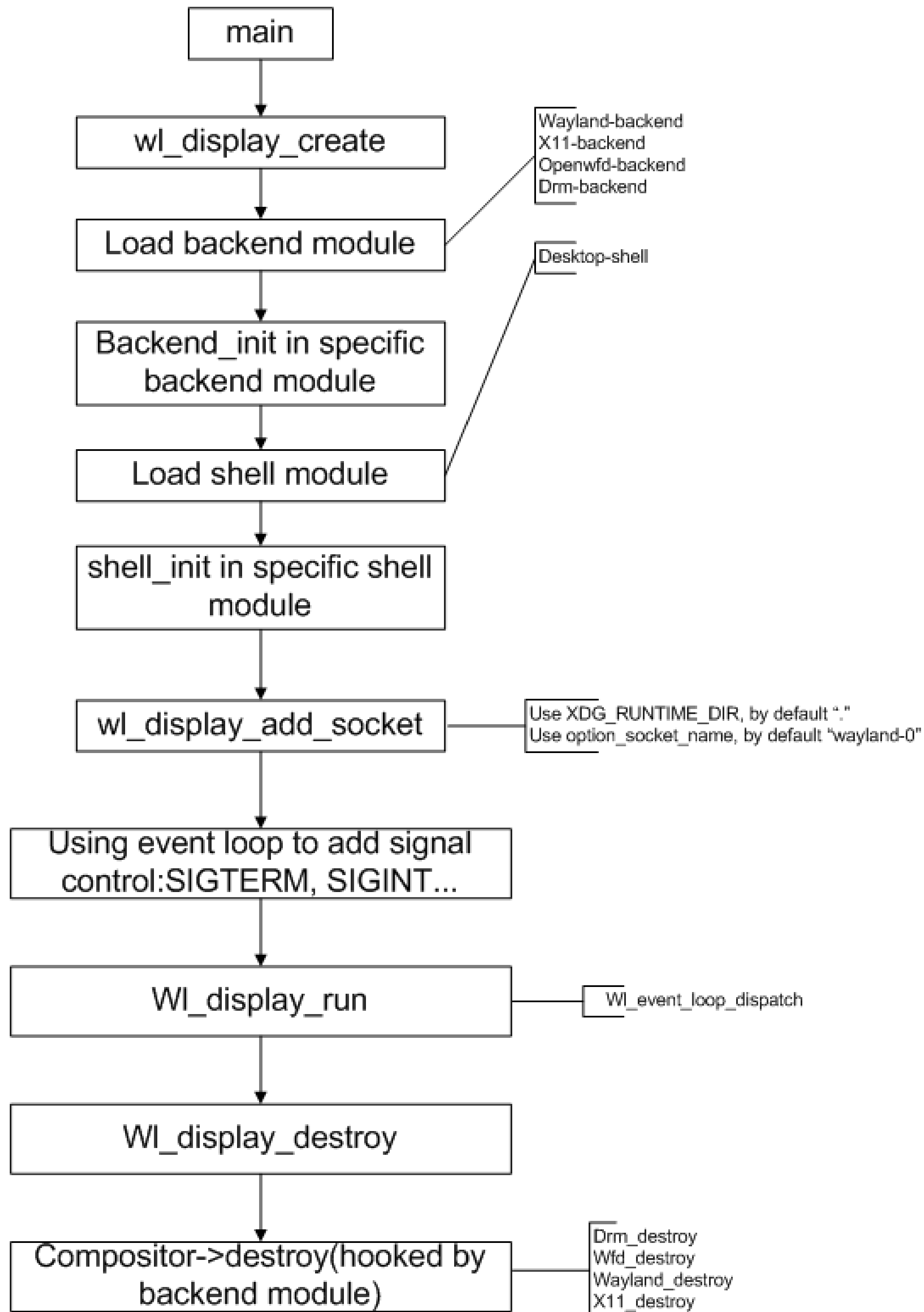
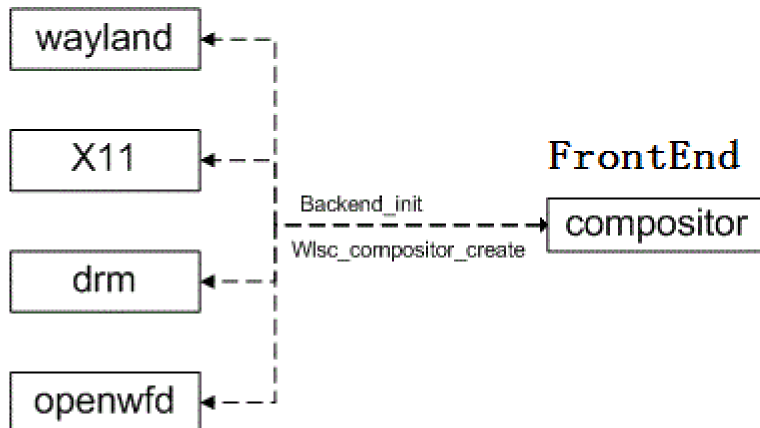
Wayland Compositor

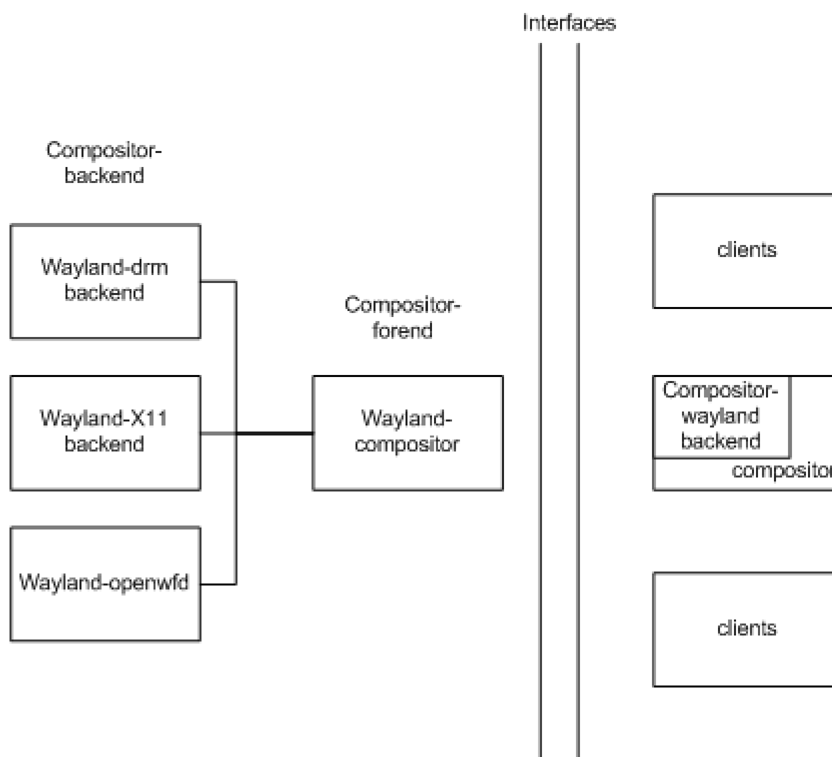
The compositor process, the sequences:



The bompositor backends and forend:



Compositor Wayland backend



backend compositor create:

1. Open device Or connect to the original display: for example /dev/dri/card0 for drm backend and connect to Xorg's display for X11 backend.
2. init egl and hook functions object
3. generate framebuffer object and bind them for drm & openwfd backend
4. call forend compositor init

5. create output
6. init output devices
7. add event source

Simple API introduction

wl_compositor_init in wayland/src/wayland-server.c:
Use the interface to initiate compositor.

```
/*Juan's notes: wl_compositor_interface is defined by xml file in wayland/protocol/wayland.xml, and
                interface is implemented in the compositor side, for example in wayland/compositor/compositor.c
                compositor_create_surface is the implementation for wl_compositor_create in the client side.*/
WL_EXPORT int
wl_compositor_init(struct wl_compositor *compositor,
                   const struct wl_compositor_interface *interface,
                   struct wl_display *display)
```

wl_shm_init in wayland/src/wayland-shm.c:
Initiate wl_shm and hook its call back functions. Except that callback functions in the compositor, it will use wl_shm_interface(shm_create_buffer in wayland/src/wayland-shm.c) to hook shm->obj.interface. [BR]]

```
wl_shm_init(struct wl_display *display,
            const struct wl_shm_callbacks *callbacks)
```

Usage example:
In wlsc_compositor_init, wayland-demos/compositor/compositor.c

```
        ec->wl_display = display;
        wl_compositor_init(&ec->compositor, &compositor_interface, display);
        ec->shm = wl_shm_init(display, &shm_callbacks);
```

Question: What does compositor's shm use for?

wl_display_add_object
in wayland/src/wayland-server.c, add one object to wl_display

```
WL_EXPORT void
```

```
wl_display_add_object(struct wl_display *display, struct wl_object
*object)
{
        object->id = display->id++;
        wl_hash_table_insert(display->objects, object->id, object);
}
```

wl_display_add_global

add one object to the global list saved in wl_display

```
WL_EXPORT int
wl_display_add_global(struct wl_display *display,
                      struct wl_object *object, wl_global_bind_func_t
func)
```

usage example screenshooter_create in wayland-demos/compositor/screenshooter.c:

```
struct screenshooter_interface screenshooter_implementation = {
        screenshooter_shoot
};

void
screenshooter_create(struct wlsc_compositor *ec)
{
        struct screenshooter *shooter;

        shooter = malloc(sizeof *shooter);
        if (shooter == NULL)
                return;
        /*notes by Juan: base is one wl_object, screenshooter_interface
is like the struct type.*/
        /*              and screenshooter_interface is defined in
wayland-demos/protocol/screenshooter.xml */
        shooter->base.interface = &screenshooter_interface;
        shooter->base.implementation =
                (void(**)(void)) &screenshooter_implementation;
        shooter->ec = ec;

        /*notes by Juan: add screenshooter object to wl_display*/
        wl_display_add_object(ec->wl_display, &shooter->base);
        wl_display_add_global(ec->wl_display, &shooter->base, NULL);
};
```

wl_display_get_event_loop
In wayland/src/wayland-server.c. Get the loop parameter in wl_display. This loop parameter is created on wl_display_create by wl_event_loop_create. It uses epoll technologies.

```
WL_EXPORT struct wl_event_loop *
wl_display_get_event_loop(struct wl_display *display)
{
        return display->loop;
}
```

wl_event_loop_create
In wayland/src/event-loop.c. Create event loop by "epoll_create1" for its epoll_fd and init its checklist. *What is checklist used for?*

```
WL_EXPORT struct wl_event_loop *
wl_event_loop_create()
```

wl_display_create
In wayland/src/wayland-server.c. The compositor will use this to create its wl_display. And the client will use wl_display_connect to connect to this display.
Malloc the storage for display; create its eventloop structure; create its objects hash table; init its frame/global/socket/client list;init the display object's interface(wl_display_interface) and its implementation:display_interface.[[BR]]

```
WL_EXPORT struct wl_display *
wl_display_create(void)
```

wl_event_loop_add_timer/fd/idle/signal
timer is one kind of event source. In its structure, it has base, fd and func. fd is created by timerfd_create.
In wayland-compositer(wayland-demos/compositor/compositor.c), it has idle_source and timer_source, all of them are timer event sources, and they hood different funcs: idle_handler and repaint.
fd: for example, drm_source and udev_drm_source are fd event sources. Another specific example, in evdev_input_add_devices->evdev_input_device_create, it will add the devnode fd as one fd event source to the event loop; and the hook function is evdev_input_device_data.
signal: the signals like "SIGTERM/SIGINT". Hook the specific functions to that signals.
idle: I did not find out the specific usage examples here.

```
WL_EXPORT struct wl_event_source *
wl_event_loop_add_timer(struct wl_event_loop *loop,
                        wl_event_loop_timer_func_t func,
                        void *data)

WL_EXPORT struct wl_event_source *
wl_event_loop_add_fd(struct wl_event_loop *loop,
                     int fd, uint32_t mask,
                     wl_event_loop_fd_func_t func,
                     void *data)
WL_EXPORT struct wl_event_source *
wl_event_loop_add_signal(struct wl_event_loop *loop,
                        int signal_number,
                        wl_event_loop_signal_func_t func,
                        void *data)

WL_EXPORT struct wl_event_source *
wl_event_loop_add_idle(struct wl_event_loop *loop,
                       wl_event_loop_idle_func_t func,
                       void *data)
```

Interfaces

Looks like the interfaces is used for client call server's functions.

wl_display_interface

defined by wayland/protocol/wayland.xml. Implemented in wayland/src/wayland-server.c.

wl_display is the core global object. It is used for wayland protocol features.

```
struct wl_display_interface display_interface = {
        display_bind, //client bind to one global object in display by
id
        display_sync, //like event tag, post one sync event?
        display_frame //request notification when the next frame is
displayed
};
```

wl_compositor_interface

defined by wayland/protocol/wayland.xml. Implemented in wayland-demos/compositor/compositor.c

The compositor object is a golbal one. The compositor is in charge of

combining the contents of multiple surfaces into one displayable output. It is the compositor's responsibility to implement it.
usage example: in wayland-demos/clients/simple-client.c,it uses wl_compositor_create_surface to create surfaces.

```
const static struct wl_compositor_interface compositor_interface = {
        compositor_create_surface,
};
```

wl_shm_interface
Shared memory support. Implemented in wayland/src/wayland-shm.c
For example, in wayland-demos/clients/window.c, the function display_create_shm_surface uses wl_shm_create_buffer to create the shm buffer.

```
const static struct wl_shm_interface shm_interface = {
        shm_create_buffer
};
```

wl_buffer_interface
A wl_buffer is a pixel buffer. Created using the drm, shm or similar objects. It has a size, visual and contents, but not a location on the screen.
Implemented in wayland/src/wayland-shm.c
For example, in wayland-demos/clients/window.c, the function display_surface_damage uses wl_buffer_damage to redraw that buffer.

```
const static struct wl_buffer_interface shm_buffer_interface = {
        shm_buffer_damage,
        shm_buffer_destroy
};
```

wl_shell_interface
In current demos, it looks like only meego-tablet-shell use this interface
wl_shell_interface is implemented in wayland-demos/compositor/shell.c

```
const static struct wl_shell_interface shell_interface = {
        shell_move,
        shell_resize,
        shell_create_drag,
        shell_create_selection,
        shell_set_toplevel,
        shell_set_transient,
        shell_set_fullscreen
```

```
};
```

wl_grab_interface is implemented in wayland-demos/compositor/shell.c

```
static const struct wl_grab_interface move_grab_interface = {
        move_grab_motion,
        move_grab_button,
        move_grab_end
};

static const struct wl_grab_interface drag_grab_interface = {
        drag_grab_motion,
        drag_grab_button,
        drag_grab_end
};

static const struct wl_drag_interface drag_interface = {
        drag_offer,
        drag_activate,
        drag_destroy,
};
```

wl_drag_offer_interface is implemented in
wayland-demos/compositor/shell.c

```
static const struct wl_drag_offer_interface drag_offer_interface = {
        drag_offer_accept,
        drag_offer_receive,
        drag_offer_reject
};
```

wl_selection_offer_interface is implemented in
wayland-demos/compositor/shell.c

```
static const struct wl_selection_offer_interface
selection_offer_interface = {
        selection_offer_receive
};
```

wl_selection_interface is implemented in
wayland-demos/compositor/shell.c

```
static const struct wl_selection_interface selection_interface = {
```

```
        selection_offer,
        selection_activate,
        selection_destroy
};
```