Reference Manual

Generated by Doxygen 1.6.3

Sat Apr 9 22:24:22 2011

Contents

1 MeeGo Keyboard Documentation **1**

1.1 General documentation 1

1.2 API reference 1

2 Keyboard Layout XML Specification **3**

2.1 Document Overview 4

2.2 Layouts Overview 4

2.3 Elements 4

2.3.1 Keyboard 4

2.3.2 Import 4

2.3.3 Layout 5

2.3.4 Section 5

2.3.5 Row 5

2.3.6 Spacer 5

2.3.7 Key 6

2.3.8 Binding 6

2.4 DTD 7

3 Styling the keyboard with CSS **9**

3.1 Quick introduction to MeeGo Touch CSS 10

3.2 General appearance: MVirtualKeyboardStyle 10

3.3 The keyboard widget: MImAbstractKeyAreaStyle 10

3.3.1 Font settings 10

3.3.2 Graphical assets 11

3.3.3 Behaviour 11

3.3.4 Key geometry 12

3.3.5 Miscelleanous 13

3.3.6 Debugging 13

Chapter 1

MeeGo Keyboard Documentation

1.1 General documentation

- Keyboard Layout XML Specification
- Styling the keyboard with CSS

1.2 API reference

All Classes

Chapter 2

Keyboard Layout XML Specification

Version: 0.1

2.1 Document Overview

This document provides information about the keyboard layout XML schema described by the DTD `VirtualKeyboardLayout.dtd`. The DTD alone can neither express all constraints nor explain the use of the layout. This is intended to be read alongside the DTD and does not purport to repeat information such as default values sufficiently described by the schema. The DTD is included at the end of the document for convenience but the canonical version is maintained in a separate file.

2.2 Layouts Overview

Layout files are like shops where you can buy keyboards for various purposes, such as inputting normal text, number or URLs. Somewhat confusingly the shop is described by a `keyboard` element and contains `language` (with region) for which the keyboards for sale are intended and a descriptive `title` that is intended for the user for selecting the right shop. The actual keyboards are described by the `layout` element.

Each keyboard may consist of multiple sections, like physical keyboards often have a numpad, arrow keys and alphanumeric in distinct areas. Each section contains one or more rows which again contain keys. Keys may have different label and behaviour depending on modifier (such as shift) state. These behaviours are described using `binding` elements.

Aside `layout` elements one may also have `import` elements. Import is used to include other layout XML files in order to facilitate reuse of layouts or layout parts with multiple languages.

2.3 Elements

2.3.1 Keyboard

Keyboard is the top level element.

Attributes:

- `version` Arbitrary file version.
- `title` Descriptive title based on which a layout file can be selected. For example, "English (UK)". This must always be specified unless the file is intended to be imported only.
- `language` Language and optionally a region in format `<language>[_<region>]` where `language` is a ISO 639-1 code and `region` a ISO 3166-1 code. This must always be specified unless the file is intended to be imported only.
- `catalog` The translation catalog used to localize the labels. Unused as of this writing.
- `autocapitalization` This attribute defines whether autocapitalization could be used or not.

2.3.2 Import

Import includes a `file` that complies to the `VirtualKeyboardLayoutDTD.dtd`. The `keyboard` element in the imported file is effectively ignored. That is, one may think of `import` element expanding

to a list of `layout` elements found in the given XML file. **Note:** since the order of `layout` elements is significant, so is the placement of `import` elements.

Attributes:

- `file` Name of the file to import without path.

2.3.3 Layout

Layout describes the actual layout for a given screen orientation. If the layout doesn't have a portrait variant, the software shall use the landscape variant when the device is in portrait orientation.

A layout inherits content from a previously read layout of same type and same orientation, if available. The previously read layout is then forgotten. If such a layout is not available, the layout inherits content from a previously read layout of same type and different orientation, if available. That is, in this case the sections of the previously read layout are effectively copied to this layout. Those inherited sections are overridden by sections within the current layout element.

Attributes:

- `type` The input context in which this layout should be used. Not all types are necessarily supported by the software using the layout.
- `orientation` Device orientation in which this layout should be used.

2.3.4 Section

Section is an area of one or more rows in the layout. All sections may not be visible at the same time. Some sections may have different looking keys. Use of different sections depends on the software for which the layout is intended.

Maemo virtual keyboard requires sections named "main" (normal key area, all keys except those that belong to the function key row), "functionkey" (function key row) and one or more sections with names starting with "symbols " (symbol key sections that are not visible at the same time with the other sections). The part of the name after "symbols " is used as the symbol key set title in the user interface.

Attributes:

- `id` Name of the section. See above.
- `movable` Unused as of this writing.
- `type` `Non-sloppy` type means that accurate input method (no error correction) is used with keys of this section by default.

2.3.5 Row

Row element describes a row of keys on the keyboard. It has no attributes, just keys and spacers. A row may be empty.

2.3.6 Spacer

Spacers can be used for alignment purposes. They distribute all remaining space of a row among them, after the possible stretcher button has been dealt with.

For instance, to get a center-aligned row, put spacer elements in the beginning and end of a row. Spacer elements are cumulative, i.e., <spacer><spacer> will get twice the space of a single spacer.

2.3.7 Key

Key element describes a visible key on the keyboard.

Attributes:

- `style` The style type this key should use. The style type is defined in the css file under theme/ directory.
- `width` The width of a key. The width itself can be defined in the keyboard's CSS file.
- `fixed` If false, this button uses a relative width. Otherwise uses fixed width.
- `rtl` If true, this button uses RTL version of icon.
- `id` Unique identifier of the key, optional. Needs to exist for keys that can be customized by the application.

2.3.8 Binding

Binding describes the look and the behaviour of a key when current modifier state matches the `shift` and `alt` attributes. Attributes affecting the behaviour are `action`, `label`, `cycleset` and . Attributes affecting the look are `action`, `label`, `secondary_label`, `accents` and `accented_labels`. `extended_labels` provides additional labels/characters that are often accented characters and displayed using a mechanism different from normal label/secondary label/accented label display.

Attributes:

- `action` Action is the primary attribute for determining what the key does. Various actions are listed separately below.
- `shift` If true, this binding applies when the shift modifier is active. If there is no binding with a true `shift` attribute, binding with false `shift` attribute shall be used when shift modifier is active.
- `alt` Just like `shift`, except for the alt modifier.
- `label` Contains text that is displayed as the keyboard label except when either a) action is not `insert`, `cycle` or `plus_minus_toggle` or b) a dead key whose label can be found in `accents` attribute has been clicked. If the `action` is `insert`, `label` also determines the text that is inserted when the key is clicked. The text can be one or more characters.
- `secondary_label` This is displayed next to the text of `label` attribute in small font. Intended for phone number keyboards.
- `accents` Contains a string of characters of equal length to the `accented_labels` attribute. For each character in the string there should be a binding that has that character as the `label` and a true `dead` attribute.
- `accented_labels` This is a string of characters of equal length to the `accents` attribute. If a dead key has been clicked and its `label` can be found in the `accents` attribute, the index of that character in `accents` is used to index `accented_labels` string and the found character is temporarily used in place of the value of the `label` attribute.

- `extended_labels` A string of characters to be made accessible in addition to other *_labels attributes.
- `cycleset` This is used together with the `cycle action` to provide multitap functionality for the key. That is, repeated key clicks cycle through the characters in the `cycleset` string.
- `quickpick` Setting quickpick value to true will make the key close the current keyboard view. Currently only supported in symbol view. This only works if the key is pressed first after symbol view is opened.

Actions:

The precise semantics of each action are determined by the software using the layout and are out of the scope of this document. The descriptions here are brief and describe only a part of the functionality at a high level.

- `insert` Insert value of `label` attribute or character from `accented_labels`, if applicable.
- `shift` Shift modifier.
- `backspace` Delete character or word to the left.
- `space` Insert space character.
- `cycle` Insert first character of `cycleset` attribure or replace the previously inserted character with the following character of `cycleset` attribute.
- `layout_menu` Open layout configuration menu.
- `sym` Open symbol section.
- `return` Insert newline, accept the input or whatever is applicable in the context.
- `decimal_separator` Insert , or . depending on the locale settings.
- `plus_minus_toggle` Toggle the sign of the number being typed.
- `commit` Commit original user input string.

2.4 DTD

Chapter 3

Styling the keyboard with CSS

3.1 Quick introduction to MeeGo Touch CSS

MeeGo Touch CSS mostly looks like regular CSS, though it only supports a subset of the CSS standards. Here are some things to keep in mind:

- Attributes must always be terminated by a semicolon.
- Each CSS class FoobarStyle usually maps to a widget named Foobar.
- Each theme provides constants that can be used as attribute values. Useful for theme-wide defaults, e.g, $COLOR_FOREGROUND. Check `/usr/share/themes/base/meegotouch/constants.ini` for more.
- Each CSS class comes with three inbuilt modes:
  - FoobarStyle {...}: Default mode.
  - FoobarStyle.Landscape {...}: Overrides default when in landscape orientation.
  - FoobarStyle.Portrait {...}: Overrides default when in portrait orientation.
- Attribute values are usually converted to certain Qt types internally. Some convenience units exists:
  - "mm" uses the device's resolution and DPI to convert into device-dependent pixels.
  - "%" uses the device's display size as a base value for the percentage.
- MeeGo Touch splits each image into nine tiles (3x3 grid) before drawing. This is useful for scaling, as the four corners can be fixed whereas the remaining five tiles are used for scaling without introducing artefacts. The size of the corner tiles can be specified for each image, as a rectangle (left top right bottom): `background-image: "meegotouch-keyboard-background" 15px 15px 15px 15px;`

Further information can be found in the MTF styling documentation.

3.2 General appearance: MVirtualKeyboardStyle

Common settings can be changed in the MVirtualKeyboardStyle section, for example the notification font which pops up when the language or plugin itself changed.

3.3 The keyboard widget: MImAbstractKeyAreaStyle

This CSS class exposes many attributes, some of them useful for debugging even.

3.3.1 Font settings

- font: Specifies the font for all key labels.
- font-color: Specifies the font color.
- secondary-font: Only used for keys that have a secondary label, e.g., phone number layouts.

3.3.2 Graphical assets

- background-image: Specified the background of the keyboard. Can be translucent.
- key-background: Specifies the key's default background, i.e., when not pressed. Key backgrounds are supposed to be opaque.
- key-background-disabled: Specifies the key's background when key is disabled.
- key-background-pressed: Specifies the key's background when key is pressed.
- key-background-seleced: Specifies the key's background when key is selected (e.g., caps-lock state for shift key).
- key-background-pressed-seleced: Specifies the key's background when key is pressed whilst being selected.
- The other attributes are the cross product of {key-background} x {higlighted, special, special-highlighted, deadkey, deadkey-highlighted} x {, pressed, disabled, selected}. Keys can be highlighted dynamically (determined by application), whereas special and deadkey are specified as the key's style attribute in the XML layout files.
- key-{backspace, menu, enter, shift, tab}-icon-id: Specifies icon for backspace, menu, enter, shift or tab key.
- key-{backspace, menu, enter, shift, tab}-icon-id-rtl: Specifies right-to-left icon variant for backspace, menu, enter, shift or tab key.
- key-{backspace, menu, enter, shift, tab}-icon-size: Specifies the icon size for backspace, menu, enter, shift or tab key.
- key-shift-uppercase-icon-id: Uppercase variant for shift key.

3.3.3 Behaviour

- long-press-timeout: How long to press key (in ms) to be recognized as long-press.
- idle-vkb-timeout: How long (in ms) does it take for the keyboard to return into idle state, after switching into a so-called speed-typing mode (triggered by pressing two keys in quick succession, for example}. Measured from the point of last key {press, move, release} event. Certain gestures are only available in idle state.
- flick-gesture-timeout: How long (in ms) should the keyboard try to recognize a gesture as a flick gesture, measured from gesture start.
- flick-gesture-threshold-ratio: Distance in [0, 1] of keyboard size (vertically and horizontally) that a finger needs to move to trigger this gesture.
- touchpoint-vertical-offset: Subtracted from y coordinate of a touchpoint, to be adjusted depending on display accuracy (there is usually a human bias to hit lower than what is assumed, due to the aiming with finger tips).
- enable-overlay-mode: In overlay mode, reactive area of first/last row spawns to top/bottom of display. Does not affect key geometry.

3.3.4 Key geometry

Each key has an area - specified by width and row height (all keys in one row share same height but can have different width) - and margins (left, top, right, bottom). Area and margins define the reactive area (all touch events inside a reactive area are handled by that key) and the spacing between keys. The one exception are the paddings of the keyboard widget, which override the key margins like so:

- padding-left: Overrides key-margin-left for left-most keys, in each row.
- padding-top: Overrides key-margin-top for top row.
- padding-right: Overrides key-margin-right for right-most keys, in each row.
- padding-bottom: Overrides key-margin-bottom for bottom row.

The other key geometry attributes are as follows:

- use-fixed-key-width: Can be true or false and determines whether keys are auto-sized regarding their relative key width in order to consume the full row width. If true, no auto-sizing takes place and the key-width-{small, medium, large, x-large, xx-large, stretched}-fixed variants are used. For the auto-sizing, the row with the most keys is used to calculate a key width unit. All other key width values (but not the fixed ones) are relative to that key width unit.
- key-width-small: Percentage between [0, 1] for <key width="small" ...>
- key-width-medium: Percentage between [0, 1] for <key width="medim" ...>
- key-width-large: Percentage between [0, 1] for <key width="large" ...>
- key-width-x-large: Percentage between [0, 1] for <key width="x-large" ...>
- key-width-xx-large: Percentage between [0, 1] for <key width="xx-large" ...>
- key-width-stretched: Percentage between [0, 1] for <key width="stretched" ...> A stretched key would usually consume all remaining width in a row, so this attribute is currently useless.
- key-margin-left: Left margin of key.
- key-margin-top: Top margin of key.
- key-margin-right: Right margin of key.
- key-margin-bottom: Bottom margin of key.
- padding-left: Left padding of keyboard widget.
- padding-top: Top padding of keyboard widget.
- padding-right: Right padding of keyboard widget.
- padding-bottom: Bottom padding of keyboard widget.
- size: Specifies size of keyboard width (w, h). -1 allows to take as much space as needed to fit all keys into it.

3.3.5 Miscelleanous

- sync-style-mode-with-key-count: A boolean. If true, special style modes that are based on the amount of keys in a given layout can be used. Discouraged for normal use but might be necessary for pixel-perfectness. The special style modes can be triggered by layouts containing 10-15 or 30-45 keys. The special style modes can be accessed like so: `MImAbstractKeyAreaStyle.Landscape:keys36 {...}`. In this case, a layout with 36 keys, when in landscape mode, would use overrides from this section on top of the regular landscape overrides for the default mode.

3.3.6 Debugging

The following attributes are useful for debugging only. Some of them draw extra information on the screen, whereas others dump the information into logfiles.

- draw-button-bounding-rects: Boolean. Whether to draw the key's bounding rectangle (key area together with key margins). Also displays size.
- draw-button-rects: Boolean. Whether to draw the key's area. Also displays size.
- draw-reactive-areas: Boolean. Whether to draw reactive areas of keys. Usually similar to bounding rectangles.
- debug-touch-points: Boolean. If true, dumps all touchpoint information into a logfile, found at `$HOME/.meego-im/<PID>-touchpoints.csv` The format is a CSV file, with the following columns:
  - time (sec.msec): Timestamp.
  - tp_id: Touch point id (useful if multiple touchpoints are active).
  - tp_state: One of pressed, moved, released.
  - start_x, start_y: Start position of touchpoint.
  - last_x, last_y: Last seen position of touchpoint.
  - current_x, current_y: Current position of touchpoint. current_i-1 = last_i.
  - center_x, center_y: Center of touched key, if any.
  - delta_x, delta_y: How much does the current touchpoint deviate from key center.
  - label: Label of currently touched key, if any.
  - label_last: Label of last touched key, if any (with respect to touch trace).
  - br_x, br_y, br_w, br_h: Bounding rectangle of current key (key area + margins)