Assembled Developer Manual

5/10/2023

Contents

This document contains the full documentation of all underlying functions within the Assembled text editor.

Configuration File Structure

The structure of the configuration file is extremely simple:

Command	Argument
keyboard	keyseq func-name:0x41,66,’c’
start_screen	background_disable

There are two columns separated by a tab.

If a command begins with a ‘#’ symbol, then it is discarded.

Keyboard Configuration

To configure the keyboard, the command must be “keyboard” and the argument must follow this format:

```
keyseq A:B
```

A is the name of the function which should be called when the sequence of keys in B are pressed.

An example of B is:

```
‘A’, 65, 0x41, 0102
```

This is a list of comma seperated values. Values can be ASCII characters (wrapped in single quotes), decimal values, hex values (prefixed with “0x”), and octal values (prefixed with “0”).

These values all describe ASCII codes.

Start Screen Configuration

To configure the start screen, the command must be “start_screen” and there are currently two arguments

background_off

This disables the background generation within the

logo_bmp:A

A is the path to the 64x40 monochrome BMP to be used as the logo.

Source Code Directory Structure

The directory structure in implementation currently looks like the following list:

- src/
  - editor/
    - config.c
    - functions.c
    - keyboard.c
  - include/
    - editor/
      - config.h
      - functions.h
      - keyboard.h
    - interface/
      - screens/
        - start.h
      - interface.h
    - global.h
    - util.h
  - interface/
    - screens/
      - start.c
    - interface.c
  - main.c
  - util.c

The program can be split into two parts: update and render. The editor part defines functions which are used for the functionality of the program, and interface defines functions which are used to display information to the user.

An example of functionality is the keyboard. The user presses a key, which the program should interpret, this interpretation is handleld by *keyboard.c*.

Likewise, an example of rendering is the screen interface. There are various different screens which the user can access and the program needs to somehow manage which screen is being rendered and updated.

main.c is used for concatenating the two halves into one functioning program.

Editor Half

The editor is responsible for handling non-visual tasks. Examples of non-visual tasks are:

- Reading the configuration file
- Reading user input
- Interpreting user input
- Updating internal text buffers

Currently there are three source files in the editor directory: *config.c*, *functions.c*, *keyboard.c*.

config.c

This is a very straight forward file, it goes through the configuration file, *~/.config/assembled/config.cfg*, line by line. It forwards the text in the argument column to the component specified in the command column.

The current command is hashed using the *general_hash* function, and the resultant value is used in a grand switch statement.

functions.c

functions.c contains the implementation of a hashmap of functions, and the functions themselves.

keyboard.c

Contains a series of functions used for submitting and interpreting key presses, as well as a function to set up, configure, the key layers which are used to interpret each key press.

Run-time

When a key is pressed, the *key* function is called. This function pushes the key to the key stack, advances the key stack pointer, and tries to collapse the stack.

Collapsing the stack is done through the *collapse_stack* function. Which will iterate from the bottom of the stack to the top, indexing each element with the letter *i*. It will use the current value it observes at *i* as the index into the function array member of the current key layer.

If the function pointer that is retrieved is NULL, the stack pointer is reset and interpretation is stopped.

If the function pointer that is retrieved is equivalent to *layer_down* then the current key layer will be swapped to the next, and the loop will be put onto its next cycle.

Finally, if the loop has found a function which is non-NULL and not a *layer_down* then it will jump to that function, Reseting the key stack pointer just before leaving the *collapse_stack* function.

Configuration

keyboard.c is configured through the use of the *init_keyboard* function. This function evaluates the given string in the following manner:

- Is this a key sequence or key bind? (Neither, return)
- What is the name of the function that is to be bound to the given set of keys
- Bind the function to the given set of keys

First it checks if the configuration is for a key sequence or a key bind (currently, only key sequences are supported). If the configuration is for neither, then the function wiill return. Next, the name of the function to be called is retrieved and looked up in *functions.c*'s hash table.

Finally a recursive function, *create_path*, is called to go through each value and has the responsibility of allocating new layers, and setting the appropriate function pointer for each value.

If the number of values in a given sequence is greater than or equal to two, then all values preceding the last value will be set to the pointer of the *layer_down* function, with only the last one being set to the desired pointer.

void key(c)

Push the given character, c, to the key stack.

c – Character to push to the key stack

char

void init_keyboard(line)

Interpret the given line and apply proper changes to internal data structures.

line – Line of configuration relevant to the keyboard

char *

Interface Half

The interface manages all possible screens which can be displayed to the user at any time.

interface/interface.c contains two functions: register_screen and switch_to_screen.

Screens should have a *register*_screen-name function which registers them into the hashmap. Functions *render*, *update*, and *local* should all be static and not defined in the screen's header.

int register_screen(name, render, update, local)

Calculates the index of the screen within the *screens* hashmap, and sets the attributes of the screen structure at that index to the given arguments.

Returns the index of the screen in the hashmap.

name – The name of the screen, used as the key within the hashmap of screens

char *

render – A function pointer to the code which renders the screen to the TUI

void (*)(struct render_context *)

update – A function pointer to the code which updates the screen

void (*)(struct render_context *)

local – A function pointer for handling special cases

void (*)()

Special cases occur for events like navigation keys. When a key mapped to a function such as “up” is pressed, the “up” function is called. This function is then told to call into the local handler of the currently active screen with the code LOCAL_ARROW_UP.

int switch_to_screen(name)

Switches the *active_screen* pointer to point to the specified screen.

Returns the indedx of the screen in the hashmap.

name – The name of the screen to switch to

char *