



Hard keys

Contents

2	Definition	2
3	Out of scope	2
4	Types of usage & example use-cases	3
5	System keys	3
6	example use-cases	3
7	Role specific keys	3
8	Example use-case	3
9	Application keys	4
10	Example use-case	4
11	Non-functional requirements	4
12	Design	4
13	Compositor Input sources	5
14	Compositor Input processing	5
15	System keys	6
16	Application keys	6
17	Role keys	6
18	Key handling recommendations	6

Definition

Hardkeys: Also known as Hardware keys; Any controls in the car system connected to the Head unit. Examples of hardkeys are volume controls, fixed function buttons (back, forward, home screen, play pause), rotary controls etc.

Traditionally hardkeys referred only to physical controls (e.g. switches, rotary dials etc), hence hardware keys. But current systems are far more flexible and due to that this design can also refer to software buttons rendered by the system UI outside of the applications control or touch sensitive areas with software controlled functionality.

As a simple guideline to determine if a control falls under this design concept is to sample ask the question: “Could this functionality have provided by a physical key/knob”. If the answer is yes, it fits into this design otherwise it doesn’t.

Out of scope

For the current design the following aspect our defined as out of scope. They are either addressed belong to other designs or could be addressed in future iterations.

- Haptic feedback
- Application controlled buttons (e.g. application configurable icons)
- short vs. long key presses handling (part of the UI framework/toolkit)
- Display systems other than wayland
- Any requirements about the type of controls that need to be available.
- Implementation design; This document explains the high-level concepts not the implementation

Types of usage & example use-cases

We recognize three different types of controls and the effect that they have on the system:

- System controls: Effect the system as a whole (volume controls, home key).
- Role controls: Effect the current system context (pause/play)
- Application controls: Effect the current foreground application

The following sections provide more detail for these various types of controls.

System keys

Buttons handled by “the system”. Regardless of application state. Example volume controls (always adjust the system volume), home screen buttons (always navigate back to the home screen), application shortcut buttons e.g. navigation (always open the specific application).

example use-cases

- Bob is using a webbrowser, presses the “home button”. The system goes back to the homescreen.
- Alice presses the “navigation”button. The system opens the navigation application

Role specific keys

Buttons that should be handled by an agent or application fulfilling a specific role. An example here are play/pause buttons which should get handled by the program currently playing audio (e.g. internet radio application, local media player etc).

Example use-case

- Simon starts playing music in the spotify application, switches to the web-browser application while the music streams keeps playing in the background. Simon doesn’t like the current song and presses the “next song”

70 button, the spotify agent running in the background switches to the next
71 song.

72 Application keys

73 Buttons that should always be handled by the currently active application. For
74 example up/down/left/right select buttons.

75 Example use-case

- 76 • Harry is browsing through a list of potential destinations in the navigation
77 application. He turns the rotary dial on the center console, the focus moves
78 to the next potential destination on the list.
- 79 • Barry looks up a new radio station in the radio application. After listening
80 a while he decides he likes the current station. Barry hold the “favourites”
81 button for a while (long press), the radio application bookmarks the cur-
82 rent station.

83 Non-functional requirements

84 Apart from the use-cases mentioned above, there are several requirements on
85 the design that don't directly impact the functionality but are important from
86 e.g. a security point of view.

- 87 • Applications should not be able to eavesdrop on the input sent to other
88 processes
- 89 • Applications should not be able to inject inputs into other processes ([Syn-
90 thesized input](#)¹)

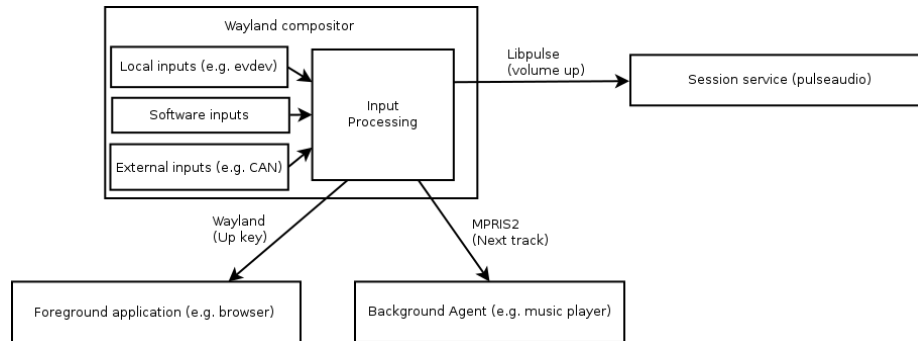
91 A more complete overview of issues surrounding input security (integrity, confi-
92 dentiality) be found on the [Compositor security](#)² page.

93 Design

94 On wayland systems the design of inputs is relatively straightforward, the com-
95 positor is responsible for gathering all the inputs from the various sources and
96 chooses how to handle them. The diagram below has a high-level overview of
97 some example input sources and examples of the various types of controls.

¹https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/compositor_security/#Synthesize_input

²https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/compositor_security/



Thanks to the Wayland input flow only having two actors (the compositor and the receiver) (as opposed to that of X11) that there is no way for applications to either spy on the inputs of other applications or to inject inputs into other applications.

Compositor Input sources

Various example inputs are shown in the diagram (though others could be defined as well):

- Local inputs: evdev is the kernel input subsystem, directly attached controls will emit key events using that subsystem (e.g. i2c attached buttons)
- External inputs: Any input sources that aren't directly attached, e.g. inputs via the CAN network or even an IP network
- Software inputs: Software defined input sources, e.g. onscreen buttons drawn by the compositor or pre-defined touchscreen areas

Note that the exact implementation of gathering input is left up to the implementer. E.g. for CAN inputs it's not recommended for the compositor to have direct access to the CAN bus, however that design is part the implementation of the generic CAN handling.

Each of these events input will feed into the input processing internally into the compositor.

Compositor Input processing

All input events are gathered into one consistent way of input processing in the compositor. From both the compositor internals and the applications/agents point of view the exact technology to gather the inputs is not relevant (Note that the device could be, e.g. steering wheel controls vs center console controls).

The task of the input processing into the compositor is to determine where the key event should be delivered and via which method. Following the classification outlined earlier.

126 **System keys**

127 Keys meant for the system are processed directly by the compositor. Resulting
128 in the compositor taking an action either purely by itself (e.g. switching to an
129 overview screen) or by calling methods on external services. In the example
130 given the diagram, the compositor processes the “volume up”key and as a result
131 of that uses the libpulse API to ask pulseaudio to increase the system volume.

132 **Application keys**

133 Keys meant for the current foreground application are simply forward to the
134 application using the wayland protocol. All further handling is up to the ap-
135 plications and its toolkit, this includes but is not limited to the recognition of
136 short-press vs. long-press, application specific interpretation of keys (e.g. open-
137 ing the bookmarks if the favourites key is pressed) etc.

138 Note that the current foreground application might well be the compositor itself.
139 For example if the compositor is responsible of rendering the status bar, it could
140 be possible to use key navigation to navigate the statusbar.

141 **Role keys**

142 Keys for a specific role are forwarded to the application or agent current fulfilling
143 that role. It is expected that each role implements a role-specific D-Bus
144 interface either for direct key handling or commands.

145 For example for music players, assuming the MPRIS2 is mandated for media
146 agents (including Radio), the compositor would determine which agent or appli-
147 cation currently has the Media role and call the MPRIS2 method corresponding
148 to the key on it (e.g. Next).

149 The reason for requiring role-specific D-Bus interfaces rather than simply for-
150 warding keys via e.g. the wayland protocol is that an agent doesn’t have to
151 connect to the display server, only to the D-Bus session bus. It also means there
152 is a clean separation between the actual key handling and the action taken. For
153 example an OEM may define a short press on a “forward”key meaning a seek,
154 while a long-press means “next track”, in this design that can purely be policy
155 in the compositor.

156 **Key handling recommendations**

157 Even though it is out of scope for this design specifically. Some general recom-
158 mendations about key handling:

- 159 • Keys should be specific to one category even when used in long vs. short
160 press or in combinations. As undoing key events is impossible.

- 161 • Any combination keys or keys with long-press abilities should only be han-
162 dled on key release (key down ignored), for the same reasoning. Cancelling
163 an in-progress action is either confusing to the user or not possible.