On-screen keyboard

Contents

Terminology and concepts **2**
Text input 2
Input method 3
Virtual keyboard 5

Use cases **6**

Requirements **6**

Obsolete Approach **6**
Using a New Widget 6

Implementations in Other Systems **7**

New Approach **7**

Evaluation Report **8**
weston-keyboard 8
Maliit Keyboard 2 8
Squeekboard 9
Implementation status 9

Recommendations **9**

Risks **10**

References **10**

Apertis can be used with a touchscreen only, in this case the user will need an on-screen keyboard to be able to enter information like passwords, URLs, messages.

This document outlines the current state of the Wayland protocols dealing with input methods, their implementation status as well as a possible approach for integrating this support into Apertis.

Terminology and concepts

In Wayland, multiple protocols are involved to allow users to enter text.

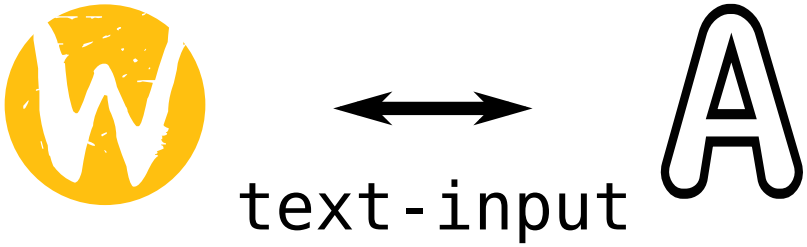
Text input

The **text-input** protocol allows compositors to send text to applications in a way which supports various input methods other than direct physical keyboard input. Examples of this include complex text composition methods such as

CJK[1] alphabets in which each character is typically composed from multiple keypresses, or state-aware input methods such as on-screen virtual keyboards which may offer text suggestions, correction, autocompletion, emoji, and other complex input types which are not supported by the traditional keyboard input mechanism.

A text input object is used to manage state of what are typically text entry fields in the application. Client applications send enable/disable events to the compositor following text input focus changes (this is typically done by the GUI framework in use), and the compositor can then decide when and where to display the on-screen keyboard.

Apart from enable/disable events, a number of state requests may also be sent by the client, allowing the compositor to keep track of the state of the input field. For example, *set_content_type* can be used by the client to specify what kind of text is expected, while *set_cursor_rectangle* can be used to specify an area around the cursor and thus allow the compositor to put a window with word suggestions near the cursor, without obstructing the text being input.



This protocol is currently on version v3[2] upstream, with v4[3] being discussed.

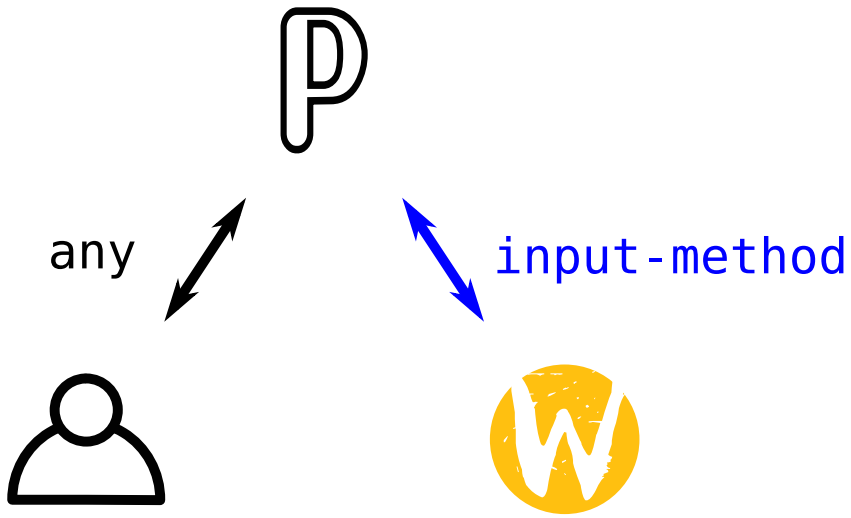
Input method

The **input-method** protocol allows the compositor to delegate work to let user input text to some other program.

[1] https://en.wikipedia.org/wiki/CJK_characters

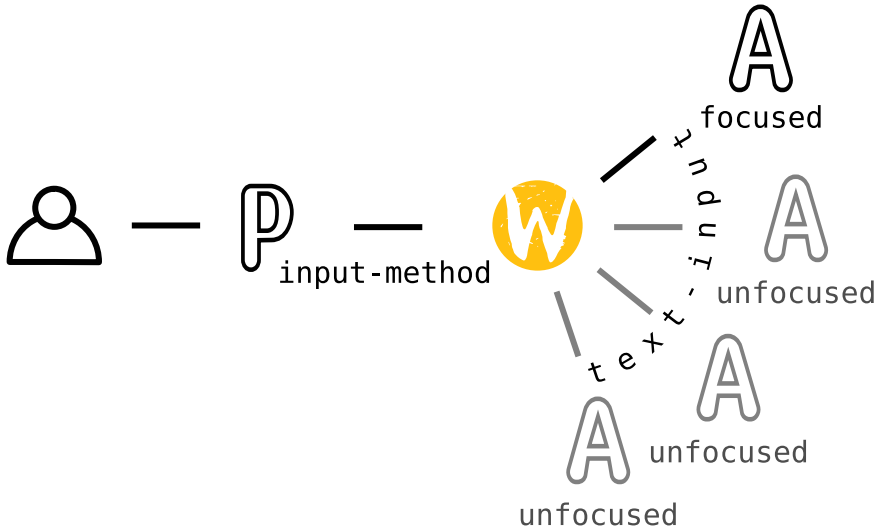
[2] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/master/unstable/text-input/text-input-unstable-v3.xml

[3] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/73#note_850436



This protocol is very similar to *text-input*, because it lets *a program* (e.g. an on-screen keyboard application) to send text *to the compositor*, and allows *the compositor* to tell *this program* what kind of text is needed.

The program will then communicate to the user (e.g. through interaction with the on-screen keyboard) and give the text to the compositor. Once received, the compositor will typically send the text onward to the currently focused application using the *text-input* protocol, creating a chain: special program → compositor → focused application.



Additionally, because there is typically only one application using this protocol, it can do things which would not work with multiple applications. One of them is grabbing the keyboard, by allowing the input method to receive all hardware keyboard input (*exclusive grab*). This allows the input method to preprocess the input before forwarding it, which is common to CJK language users, for example by allowing the input method to send the text “你好”when “nihao”is typed.

The latest protocol supported upstream is on version v1[4], with version v2[5] available and v3[6] under development.

Virtual keyboard

The **virtual-keyboard** protocol is designed for programs which want to tell the compositor to issue “fake”keyboard events, as if they came from a physical keyboard.

This should allow inputting text in legacy applications which don’t support the *text-input* protocol or triggering actions which would normally need a keyboard, and is done by emulating key presses.

Important to note that if the compositor enables a keyboard to perform arbitrary actions, it should prevent untrusted clients from using this interface.

[4] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/master/unstable/input-method/input-method-unstable-v1.xml

[5] https://github.com/swaywm/wlroots/blob/master/protocol/input-method-unstable-v2.xml

[6] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/112

This protocol is not yet available upstream, with a proposal[7] adding v1 support currently under discussion.

Use cases

- A user wants to enter a text without a physical keyboard (i.e. using an on-screen keyboard)
- A user wants to be able to enter text in a number of languages and writing systems[8] (e.g. English/Latin, CJK)
- A user wants to be able to make use of text input features such as correction and completion suggestions
- A user wants to be able to select and input emoji characters

Requirements

The chosen on-screen keyboard implementation must:

- allow to configure the keyboard layout
- be automatically enabled when user selects a text input field and allow users to show it manually (for legacy applications - see below)
- not require any changes to the applications themselves

Obsolete Approach

In previous versions of Apertis, a custom widget in the client application[9] was used for spell checking. This widget was built to exclusively target the legacy Mildenhall platform, and thus it brings several problems:

- It is exclusively tied to Mildenhall applications; no other GUI frameworks are supported.
- Each app has their own instance of the widget, thus the application side is also responsible for tasks such as positioning the keyboard on the screen, while not actually knowing the full screen layout as a compositor does.

Attempting to migrate it away from being dependent on Mildenhall would essentially amount to a full rewrite, resulting in little advantage versus an alternative solution.

Using a New Widget

Even if a new in-application widget were created, both of these points would still apply:

[7] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/11
[8] https://en.wikipedia.org/wiki/Writing_system
[9] https://gitlab.apertis.org/pkg/mildenhall/-/blob/apertis/v2021/widgets/mildenhall-speller/mildenhall_speller.c

- There would need to be a widget for every graphical application framework, and every application using the framework would need to explicitly include the widget.
- The problems and inefficiency with having the application position the keyboard on the screen without full knowledge of the entire screen persist outside of Mildenhall, as it is purely an architectural problem.

Implementations in Other Systems

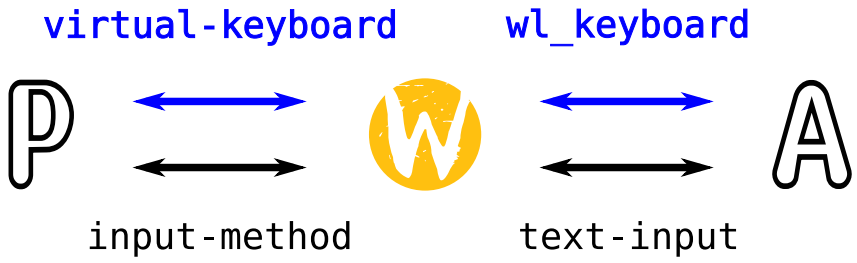
Qt uses a framework-specific, client-side plugin, qtvirtualkeyboard[10]. Thus, like the obsolete Mildenhall speller widget mentioned previously, qtvirtualkeyboard is exclusively tied to Qt applications. In addition, code reuse is not possible, as it is under the GPLv3 license.

Samsung's Tizen has a custom IME framework[11] built on top of HTML5 and JavaScript, as with the rest of the OS's applications. The only native code used is essentially for communication between the IME and the client application, resulting in little use for building off of Tizen's approach.

LG webOS OSE uses custom plugins[12] on top of the Maliit IME framework[13]. Although Maliit itself is LGPL-2.1, the reference keyboard implementation is under LGPL-3.

New Approach

A fully-fledged input method program will be a Wayland client using the *input-method* protocol for submitting text, but also supporting *virtual-keyboard* for submitting actions, and as a fallback for legacy applications.



A compositor would ferry text around between the input method program and whichever application is focused. It would also carry synthetic keyboard events

[10] https://doc.qt.io/qt-5/qtvirtualkeyboard-deployment-guide.html
[11] https://github.com/Samsung/tizen-docs/blob/master/docs/application/web/guides/text-input/input-method.md
[12] https://github.com/webosose/ime-manager
[13] https://maliit.github.io/

from the input method program to the focused application.

An application consuming text would support *text-input*, generally through a GUI framework like GTK or Qt, and it would send enable and disable events whenever a text input field comes into focus or becomes unfocused.

Legacy applications won't send enable and disable events, even when a text field is focused and the user is ready to type. When that happens, the compositor and the input method won't realize when to display the on-screen keyboard or when text should be submitted. Because of that, it's best to always make sure the user can bring up the on-screen keyboard to be able to input text, which would then be delivered as keyboard events (which are always supported by applications) via the *virtual-keyboard* protocol.

Currently the majority of the on-screen keyboard applications was developed for the `X11` display server. For `Wayland` only a few are available:

- weston-keyboard
- Maliit Keyboard 2[14]
- Squeekboard[15]

Evaluation Report

weston-keyboard

Simple implementation of an on-screen keyboard. The application only supports roman, numeric and arabic keyboards, which are hardcoded, and it is built on top of outdated versions of the *text-input* and *input-method* protocols. (This can be improved, however.)

- License: X11, MIT and CC-BY-SA
- Languages: C

Maliit Keyboard 2

Maliit Keyboard 2 is an evolution of the Ubuntu Keyboard[16] plugin for Maliit, which can be run standalone and supports many different languages and emoji.

- License: LGPL-3, BSD and CC-BY (The license of the combined work is LGPL-3.0-only)
- Languages: QML, C++

[14] https://github.com/maliit/keyboard
[15] https://source.puri.sm/Librem5/squeekboard
[16] https://launchpad.net/ubuntu-keyboard

Squeekboard

Squeekboard has been developed to be the on-screen keyboard of Librem 5 phone OS, using Phoc[17] compositor which is based on wlroots[18].

- License: GPL-3
- Languages: Rust, C

Implementation status

The following table lists GUI frameworks, clients and compositors and their corresponding implementation status:

	text-input	input-method	virtual-keyboard
GTK	v3	-	-
Qt	v2	-	-
IBus	-	v1	-
Chromium	v1	-	-
WebKitGTK	v3	-	-
Weston	v1	v1	-
weston-keyboard	v1	v1	-
Squeekboard	v3	v2	v1
Maliit Keyboard	-	v1	-

Recommendations

Given the only (currently) GPL-3 free (matching Apertis licensing expectations[19]) on-screen keyboard implementation is a simple/demo version (*weston-keyboard*), Apertis may either opt to improve it, use one of the other existing implementations (as a GPL-3 exception) or implement a new one from scratch.

The recommended approach is to patch Weston to support the latest protocols versions and ship *weston-keyboard* as the reference on-screen keyboard implementation. A merge request[20] exists to integrate *text-input* v3, *input-method* v2 and *virtual-keyboard* v1 to Weston (including *weston-keyboard*) and could be used as a starting point.

As needed, *weston-keyboard* could potentially be forked as a separate project from Weston to allow using a more modern GUI toolkit for its implementation.

Optionally, other changes could be made to *weston-keyboard* to improve or implement new features such as supporting more languages or adding emoji support.

[17] https://gitlab.gnome.org/World/Phosh/phoc
[18] https://github.com/swaywm/wlroots
[19] https://www.apertis.org/policies/license-expectations/
[20] https://gitlab.freedesktop.org/wayland/weston/-/merge_requests/150

This should allow existing applications to interact with the on-screen keyboard without modifications, even for legacy applications not supporting the *text-input* protocol.

Risks

The different Wayland protocols involved in an on-screen keyboard are currently under development and subject to change, see Stalled Upstream Protocol Work[21].

References

This documentation and some of the illustrations are based on or come from:

- Wayland and input methods[22] blog post
- It's not about keyboards[23] blog post
- Input Method Hub[24] Wayland issue

[21] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/issues/39#stalled-upstream-protocol-work
[22] https://dcz_self.gitlab.io/posts/input_method/
[23] https://dcz_self.gitlab.io/posts/not_keyboard/
[24] https://gitlab.freedesktop.org/wayland/wayland-protocols/-/issues/39