Tizen Input Service Framework Overview

Ji-hoon Lee
Samsung Electronics
Contents

• Introduction to Input Service Framework
• Architecture and Event Flow
• Utilizing Input Service Framework
Introduction to Input Service Framework
What Is Input Service Framework?

- Provides **text input-related** services
- **Plays major role in:**
  - Multilingual text input support
  - Text input support for devices without physical keyboard
Multilingual Text Input

• Some languages need complex processing when typing text
• Support for the following is needed:
  • Pre-edit strings: Strings that have not been fixed
  • Commit strings: Strings that are finalized and being appended
  • Candidate window: Window that lists and lets you to select among the possible conversions

Type “doumoarigatou” Press Space Press left or right key Press Enter
Input Service Framework Basics

• Based on open source SCIM 1.4.7
  • SCIM = Smart Common Input Method
  • Client/Server architecture
  • Dynamically loadable plug-in module support
  • Provides a simple and powerful programming interface
  • Widely used in many Linux distributions

• Added support for mobile devices
  • Application-side interface for interacting with S/W keyboard engines
  • S/W keyboard engine-side interface for full-featured text input services
Benefits of Using Open Source SCIM

- Powered by open source keyboard engines
- Tizen keyboard supports 40+ languages including CJK languages, without any language-specific code
- Korean language XML layout example of the Tizen keyboard:

```xml
<row x="4" y="10">
  <key use_magnifier="true" longkey_magnifier="true" long_key_value="1" multitouch_type="settle_previous">
    <label>
      <rec shift="off" multi="0">ㅂ</rec>
      <rec shift="on" multi="0">ㅃ</rec>
      <rec shift="loc" multi="0">ㅃ</rec>
      <rec multi="1">1</rec>
    </label>
    <key_value>
      <rec auto_upper="true">q</rec>
    </key_value>
  </key>
</row>
```
Architecture and Event Flow
Architecture Diagram

Native application

Web application

Native framework

Web framework

EFL

Ecore IMF

X11

Kernel

Legend

- ISF
- Process
- IPC
- Plug-in
- Plug-in module

Input service

manager

Panel

Input service engine

S/W keyboard engine

H/W keyboard engine

H/W keyboard engine

...
Ecore_IMF

- Interface for interacting with the input service framework
- isf-imf-context: plug-in module connecting the Ecore_IMF interface and input service framework
Input Service Manager

- Process handling H/W key events
- Loads all available H/W keyboard engines, and requests the currently selected one to handle the given key event generated by the H/W keyboard
Input Service Engine

- Process that loads the S/W keyboard engine
- The S/W keyboard engine does not necessarily have to be an actual keyboard (handwriting module, Wi-Fi text receiver)
- Existing input service engine process is terminated and a new process is launched every time a different S/W keyboard engine is selected
Panel

- Mediator between an application and input service engine
- When an application requests something to an input service engine, the panel receives the event and delivers it to the currently selected input service engine
- UI representation module of input service framework (such as candidate window)
S/W Keyboard Engine Life-cycle

- **Input service engine process launched when:**
  - System starts
  - S/W keyboard engine is selected

- **Exit request message received when:**
  - H/W keyboard is attached
  - S/W keyboard engine is unselected
Event Flow – H/W Keyboard

Kernel / X11

Editable widget

Ecore IMF

isf-context

KeyPress

X KeyEvent

filter_event()

isf_imf_context_filter_event()

PROCESS_KEY_EVENT

process_key_event()

UPDATE_LOOKUP_TABLE

DISPLAY_LOOKUP_TABLE

Display candidate window

Select candidate item

Update pre-edit string callback

Display pre-edit string

Commit string callback

Append committed string

Application process

Input service manager process

H/W Keyboard Engine

Panel process

Function call

IPC message

TIZEN

DEVELOPER CONFERENCE
2013 SAN FRANCISCO
Event Flow – S/W Keyboard

Application process
- Editable widget
- Ecore
- isf-imf-context

Input service engine process
- scim-helper-launcher
- S/W keyboard engine

Panel process
- isf-panel-efl

Kernel / X11

Editable widget

Ecore

isf-imf-context

Screen touch

X mouse press event

Select candidate item

Update pre-edit string callback

Display pre-edit string

Commit string callback

Append committed string

UPDATE_LOOKUP_TABLE

SHOW_LOOKUP_TABLE

UPDATE_PREEDIT_STRING

SELECT_CANDIDATE

select_candidate()

COMMIT_STRING

Function call

IPC message
Event Flow – Dummy S/W Keyboard

Application process
- Input service engine process
  - scim-helper-launcher
  - S/W Keyboard Engine

Input service manager process

Kernel / X11
- Editable widget
- Ecore IMF
- isf-imf-context

Screen touch
- X key press event
  - filter_event()
  - isf_imf_context_filter_event()
  - XSendEvent()

Exactly the same steps as the H/W Keyboard case

SEND_KEY_EVENT

Identify button that was pressed
Utilizing Input Service Framework
Application Side Core APIs (1/5)

- Core API provides the common interface for both native and Web applications
- Native and Web frameworks have their own mappings for using Core APIs
Application Side Core APIs (2/5)

- Show and hide S/W keyboard
- Handle commit and pre-edit strings

```c
static Eina_Bool _preedit_cb (void *data, int type, void *event)
{
    ecore_imf_context_preedit_string_get (imf_context, &preedit_string, &len);
    printf ("entry get preedit string: \%s", preedit_string);
    return ECORE_CALLBACK_RENEW;
}
static Eina_Bool _commit_cb (void *data, int type, void *event)
{
    Ecore_IMF_Event_Commit *ev = (Ecore_IMF_Event_Commit *) event;
    printf ("entry get commit string: \%s", (char *)(ev->str));
    return ECORE_CALLBACK_RENEW;
}
ecore_event_handler_add (ECORE_IMF_EVENT_PREEDIT_CHANGED, _preedit_cb, NULL);
ecore_event_handler_add (ECORE_IMF_EVENT_COMMIT, _commit_cb, NULL);
ecore_imf_context_input_panel_show (imf_context);
ecore_imf_context_input_panel_hide (imf_context);
```
Application Side Core APIs (3/5)

- Layouts and variations

<table>
<thead>
<tr>
<th>Layouts</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td></td>
</tr>
<tr>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>EMAIL</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>PHONENUMBER</td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>MONTH</td>
<td></td>
</tr>
<tr>
<td>NUMBERONLY</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBERONLY_NORMAL</td>
<td></td>
</tr>
<tr>
<td>NUMBERONLY_SIGNED</td>
<td></td>
</tr>
<tr>
<td>NUMBERONLY_DECIMAL</td>
<td></td>
</tr>
<tr>
<td>NUMBERONLY_SIGNED_AND_DECIMAL</td>
<td></td>
</tr>
</tbody>
</table>

```c
ecore_imf_context_input_panel_layout_set(imf_context, ECORE_IMF_INPUT_PANEL_LAYOUT_EMAIL);
```
Application Side Core APIs (4/5)

- Auto-scrolling, status callbacks
- Auto-capitalization

```c
static void _input_panel_state_cb (void *data, Ecore_IMF_Context *ctx, int value) {
    if (value == ECORE_IMF_INPUT_PANEL_STATE_SHOW) {
        printf ("[%s] Input panel is shown. ctx : %p\n", __func__, ctx);
    } else if (value == ECORE_IMF_INPUT_PANEL_STATE_HIDE) {
        printf ("[%s] Input panel is hidden. ctx : %p\n", __func__, ctx);
    }
}

static void _input_panel_resize_cb (void *data, Ecore_IMF_Context *ctx, int value) {
    ecore_imf_context_input_panel_geometry_get (ctx, &x, &y, &w, &h);
    printf ("[%s] x : %d, y : %d, w : %d, h : %d\n", __func__, x, y, w, h);
}

ecore_imf_context_input_panel_event_callback_add (ic,
    ECORE_IMF_INPUT_PANEL_STATE_EVENT, _input_panel_state_cb, NULL);
ecore_imf_context_input_panel_event_callback_add (ic,
    ECORE_IMF_INPUT_PANEL_GEOMETRY_EVENT, _input_panel_resize_cb, NULL);

ecore_imf_context_autocapital_type_set (ic, ECORE_IMF_AUTOCAPITAL_TYPE_SENTENCE);
```
Application Side Core APIs (5/5)

• Return key types, enabling, and disabling

```c
ecore_imf_context_input_panel_return_key_type_set(imf_context,
    ECORE_IMF_INPUT_PANEL_RETURN_KEY_TYPE_GO);
ecore_imf_context_input_panel_return_key_disabled_set(imf_context,
    EINA_TRUE);
```
Input Service Engine Side Core APIs (1/5)

- Downloadable S/W keyboard engines can be developed by using native APIs that are mapped to Core APIs
Input Service Engine Side Core APIs (2/5)

- **Send key events**
  - `send_key_event()` works as if a H/W key was pressed
  - Since a H/W key press event accompanies release event, it is better to send key release event right after key press events

```
KeyEvent key_press (SCIM_KEY_A, SCIM_KEY_NullMask);
KeyEvent key_release (SCIM_KEY_A, SCIM_KEY_ReleaseMask);
helper_agent.send_key_event (-1, "", key_press);
helper_agent.send_key_event (-1, "", key_release);

SCIM_KEY_NullMask       = 0,            /**< Key press event without modifier key. */
SCIM_KEY_ShiftMask      = (1<<0),       /**< The Shift key is pressed down */
SCIM_KEY_CapsLockMask   = (1<<1),       /**< The CapsLock key is pressed down */
SCIM_KEY_ControlMask    = (1<<2),       /**< The Control key is pressed down */
SCIM_KEY_AltMask        = (1<<3),       /**< The Alt key is pressed down */
...                      
SCIM_KEY_ReleaseMask    = (1<<15),      /**< It's a key release event */
SCIM_KEY_AllMasks       = 0xC0FF        /**< All valid Masks */
```
Input Service Engine Side Core APIs (3/5)

- **Forward key events**
  - In some cases, keys must not be translated by H/W keyboard engines, but forwarded instead of sending them

- **Send commit strings**
  - Even the result is the same, it is better to send key events than commit strings for characters in ASCII code range, since some applications rely on key events

```java
KeyEvent key_press (SCIM_KEY_A, SCIM_KEY_NullMask);
KeyEvent key_release (SCIM_KEY_A, SCIM_KEY_ReleaseMask);
helper_agent.forward_key_event (-1, "", key_press);
helper_agent.forward_key_event (-1, "", key_release);

commit_string (-1, "", "Sample String");
```
Input Service Engine Side Core APIs (4/5)

- **Updating pre-edit strings**
  - We can show and hide current pre-edit string after updating it
  - Pre-edit strings have their attributes, to distinguish their conversion state
  - Pre-edit string has its own cursor and we can update the position of it

```cpp
scim::AttributeList attrs;
scim::WideString str(L"abcd");
attrs.push_back (scim::Attribute (0, 2, scim::SCIM_ATTR_DECORATE, scim::SCIM_ATTR_DECORATE_UNDERLINE));
attrs.push_back (scim::Attribute (2, 2, scim::SCIM_ATTR_DECORATE, scim::SCIM_ATTR_DECORATE_HIGHLIGHT));
helper_agent.update_preedit_string(-1, "", str, attrs);
helper_agent.update_preedit_caret(3);

enum AttributeType
{
    SCIM_ATTR_NONE,///< No attribute.
    SCIM_ATTR_DECORATE,///< A decorate attribute, eg. underline etc.
    SCIM_ATTR_FOREGROUND,///< A foreground color attribute, in RGB format.
    SCIM_ATTR_BACKGROUND,///< A background color attribute, in RGB format.
};
```
Input Service Engine Side Core APIs (5/5)

- Dealing with a candidate window
  - Updating the content of the candidate window
  - Show and hide candidate window
  - Expand and contract the candidate window
  - Set candidate window style

```cpp
helper_agent.set_candidate_style(ONE_LINE_CANDIDATE, FIXED_CANDIDATE_WINDOW);

std::vector<scim::WideString> labels;
scim::CommonLookupTable table;
labels.push_back(scim::WideString(L"1"));
labels.push_back(scim::WideString(L"2"));
table.set_candidate_labels(labels);
table.append_candidate(scim::WideString(L"item1"));
table.append_candidate(scim::WideString(L"item2"));
helper_agent.update_candidate_string(table);

helper_agent.show_candidate_string();
helper_agent.expand_candidate();
```
And More…

- We are still incorporating the features of Core APIs into the Native framework
- For the Core API header files, see:
  - https://review.tizen.org/git/?p=framework/uifw/ecore.git;a=blob;f=src/lib/ecore_imf/Ecore_IMF.h
  - https://review.tizen.org/git/?p=framework/uifw/isf.git;a=blob;f=ism/src/scim_helper.h