EWK_Frame.h

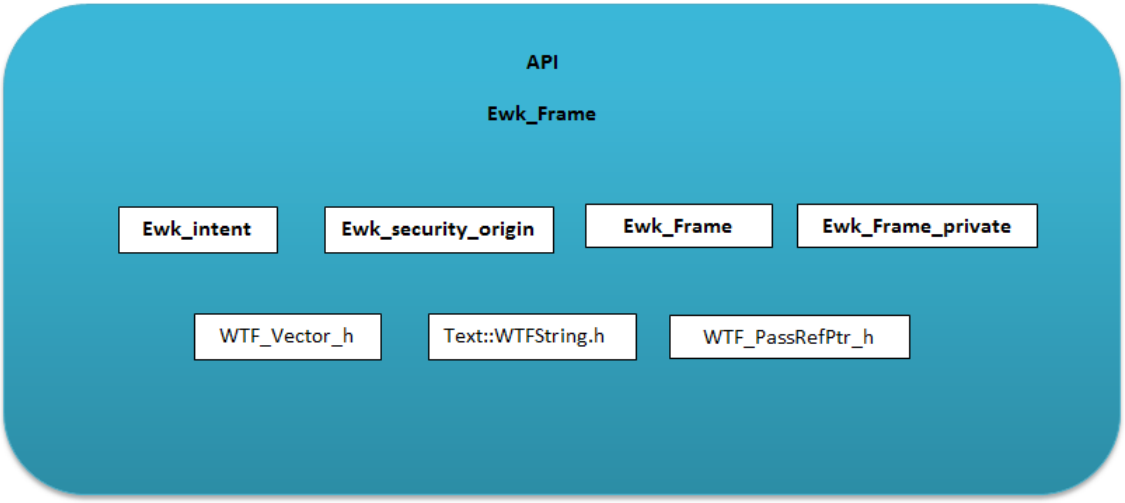
Introduction

When using an application such as a browser to visualize some web content, firstly an User Interface is needed in order to provide the functionalities and interactions the user may have with. In this way, such application displays a main frame which purspose is to contain different elements on the window and, of course, a second frame in which web content will be showed and the user can interact directly with web elements.

So, for this case, the purpose of Ewk_Frame.h and Ewk_frame_private.h libraries is to provide functions to work with when displaying html content in a browser is needed.

Dependencies

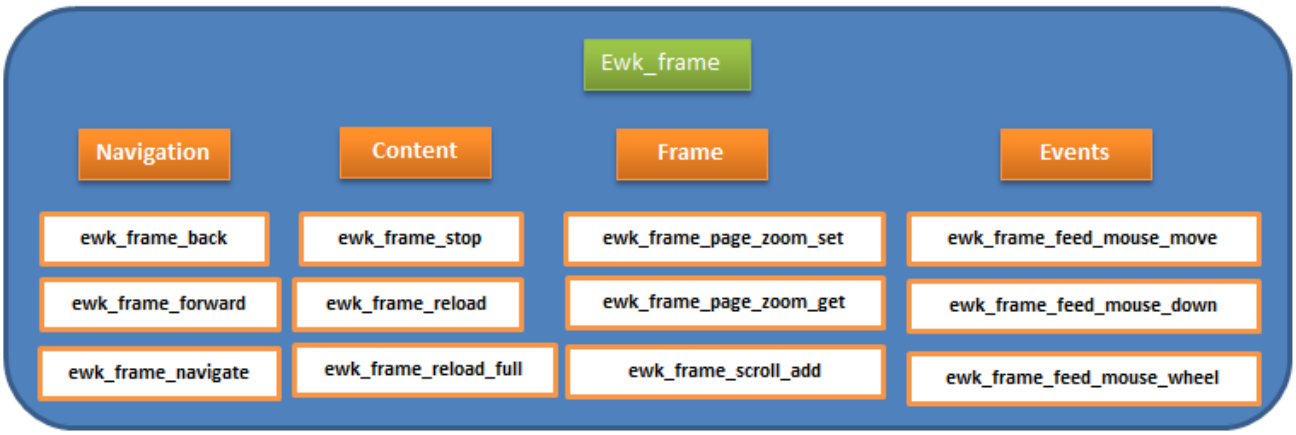
On the figure below, the libraries needed in Ewk_frame.h and Ewk_frame_private.h are shown to give an aidea about which files muts be included in thoses dependencies.



First, it is important to know that these modules have dependencies within ewk_view.h & ewk_view.cpp since the view is the extern part of the application wich will contain two frames (as it is handle in such an aplication for browser): the main frame including menu bars and the main html frame (the role of Ewk_frame).

Content

To start with the explanation about the use, first we have to know some of the functions can be used from these libraries in order to make an implementation of them. For this explanation, the most important functions will be handled to give an idea about this and which modules are needes as well.



How does it work?

Once we have considered this, it is possible to focus on how functions work. Each function of these libraries has one common argument: Evas_Object * o. This object represents such a frame where events, content, and information are displayed (according to the html frame for the browser).

Examples of definitions of functions:

EAPI Eina_Bool ewk_frame_stop(Evas_Object *o);
EAPI Eina_Bool ewk_frame_reload(Evas_Object *o);
EAPI Eina_Bool ewk_frame_reload_full(Evas_Object *o);
EAPI Eina_Bool ewk_frame_back(Evas_Object *o);
EAPI Eina_Bool ewk_frame_forward(Evas_Object *o);

It is important to mention that Ewk_view has two main frames: the one that is part of the application running on the OS and the second one that is for the Html content. In this case, by using functions belonging to the view library, it is easy to know how some functions are used, by just calling the function from the view and passing just one of their frames to perfom some of the functions commonly used for html content on the frame.

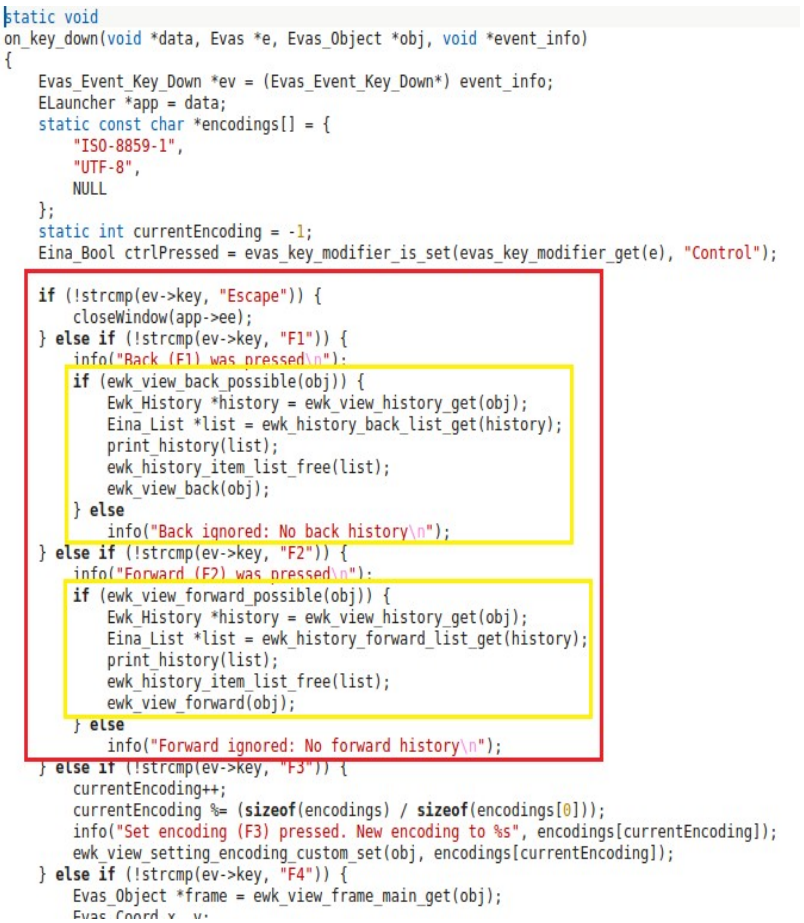
```
Eina_Bool ewk_view_stop(Evas_Object* ewkView)
{
    EWK_VIEW_SD_GET_OR_RETURN(ewkView, smartData, false);
    return ewk_frame_stop(smartData->main_frame);
}

Eina_Bool ewk_view_reload(Evas_Object* ewkView)
{
    EWK_VIEW_SD_GET_OR_RETURN(ewkView, smartData, false);
    return ewk_frame_reload(smartData->main_frame);
}

Eina_Bool ewk_view_reload_full(Evas_Object* ewkView)
{
    EWK_VIEW_SD_GET_OR_RETURN(ewkView, smartData, false);
    return ewk_frame_reload_full(smartData->main_frame);
}

Eina_Bool ewk_view_back(Evas_Object* ewkView)
{
    EWK_VIEW_SD_GET_OR_RETURN(ewkView, smartData, false);
    return ewk_frame_back(smartData->main_frame);
}
```

Figure 1.3 Some functions are shown. The argument is an Evas_Object called main_frame, used to display the content or perform different behaviors.



```
static void
on_key_down(void *data, Evas *e, Evas_Object *obj, void *event_info)
{
    Evas_Event_Key_Down *ev = (Evas_Event_Key_Down*) event_info;
    ELauncher *app = data;
    static const char *encodings[] = {
        "ISO-8859-1",
        "UTF-8",
        NULL
    };
    static int currentEncoding = -1;
    Eina_Bool ctrlPressed = evas_key_modifier_is_set(evas_key_modifier_get(e), "Control");

    if (!strcmp(ev->key, "Escape")) {
        closeWindow(app->ee);
    } else if (!strcmp(ev->key, "F1")) {
        info("Back (F1) was pressed\n");
        if (ewk_view_back_possible(obj)) {
            Ewk_History *history = ewk_view_history_get(obj);
            Eina_List *list = ewk_history_back_list_get(history);
            print_history(list);
            ewk_history_item_list_free(list);
            ewk_view_back(obj);
        } else
            info("Back ignored: No back history\n");
    } else if (!strcmp(ev->key, "F2")) {
        info("Forward (F2) was pressed\n");
        if (ewk_view_forward_possible(obj)) {
            Ewk_History *history = ewk_view_history_get(obj);
            Eina_List *list = ewk_history_forward_list_get(history);
            print_history(list);
            ewk_history_item_list_free(list);
            ewk_view_forward(obj);
        } else
            info("Forward ignored: No forward history\n");
    } else if (!strcmp(ev->key, "F3")) {
        currentEncoding++;
        currentEncoding %= (sizeof(encodings) / sizeof(encodings[0]));
        info("Set encoding (F3) pressed. New encoding to %s", encodings[currentEncoding]);
        ewk_view_setting_encoding_custom_set(obj, encodings[currentEncoding]);
    } else if (!strcmp(ev->key, "F4")) {
        Evas_Object *frame = ewk_view_frame_main_get(obj);
        Evas_Coord x, y;
```

Figure 1.4 Although these functions are implemented in the view, these make usage of frame's functions as shown on the Figure 1.3

The above image describes that according to the common functionalities to go backward or forward in the history depending on the visited web sites, we can go through this list by using the frame for the html and the functions such as back or forward.

```
} else if (!strcmp(ev->key, "F5")) {
    info("Reload (F5) was pressed, reloading.\n");
    ewk_view_reload(obj);
} else if (!strcmp(ev->key, "F6")) {
    info("Stop (F6) was pressed, stop loading.\n");
    ewk_view_stop(obj);
} else if (!strcmp(ev->key, "F12")) {
    Eina_Bool status = ewk_view_setting_spatial_navigation_get(obj);
    ewk_view_setting_spatial_navigation_set(obj, !status);
    info("Command::keyboard navigation toggle\n");
} else if ((!strcmp(ev->key, "minus") || !strcmp(ev->key, "KP_Subtract")) && ctrlPressed) {
    if (currentZoomLevel > MIN_ZOOM_LEVEL && zoom_level_set(obj, currentZoomLevel - 1))
        currentZoomLevel--;
    info("Zoom out (Ctrl + '-') was pressed, zoom level became %.2f\n", zoomLevels[currentZoomLevel] / 100.0);
} else if ((!strcmp(ev->key, "equal") || !strcmp(ev->key, "KP_Add")) && ctrlPressed) {
    if (currentZoomLevel < MAX_ZOOM_LEVEL && zoom_level_set(obj, currentZoomLevel + 1))
        currentZoomLevel++;
    info("Zoom in (Ctrl + '+') was pressed, zoom level became %.2f\n", zoomLevels[currentZoomLevel] / 100.0);
} else if (!strcmp(ev->key, "0") && ctrlPressed) {
    if (zoom_level_set(obj, DEFAULT_ZOOM_LEVEL))
        currentZoomLevel = DEFAULT_ZOOM_LEVEL;
    info("Zoom to default (Ctrl + '0') was pressed, zoom level became %.2f\n", zoomLevels[currentZoomLevel] / 100.0);
} else if (!strcmp(ev->key, "n") && ctrlPressed) {
    info("Create new window (Ctrl+n) was pressed.\n");
    browserCreate("http://www.google.com", app->userArgs);
} else if (!strcmp(ev->key, "g") && ctrlPressed ) {
    Evas_Coord x, y, w, h;
    Evas_Object *frame = ewk_view_frame_main_get(obj);
    float zoom = zoomLevels[currentZoomLevel] / 100.0;

    ewk_frame_visible_content_geometry_get(frame, EINA_FALSE, &x, &y, &w, &h);
```

Figure 1.5 Reload and stop. Functions commonly used in Html frames

After defining a URI, which wil be the resource searched by the browser to display its content, we can see that the same functions are involved inside the View as follows:

```
Eina_Bool ewk_view_uri_set(Evas_Object* ewkView, const char* uri)
{
    EWK_VIEW_SD_GET_OR_RETURN(ewkView, smartData, false);
    return ewk_frame_uri_set(smartData->main_frame, uri);
}

const char* ewk_view_uri_get(const Evas_Object* ewkView)
{
    EWK_VIEW_SD_GET_OR_RETURN(ewkView, smartData, 0);
    return ewk_frame_uri_get(smartData->main_frame);
}
```

Figure 1.6 Functions to set or get the resource on the browser.

And as it was defined above on the *browserCreate("http://www.google.com", app->userArgs)* function (Figure 1.5), this is the URI needed for the browser to start searching. An example of it can be like follows:

```
evas_object_event_callback_add(appBrowser->browser, EVAS_CALLBACK_DEL, on_browser_del, appBrowser);
evas_object_event_callback_add(appBrowser->browser, EVAS_CALLBACK_FOCUS_IN, on_focus_in, appBrowser);
evas_object_event_callback_add(appBrowser->browser, EVAS_CALLBACK_FOCUS_OUT, on_focus_out, appBrowser);
evas_object_event_callback_add(appBrowser->browser, EVAS_CALLBACK_KEY_DOWN, on_key_down, appBrowser);
evas_object_event_callback_add(appBrowser->browser, EVAS_CALLBACK_MOUSE_DOWN, on_mouse_down, appBrowser);

ewk_view_setting_enable_developer_extras_set(appBrowser->browser, EINA_TRUE);

appBrowser->url_bar = url_bar_add(appBrowser->browser, DEFAULT_WIDTH);

evas_object_move(appBrowser->browser, 0, URL_BAR_HEIGHT);
evas_object_resize(appBrowser->browser, userArgs->geometry.w, userArgs->geometry.h - URL_BAR_HEIGHT);

ewk_view_uri_set(appBrowser->browser, url);

evas_object_show(appBrowser->browser);
ecore_evas_show(appBrowser->ee);

evas_object_focus_set(appBrowser->browser, EINA_TRUE);
```

Figure 1. 6 Part when the URI is specified on the html frame of the browser.

More Information

Main Frame

Begining with the functions about the frame it is important to notice that all the functions are implemented on the WebCore:: which enables to respond to events (mouse, keyboard), elements when rendering the html page (loaders, displayers) and some other elements that allows us to get functionality.

There are some basic steps to consider using this library. Most of this, is based on some examples provided by the WebKit sourcecode and were analyzed enough to understand the sequence of the browser's steps.

So, firstly, we have to create a frame that will contain both menu bars with buttons for the interaction with the user and the interface and the html frame to display content. So, a URI can be specified to reach the resource or waits until the user enters the URI(on the URI bar). Once it happened, the frame's functions such as load_content must perform its behavior. Considering user's options to stop, reload or change uri, the external/internal frame must have communication (Ewk_view) holding this frames to show the changes. When the content is displayed, some events are implemented in order to response to some actions inside the internal frame to manipulate it.

By using an special type of data (depends on Evas library) is possible to store and retrieve information about the history or records from the visited uri's, (unless the **private navigation** is activated which does not allow to store information while browsing) and here is when appears those functions such as, backward, forward, navigation, etc.

There are other concepts called signals which are useful for transmission of notifications due to certain kind of behavior to let the user know about it through the main view, and all this signals may coccur, most of the times, when an error is produced when loading the page.