In this document, I want to explain the basic step to use new HMI Framework for Wayland Client.

Here is a sample code in wayland source named “simple-egl”, some people who use opengl may know it. I will use it to explain.

- Global architecture
- How-to

*1:homescreen-service(referred as HS), windowmanager-service(referred as WM)
*2:https://cgit.freedesktop.org/wayland/weston/tree/clients/simple-egl.c
# Global architecture

<table>
<thead>
<tr>
<th>Original architecture</th>
<th>New architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apps</strong></td>
<td></td>
</tr>
<tr>
<td><code>simple-egl</code></td>
<td><code>simple-egl</code></td>
</tr>
<tr>
<td><code>libegl libgl</code></td>
<td><code>libhomescreen</code></td>
</tr>
<tr>
<td><code>libegl libgl</code></td>
<td><code>libwindowmanager</code></td>
</tr>
<tr>
<td><strong>Middleware</strong></td>
<td></td>
</tr>
<tr>
<td><code>OpenGl ES</code></td>
<td><code>HS</code></td>
</tr>
<tr>
<td><code>Ivi-shell / desktop-shell</code></td>
<td><code>Opengl ES</code></td>
</tr>
<tr>
<td><code>wayland</code></td>
<td><code>wayland</code></td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td></td>
</tr>
<tr>
<td><code>Linux OS</code></td>
<td><code>Linux OS</code></td>
</tr>
</tbody>
</table>
How-to (new HMI Framework)

These three steps need to do by using new HMI Framework.

- **Launch simple-egl**
  When user click the icon on homescreen, HS will launch “simple-egl”, and send TapShortcut event to application.

- **Get ivi surface id from WM**
  Using requestSurface() API in WM, “simple-egl” can use surface id given by wm for ivi surface.

- **Display surface**
  Using activeSurface() API in WM, “simple-egl” can display its surface on the monitor.
How-to (HMI sequence)

This is the sequence for using new HMI Framework. You can found full version in https://wiki.automotivelinux.org/windowmanager.

**Step 2:** requestSurface

**Step 3:** activateSurface & endDraw

**Step 4:** Event_TabShorcut

**Misc:** window resize
Step 1: Preparing

- First we should add two header files in the source.
  ```cpp
  #include <libwindowmanager.h>
  #include <libhomescreen.hpp>
  ```

- Define pointers to these two objects.
  ```cpp
  LibHomeScreen* hs;
  LibWindowManager* wm;
  ```

- Define application name using by **HS, WM**. Here we use “Navigation” as sample.
  ```cpp
  string app_name = string("Navigation");
  ```

- Get port and token from application arguments.
  ```cpp
  int port = strtol(argv[1], NULL, 10);
  string token = argv[2];
  ```

- Add libhomescreen and libwindowmanager librarys into CMakeLists.txt
  ```cpp
  libwindowmanager.so
  libhomescreen.so
  ```
Step2: requestSurface

- Initialize libwindowmanager object pointer.

```c
wm = new LibWindowManager();
if(init_wm(wm, &window) != 0) {
    fini_egl(&display);
    if (display.ivi_application)
        ivi_application_destroy(display.ivi_application);
    if (display.compositor)
        wl_compositor_destroy(display.compositor);
    wl_registry_destroy(display.registry);
    wl_display_flush(display.display);
    return -1;
}
```

- Init libwindowmanager

```c
if (wm->init(port, token.c_str()) != 0) {
    HMI_ERROR("simple-egl", "wm init failed.");
    return -1;
}
```

- requestSurface for simple-egl.

```c
json_object *obj = json_object_new_object();
json_object_object_add(obj, wm->kKeyDrawingName, json_object_new_string(app_name.c_str()));
g_id_ivities = wm->requestSurface(obj);
if (g_id_ivities < 0) {
    HMI_ERROR("simple-egl", "wm request surface failed ");
    return -1;
}
```
Step3: activateSurface & endDraw

We need request a surface id before using it in create_ivi_surface. And after that, we can display this surface by activateSurface().

eglSwapBuffers(window.display->egl_dpy, window.egl_surface);
json_object *obj = json_object_new_object();
json_object_object_add(obj, wm->kKeyDrawingName, json_object_new_string(app_name.c_str()));
json_object_object_add(obj, wm->kKeyDrawingArea, json_object_new_string("normal.full"));
wm->activateSurface(obj);

And call endDraw in Event_SyncDraw().

wm->set_event_handler(LibWindowmanager::Event_SyncDraw, [wm, window](json_object *object) {
    const char *label = json_object_get_string(
        json_object_object_get(object, wm->kKeyDrawingName));
    const char *area = json_object_get_string(
        json_object_object_get(object, wm->kKeyDrawingArea));

    json_object *obj = json_object_new_object();
    json_object_object_add(obj, wm->kKeyDrawingName, json_object_new_string(app_name.c_str()));

    wm->endDraw(obj);
});
Step 4: Event_TabShorcut

- Initialize libhomescreen object pointer.

```c
hs = new LibHomeScreen();
if(init_hs(hs)! = 0){
    fini_egl(display);
    if (display.ivi_application)
        ivi_application_destroy(display.ivi_application);
    if (display.compositor)
        wl_compositor_destroy(display.compositor);
    wl_registry_destroy(display.registry);
    wl_display_flush(display.display);
    return -1;
}
```

- Init libhomescreen and set event handler for Event_TapShortcut.

```c
int init_hs(LibHomeScreen* hs){
    if(hs->init(port, token)! = 0)
    {
        HMI_ERROR("simple-egl","homescreen init failed. ");
        return -1;
    }

    hs->set_event_handler(LibHomeScreen::Event_TapShortcut, []{json_object *object){
        const char *application_name = json_object_get_string(
            json_object_object_get(object, "application_name"));
        HMI_DEBUG("simple-egl","Event_TapShortcut application_name = %s ", application_name);
        if(strcmp(application_name, app_name.c_str()) == 0)
        {
            HMI_DEBUG("simple-egl","try to activesurface %s ", app_name.c_str());
            json_object *obj = json_object_new_object();
            json_object_object_add(obj, wm->kKeyDrawingName, json_object_new_string(app_name.c_str()));
            json_object_object_add(obj, wm->kKeyDrawingArea, json_object_new_string("normal.full"));
            wm->activateSurface(obj);
        }
    }});

    return 0;
}
```
After Step1~4, we can run this application and display normally.

But there is on more thing may be need to do.
In the original simple-egl, window is resized in the ivi_surface_listener callback. It can be done in the Event_SyncDraw, instead of ivi_surface_listener.

```c
wm->set_event_handler(LibWindowmanager::Event_SyncDraw, [wm, window](json_object *object) {
    const char *label = json_object_get_string(
        object, wm->kKeyDrawingName);
    const char *area = json_object_get_string(
        object, wm->kKeyDrawingArea);
    WM::egl_window_resize(window->native, 1080, 1488, 0, 0);
    window->geometry.width = 1080;
    window->geometry.height = 1488;
    HMI_DEBUG("simple-egl","Surface %s got syncDraw! Area: %s ", label, area);
    if ((wm->kStrLayoutNormal + "." + wm->kStrAreaFull) == std::string(area)) {
        wl_egl_window_resize(window->native, 1080, 1488, 0, 0);
        window->geometry.width = 1080;
        window->geometry.height = 1488;
    } else if ((wm->kStrLayoutSplit + "." + wm->kStrAreaMain) == std::string(area) ||
        (wm->kStrLayoutSplit + "." + wm->kStrAreaSub) == std::string(area)) {
        wl_egl_window_resize(window->native, 1080, 744, 0, 0);
        window->geometry.width = 1080;
        window->geometry.height = 744;
    }
    if (!window->fullscreen) {
        window->window_size = window->geometry;
        json_object *obj = json_object_new_object();
        json_object_object_add(obj, wm->kKeyDrawingName, json_object_new_string(app_name.c_str()));
        wm->endDraw(obj);
    }
});
```
Thank you very much!

Above all, we just finish the Wayland Client with new HMI Framework.

There is a full sample code in AGL gerrit.

https://gerrit.automotivelinux.org/gerrit/gitweb?p=src/libhomescreen.git;a=tree:f=sample/simple-egl;h=3b6a583636567cc5cbb41b46559b4e57ce00d7dd;hb=62e013c3bfa1ba66ceb459b5cc5e733335e8d6e7