ijprest / keyboard-layout-editor Public

Code Issues 147 Pull requests 20 Actions Projects Wiki Secu

Serialized Data Format

Edit New page Jump to bottom

Ian Prest edited this page on Jul 3, 2019 · 35 revisions

This page describes the serialized data format used by keyboard-layout-editor.com.

If you're interested in reading KLE layouts, check out kle-serial, a permissively-licensed library specifically for this purpose!

Top-Level Structure:

The serialized data is a JSON array of keyboard "rows", with an optional JSON object describing keyboard metadata.

```
[
  { /*keyboard metadata; optional*/ },
  [ /*row 1*/ ],
  /*...*/
  [ /*row n*/ ]
]
```

Keyboard Metadata:

Optionally, the first entry in the top-level JSON array can be a JSON object describing the keyboard metadata. Supported metadata properties are:

- `author` (string); the name of the author
- `backcolor` (string); the background color of the keyboard editor area (default `#eeeeee`)
- `background` (object; optional); `{ name: "name", style: "background-image: url(...)" }`
    - Overrides `backcolor` if specified.
    - The `name` identifies it from the list of backgrounds in the editor.
    - The `style` is some custom CSS that will override the background color.
    - It is theoretically possible to use a custom image, but the editor doesn't currently support this.
- `name` (string); the name of the keyboard layout
    - Appears in the editor, below the keyboard.
    - Identifies the keyboard among your saved layouts.
    - Used to generate a filename when downloading or rendering the keyboard.
- `notes` (string); notes about the keyboard layout, in GitHub-flavored Markdown.
- `radii` (string); the radii of the keyboard corners, in CSS `border-radius` format, e.g., `20px`.
- `switchBrand`, `switchMount`, `switchType` (string); the *default* brand/mount/type of switches on your keyboard

- Default can be overridden on individual keys.
- See known values here: https://github.com/ijprest/keyboard-layout-editor/blob/master/switches.json

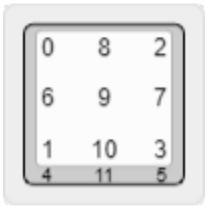
Keyboard Rows:

Each keyboard row in the serialized data is a JSON array of keycaps, each of which is a string:

```
[ "Q", "W", "E", "R", "T", "Y" /*etc*/ ], // row 1
[ "A", "S", "D", "F", "G", "H" /*etc*/ ], // row 2
// ...
```

If a keycap has multiple legends, each legend is separated by a newline character `\n` . The order of legend positions is as follows:

```
[{a:0,w:1.5,h:1.5},"0\n1\n2\n3\n4\n5\n6\n7\n8\n9\n10\n11"]
```



(Note that if the alignment flag `a` is something other than zero, the legend positions will differ. Due to the evolution of the format over time, the default alignment is `4` .)

The string of the keycap is cut off at the last non-empty legend. For example, a keycap that is labeled 'Q' at the top left and '1' at the top right is encoded by the string `"Q\n\n1"` .

When processing keys within a row, the *current position* is updated after each key:

- The first row starts with coordinate *y = 0* by default; each subsequent row increments the *y* coordinate by 1.0.
- Each row resets the coordinate *x = 0*.
- After each keycap, the current *x* coordinate is incremented by the previous cap's width.

Keycap properties can be modified by inserting a JSON object into the row before the key(s) to which it applies, e.g.:

```
[
  { "c" : "#ff0000" }, // set keycap color to red
```

```
    "Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "[", "]",
    { "w" : 1.5 }, // set keycap width to 1.5u
    "\\"
  ]
```

Note that some of the properties apply only to the *next* keycap, while other properties apply to *all subsequent* keycaps. The following properties apply only to the *next* keycap:

- `x` , `y` (number)
  - These specify *x* and *y* values to be *added* to the current coordinates.
  - For example, specifying *x = 1* will leave a 1.0x gap between the previous key and the next one.
- `w` , `h` (number)
  - These specify the *width* and *height* of the next key.
- `x2` , `y2` , `w2` , `h2` (number)
  - These specify the *offset* and *size* of the second rectangle that is used to define oddly-shaped keys.
- `l` (boolean)
  - Specifies that the next key is a "stepped" key (often used for "Caps Lock" keys).
- `n` (boolean)
  - Specifies that the next key is a "homing" key (which usually renders as a little "nub").
- `d` (boolean)
  - Specifies that the next key is a "decal" (does not render the keycap, only the legends).

The following properties apply to *all subsequent* keycaps:

- `c` (string)
  - Specifies the color of the keycap, in "#rrggbb" format, e.g., "#ff0000" indicates red.
- `t` (string)
  - Specifies the color of the text on the keycap, in "#rrggbb" format.
- `g` (boolean)
  - Specifies that the following keys are 'ghosted'
- `a` (number)
  - Specifies the text alignment. This is a bit field of one or more of the following:
    - 0x01 - Center labels in the X direction
    - 0x02 - Center labels in the Y direction
    - 0x04 - Center the side-printed text
- `f` , `f2` (number)

  - Specifies the primary & secondary font heights, on a scale of 1-9 (default 3)
- p (string)
  - Specifies the profile & row of the keycaps.
  - Supported profiles: DCS, DSA, SA
  - Supported profile "modifiers": SPACE, R1, R2, R3, R4, R5

The following properties are not implemented, but are proposed for future use:

- s (string)
  - Style of the keycap; this is a reference to a keycap prototype.

Note:

While the data format is standard JSON, to reduce the visual clutter and make it more readable and easier to edit, the "raw data" tab in the editor relaxes the normal rules by allowing key names to be specified without quotation marks. It also removes the opening and closing square brackets around the entire array.

If you 'upload' or 'download' the layout as a JSON file, or save it on GitHub, it will adhere to the strict JSON grammar.