



On-screen keyboard

1	Contents	
2	Terminology and concepts	2
3	Text input	2
4	Input method	3
5	Virtual keyboard	5
6	Use cases	6
7	Requirements	6
8	Obsolete Approach	6
9	Using a New Widget	6
10	Implementations in Other Systems	7
11	New Approach	7
12	Evaluation Report	8
13	weston-keyboard	8
14	Maliit Keyboard 2	8
15	Squeekboard	9
16	Implementation status	9
17	Recommendations	9
18	Risks	10
19	References	10
20	Apertis can be used with a touchscreen only, in this case the user will need	
21	an on-screen keyboard to be able to enter information like passwords, URLs,	
22	messages.	
23	This document outlines the current state of the Wayland protocols dealing with	
24	input methods, their implementation status as well as a possible approach for	
25	integrating this support into Apertis.	

26 Terminology and concepts

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

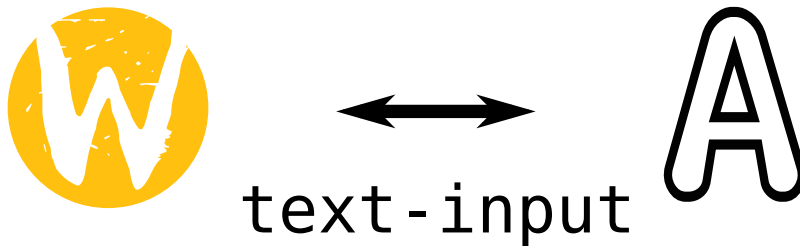
28 Text input

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

32 CJK¹ alphabets in which each character is typically composed from multiple
33 keypresses, or state-aware input methods such as on-screen virtual keyboards
34 which may offer text suggestions, correction, autocompletion, emoji, and other
35 complex input types which are not supported by the traditional keyboard input
36 mechanism.

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

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



48

49 This protocol is currently on version v3² upstream, with v4³ being discussed.

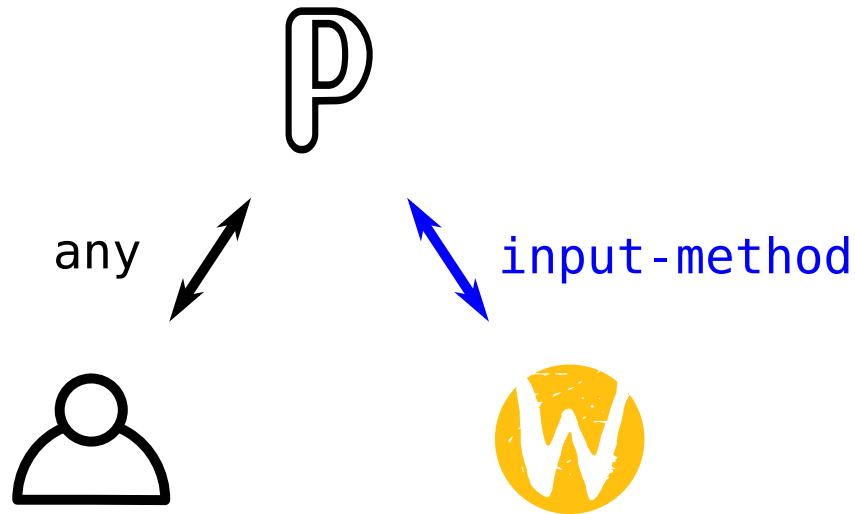
50 Input method

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

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

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

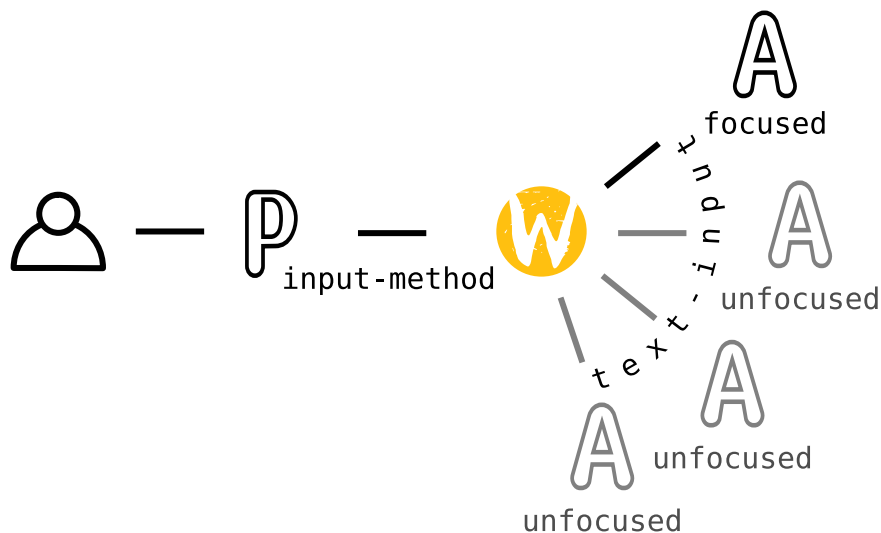
³https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/73#note_850436



53

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

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



62

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

70 The latest protocol supported upstream is on version [v1](https://gitlab.freedesktop.org/wayland/wayland-protocols/-/blob/master/unstable/input-method/input-method-unstable-v1.xml)⁴, with version [v2](https://github.com/swaywm/wlroots/blob/master/protocol/input-method-unstable-v2.xml)⁵
 71 available and [v3](https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/112)⁶ under development.

72 Virtual keyboard

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

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

79 Important to note that if the compositor enables a keyboard to perform arbitrary
 80 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>

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

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

83 Use cases

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

91 Requirements

92 The chosen on-screen keyboard implementation must:

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

97 Obsolete Approach

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

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

106 Attempting to migrate it away from being dependent on Mildenhall would essen-
107 tially amount to a full rewrite, resulting in little advantage versus an alternative
108 solution.

109 Using a New Widget

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

⁷https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/11

⁸https://en.wikipedia.org/wiki/Writing_system

⁹https://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

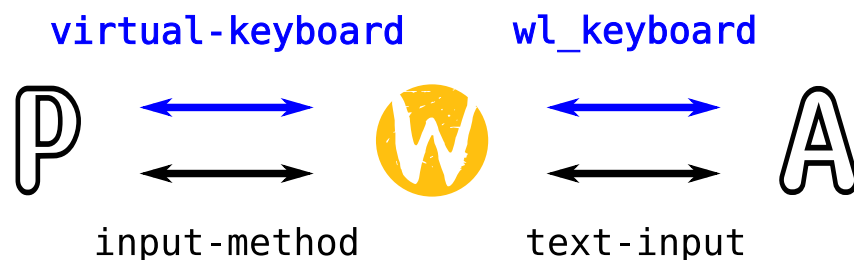
Qt uses a framework-specific, client-side plugin, `qtvirtualkeyboard`¹⁰. Thus, like the obsolete Mildenhall speller widget mentioned previously, `qtvirtualkeyboard` is exclusively tied to Qt applications. In addition, code reuse is not possible, as it is under the GPLv3 license.

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.



134

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/>

137 from the input method program to the focused application.

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

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

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

- 150 • weston-keyboard
- 151 • [Maliit Keyboard 2](#)¹⁴
- 152 • [Squeekboard](#)¹⁵

153 Evaluation Report

154 weston-keyboard

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

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

161 Maliit Keyboard 2

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

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

¹⁴<https://github.com/maliit/keyboard>

¹⁵<https://source.puri.sm/Librem5/squeekboard>

¹⁶<https://launchpad.net/ubuntu-keyboard>

167 Squeekboard

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

- 170 • License: GPL-3
- 171 • Languages: Rust, C

172 Implementation status

173 The following table lists GUI frameworks, clients and compositors and their
174 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	-

175 Recommendations

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

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

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

187 Optionally, other changes could be made to *weston-keyboard* to improve or imple-
188 ment 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://apertis-website-0b3586.pages.apertis.org/policies/license-expectations/>

²⁰https://gitlab.freedesktop.org/wayland/weston/-/merge_requests/150

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

192 Risks

193 The different Wayland protocols involved in an on-screen keyboard are cur-
194 rently under development and subject to change, see [Stalled Upstream Protocol](#)
195 [Work](#)²¹.

196 References

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

- 198 • [Wayland and input methods](#)²² blog post
- 199 • [It's not about keyboards](#)²³ blog post
- 200 • [Input Method Hub](#)²⁴ Wayland issue

²¹<https://gitlab.freedesktop.org/wayland/wayland-protocols/-/issues/39#stalled-upstream-protocol-work>

²²https://dcz_self.gitlab.io/posts/input_method/

²³https://dcz_self.gitlab.io/posts/not_keyboard/

²⁴<https://gitlab.freedesktop.org/wayland/wayland-protocols/-/issues/39>