Plugin Structure

A chunkwm plugin consists of four essential parts. These are an init, deinit and main function, as well as a list of notifications that the plugin will subscribe to. In addition to the above, a plugin must provide both a name and a version.

The first step we have to do when creating a new plugin, is to include the necessary macros and definitions provided through the plugin api:

```
#include "../../api/plugin_api.h"
```

This file contains definitions for the init, deinit and main function as well as specifying the ABI version we are building against.

The init function gets called after a plugin has been loaded and the ABI version has been verified. The plugin is passed a pointer to a chunkwm function allowing it to broadcast custom notifications to other plugins.

The init function is defined through the PLUGIN_BOOL_FUNC macro and should return true if initialization succeeded, or false otherwise.

```
/*
 * NOTE(koekeishiya):
 * parameter: plugin_broadcast *Broadcast
 * return: bool -> true if startup succeeded
 */
PLUGIN_BOOL_FUNC(PluginInit)
{
    plugin_broadcast *ChunkWMBroadcastEvent = Broadcast;

    //
    // initialization code
    //

    return true;
}
```

The deinit function is called when a plugin is unloaded and is responsible for teardown and cleanup. This function is defined through the PLUGIN_VOID_FUNC macro.

```
PLUGIN_VOID_FUNC(PluginDeInit)
{
    //
    // deinitialization code
    //
}
```

The main function is invoked when the plugin is notified by chunkwm regarding system events or events broadcasted by other plugins. This function is defined through the PLUGIN_MAIN_FUNC macro. The return value is currently unused, but should return true for events that were handled properly, and false otherwise.

```
// contains definition of macos_application struct
#include "../../common/accessibility/application.h"

/*
 * NOTE(koekeishiya):
 * parameter: const char *Node
 * parameter: void *Data
 * return: bool
 * */
PLUGIN_MAIN_FUNC(PluginMain)
{
    if(strcmp(Node, "chunkwm_export_application_launched") == 0)
    {
        macos_application *Application = (macos_application *) Data;
        return true;
    }
    else if(strcmp(Node, "chunkwm_export_application_terminated") == 0)
    {
        macos_application *Application = (macos_application *) Data;
        return true;
    }

    return false;
}
```

After the above functions have been implemented, it is time to start constructing the plugin. The first thing we have to do is link our function pointers to the functions that we have implemented. This is done through the CHUNKWM_PLUGIN_VTABLE macro.

```
// NOTE(koekeishiya): Initialize plugin function pointers.
CHUNKWM_PLUGIN_VTABLE(PluginInit, PluginDeInit, PluginMain)
```

Secondly, we tell our plugin which events that we want to subscribe to using the CHUNKWM_PLUGIN_SUBSCRIBE macro. The array can remain empty if no events are necessary for the plugin to do whatever it was made for.

```
// NOTE(koekeishiya): Subscribe to ChunkWM events!
chunkwm_plugin_export Subscriptions[] =
{
    chunkwm_export_application_terminated,
    chunkwm_export_application_launched,
};
CHUNKWM_PLUGIN_SUBSCRIBE(Subscriptions)
```

Finally, we are ready to generate the plugin entry point used by chunkwm

```
// NOTE(koekeishiya): Generate plugin
static const char *PluginName = "template";
static const char *PluginVersion = "0.0.2";
CHUNKWM_PLUGIN(PluginName, PluginVersion);
```

Click here to view full the full plugin template

Notifications

Application

chunkwm_export_application_launched

Fired when an interactable application is launched
param: macos_application *

chunkwm_export_application_terminated

Fired when an interactable application is terminated
param: macos_application *

chunkwm_export_application_activated

Fired when an interactable application is activated
param: macos_application *

chunkwm_export_application_deactivated

Fired when an interactable application is deactivated
param: macos_application *

chunkwm_export_application_hidden

Fired when an interactable application is hidden
param: macos_application *

chunkwm_export_application_unhidden

Fired when an interactable application is unhidden
param: macos_application *

Window

chunkwm_export_window_created

Fired when a window is created
param: macos_window *

chunkwm_export_window_destroyed

Fired when a window is destroyed
param: macos_window *

chunkwm_export_window_focused

Fired when a window is given focus
param: macos_window *

chunkwm_export_window_moved

Fired when a window is moved
param: macos_window *

chunkwm_export_window_resized

Fired when a window is resized
param: macos_window *

chunkwm_export_window_minimized

Fired when a window is minimized
param: macos_window *

chunkwm_export_window_deminimized

Fired when a window is deminimized
param: macos_window *

Display

chunkwm_export_display_added

Fired when a new display is detected
param: CGDirectDisplayID

chunkwm_export_display_removed

Fired when a display is removed / disconnected
param: CGDirectDisplayID

chunkwm_export_display_moved

Fired when a display is moved (arrangement changes)
param: CGDirectDisplayID

chunkwm_export_display_resized

Fired when a display changes resolution
param: CGDirectDisplayID

chunkwm_export_display_changed

Fired when the display which holds key-input changes

Space

chunkwm_export_space_changed

Fired after a space transition has finished