");<br>        return false;<br>    }<br>    else<br>    {<br>        alert("Key code received: " + keyCode + ", allow IME to process");<br>        /* Do some action here based on keyCode */<br>        return true;<br>    }<br>}<br>ime.setKeyFunc(tvKey.KEY_LEFT, callback);<br>ime.setKeyFunc(tvKey.KEY_INFO, callback);<br>ime.setKeyFunc(tvKey.KEY_MENU, callback);</pre> |

| ***Function*** **setKeypadPos** | |
|---|---|
| Set the position for the display of the IME keypad window | |
| **Syntax** | setKeypadPos(x, y, z) |
| **Parameter** | • x: horizontal offset of the IME keypad display from the left edge of the screen, in pixels<br>• y: vertical offset of the IME keypad display from the top edge of the screen, in pixels<br>• z: z-index offset of the IME keypad. This value deside CSS z-index of the keypad. This parameter is optional. (default value: 9) |
| **Return Value** | None |
| **Remarks** | None |
| **Example** | `ime.setKeypadPos(500, 40, 9);` |

| ***Function*** <br> **setWordBoxPos** | |
|---|---|
| Set the position for the display of the IME word suggestion window | |
| **Syntax** | setWordBoxPos (x, y, z) |
| **Parameter** | • x: horizontal offset of the IME wordbox display from the left edge of the screen, in pixels<br>• y: vertical offset of the IME wordbox display from the top edge of the screen, in pixels<br>• z: z-index offset of the IME wordbox. This value deside CSS z-index of the keypad. This parameter is optional. (default value: 9) |
| **Return Value** | None |
| **Remarks** | None |
| **Example** | `ime. setWordBoxPos (100, 40, 9);` |

| ***Function*** <br> **setEnterFunc** | |
|---|---|
| Set a callback function that will be called when the enter key is pressed on the input box. | |
| **Syntax** | setEnterFunc(func) |
| **Parameter** | • func: callback function with the following parameters:<br>○ string: character string currently entered in the input box<br>○ No return value |
| **Return Value** | none |
| **Remarks** | This is commonly used to indicate that entry of text on a form is completed. |
| **Example** | `var callback = function(string)`<br>`{`<br>`    alert("User pressed enter, final text is " + string);` |

| | |
|---|---|
| | }<br>ime. setEnterFunc(callback); |

| ***Function***<br>**setMaxLengthFunc** | |
|---|---|
| Set a callback function that will be called when the maximum length of the HTML input element is reached. | |
| **Syntax** | setMaxLengthFunc(func) |
| **Parameter** | • func: callback function with no parameters or return value |
| **Return Value** | none |
| **Remarks** | The maximum length is specified as the HTML property "maxlength". This callback will not be triggered if a "maxlength" property has not been set for the input element. |
| **Example** | var callback = function()<br>{<br>    alert("Maximum length reached");<br>}<br>ime.setMaxLengthFunc(callback); |

| ***Function***<br>**setPrevEventOnLeftFunc** | |
|---|---|
| Set a callback function that will be called when the left key is pressed, and the cursor position is at the start of the input box. | |
| **Syntax** | setPrevEventOnLeftFunc(func) |
| **Parameter** | • func: callback function with no parameters or return value |
| **Return Value** | none |
| **Remarks** | none |

| | |
|---|---|
| **Example** | var callback = function()<br>{<br>    alert("Left key pressed at left side of input box");<br>}<br>ime.setPrevEventOnLeftFunc(callback); |

| ***Function*** | |
|---|---|
| **setOnCompleteFunc** | |
| Set a callback function that will be called when the entry of one character is completed. | |
| **Syntax** | setOnCompleteFunc(func) |
| **Parameter** | • func: callback function with no parameters or return value |
| **Return Value** | none |
| **Remarks** | This function will be called when:<br>1. Right key pressed<br>2. Delete key pressed (delete key is indicated by IME display)<br>3. A space character is added<br>4. Entry of a character is completed<br>For example, in a text style input box, the user may press a numeric key 3 times to choose the correct character. This callback waill not be triggered until a short time is elapsed after this, or the user presses a different numeric key – this means that the user has chosen to keep this character in the string, and so the entry of one character is completed. This callback can be used, for example, to create a list of suggestions for completing entries in a search box. |
| **Example** | var callback = function()<br>{<br>    alert("Entry of a character is complete");<br>}<br>ime. setOnCompleteFunc(callback); |

| ***Function*** **setBlockSpace** | |
|---|---|
| Block space | |
| **Syntax** | setBlockSpace(block) |
| **Parameter** | • block (optional, default: true)<br>○ true – IME will not allow the space character to be entered<br>○ false – IME will allow the space character to be entered |
| **Return Value** | None |
| **Remarks** | This function can be used to block entry of the space character in an input box. For example, this can be useful when entering a user id or password. |
| **Example** | `ime.setBlockSpace (true);` |

| ***Function*** **setString** | |
|---|---|
| Set the string for display in the HTML input box associated with the IME object | |
| **Syntax** | setString(string) |
| **Parameter** | • string: character string to display in the input box |
| **Return Value** | none |
| **Remarks** | This function should be used instead of directly setting the value property of the HTML input box (element.value="…"). Directly setting the value property will cause unexpected behaviour in the IME. |
| **Example** | `ime.setString("Hello world");` |

## 8. Concluding remarks

This tutorial has explored the main functions of the IME feature of the widget manager. It has shown how to enter text in both the standard and password style of input box. It has also shown how to receive notifications from the IME when certain events happen, and how to update text from JavaScript code. Using the IME and its API, text entry can easily be added to widgets to enable search, login and other kinds of application features that require text entry in forms.