The Smoothie firmware is a community-developed Open-Source CNC control firmware, mainly aimed at the Smoothieboard Open-Source Hardware community-developed controller board.

One functionality of the firmware, is the ability to interface with a combination of a LCD screen and a control mechanism (buttons or encoder), called a “Panel” in the community.
On this LCD is displayed an interface, which the user controls. This forms a https://en.wikipedia.org/wiki/Human%E2%80%93machine_interface .

This interface is currently entirely coded and stored in the firmware itself (ROM memory). This takes a lot of place, and any modification requires re-compilation. It’s also impractical to implement a multi-language interface, and the dev team is reluctant to add functionality to the interface due to limited ROM space.

Introduction

A solution to this problem has been imagined : store all data (text, menus, interface description, structure) on the SD card, as folders and text files, and have only the most basic/necessary functions actually implemented as code in the firmware.
This would severely reduce ROM and RAM usage of the panel functionality, increase customizability, increase the number of functions, and allow for multi-lingual interfaces.
The main cost would be the initial required coding, and the necessity for the user to add files to the SD card that describe the interface (though this can be done at the factory for most boards).

This document describes a proposed implementation of this solution.
This is done mostly by describing the format of the files that would describe the interface on the SD card.

Functional description

General concept

A menu structure is essentially a tree. Folder-based file structures also happen to be trees. Therefore we can use a folder-based file structure to easily describe as complex a menu structure as we want.

Each folder would contain a list of files, each file describing a single menu entry.

When an entry is selected, a specific action is executed, which can be entering a folder further down, or a special action.

Folder structure

Each folder describes a menu the user can select an entry in.
Each folder contains a list of text files.
Each text file contains a description of an entry in the menu.
The files are named with a number, followed by a dash, followed by a name, followed by the .txt extension

For example :
`02-calibration.txt`

The number is used to display the entries in the right order when displaying the menu. The name is never shown to the user, and is only so that developers can locate entries easily. The .txt extension is just so that it's easier for operating systems to know what to open the file with.

Here is an example of a list of files for a menu folder :

```
01-watch.txt
02-jog.txt
03-prepare.txt
04-custom.txt
05-configure.txt
06-probe.txt
```

File structure

Each file describes an entry in the menu.
This includes what to display as the entry, what action to take if the entry is clicked, and anything special about this specific entry.

Here is what a menu entry file can look like for example :

```
label-en Configure
label-fr Configuration
label-es Configuracion
action   goto-menu /sd/panel/configure/
```

This means that when the users clicks on “configure” (or whatever else is displayed based on their language), the “configure” menu is entered and displayed.

Special variables

Some options can use special variables in them. For example, if a file is currently selected, and you want it displayed in a label, you do :

```
label-en    Delete *s
```

Which will display “Delete test.gcode” if test.gcode is the currently selected file
This is also useful for actions

Special options

Some menu entries need to do some special things.
We tell the menu/panel system this by adding special lines to the menu entry file.
For example :

```
label-en  Suspend
label-fr  Suspendre
label-es  Suspender
action    run-command suspend
only-if-playing-is 1
```

Would only get displayed if a file is currently playing.

Only-if-playing-is

Only display this menu entry if a file is currently playing
Parameter : 0 to display only if not playing, and 1 to display only if currently playing

Only-if-halted-is

Only display this menu entry if the board is currently in the halted state (needs M999 to start back again)

Parameter : 0 to display only if not halted, and 1 to display only if currently halted

Only-if-suspended-is

Only display this menu entry if the board is currently playing a file and that file is suspended
Parameter : 0 to display only if not suspended, and 1 to display only if currently suspended

Only-if-file-is-gcode

Only display this menu entry if the currently selected file is a playable gcode
Parameter : 0 to display only if not playable, and 1 to display only if playable

Is-title

If this appears in the file, display the label in a special way (like : negative, and centered, would be nice). Also stays sticked to the top of the screen.

Not-selectable

If this appears in the file, then this specific line/menu entry is not selectable, and the cursor can not select it or click on it

File-selector

If this is in the file, then this line is replaced by a list of file, and clicking each folder re-loads this screen but displaying that folder instead. Clicking one of the files starts the action specified by the action line.
Two parameters :

- First parameter is where to start exploring the file system
- Second parameter is a path above which the user can not go. For example /sd/ means the user won't ever be able to go to /ext/

Example : `file-selector /sd/gcode/examples/ /sd/gcode/`
If a file is selected, it is remembered so it can be used further down in the menu

Module-specific

Self-explanatory :
`only-if-extruder`
`only-if-temperature-control`
`only-if-laser`

Actions

When a user clicks on a menu entry, a given number of things can happen, based on what the "`action`" line for the entry file specifies.
The menu/panel system reads this line, and "executes" it. This means that it recognizes a given number of actions. This is a list of those actions :

Goto-menu

Enters a new menu, then displays it.

Parameter : the path to the menu folder to enter and display.
Example : `goto-menu /sd/panel/configure/`

Run-command

Run a command through the internal Smoothie system, as if it were received over serial

Parameter : the command to run
Example : `run-command play /sd/calibrate.gcode`

Possible options :

> `display-answer-in-menu /sd/extruder/get-diameter` : Displays the answer to the command, at the bottom of a specific menu designed specifically for that command.

Control-axis

Allows to control a given axis with a given step size
Example : `action control-axis X 10`

Parameters : the axis, and the distance to move each tick
Example : `control-axis Y 10`

Control-extruder

Allows to control a given extruder with a given step size
Parameters : the extruder, and the distance to move each tick
Example : `control-extruder 2 5`

File-select

Allows the user to select a specific file, starting at a specific point, and execute a given menu once the file is selected.
Example : `file-select /sd/gcode/ /sd/panel/file/execute-file/`

Display-watch-screen

Enters and displays the watch screen
(note : this might get scrapped and just have the watch screen implemented as a menu, which would allow users to design it themselves).

Implementation

The general idea of implementing this system, would be to keep the general structure of the current panel code (which essentially separates the functionality into “Screen” objects), but to add new «Screens» that implement the new functionality.
Part of the older screens would then become obsolete and could be removed.

In this new system, any screen that is not currently displayed wouldn’t be stored in RAM.

The current set of screens can be seen at
: https://github.com/Smoothieware/Smoothieware/tree/edge/src/modules/utils/panel/screens

Here is a screen by screen description of what needs to be implemented :

Menu-screen

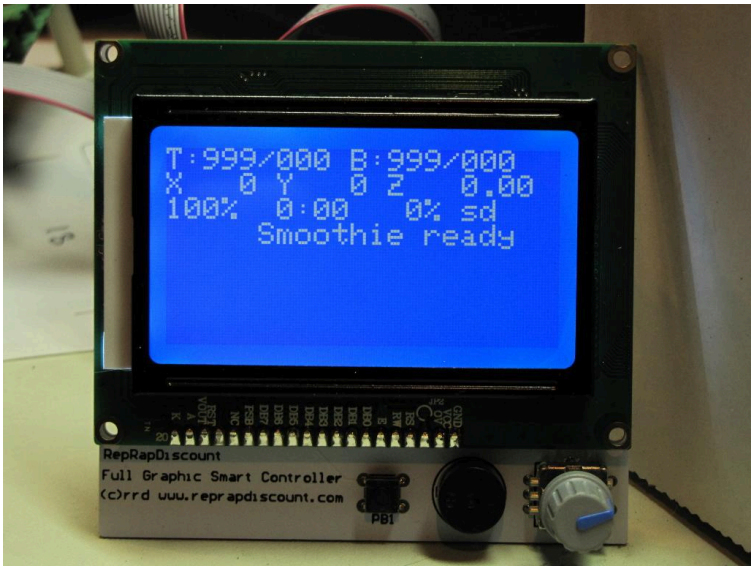
This is the backbone of the new system.
The general idea is : a given folder describes a menu. When that menu is entered, every file in that folder is read, each file describes a line (entry) in the menu. If the screen is updated (for example because the user scrolled down), the screen is entirely refreshed by reading the sd card again.
This assumes that reading the SD card is fast enough for smooth display/updates. If it is not the case, a small cache could be implemented for the text of the current lines, then only the next line needs to be read when scrolling.
For each line, the language is selected based on the currently set language (with English as the default/backup).
Special options for each line also can do special things, like hide the line if a condition is true for example.
Once a line is selected/clicked, the action for that line is read from the file, then executed.
This can get us to another menu, or run a Gcode, or go to another screen etc.

The file selection screen, is not implemented as a separate screen. Rather it is a special type of entry for the menu screen. This allows us to customize the appearance of file selection menus.

This should in theory be capable of replacing most screens.

Status-screen

This is a replacement for the “Watch” screen. It serves the same purpose, but it’s entierely defined in a file.
This means the user can customize it, but also that we can have several different watch screens if we want, and that we can have different implementations for different physical screens.
It would also implementing something cuter than this :



This can’t be read every time it’s updated, so it’d have to be stored in RAM.

Bitmap-screen

This displays a full-screen bitmap file read from the SD card, for a specific duration, then goes to the main menu. It’s mostly useful for the welcome screen and for practical jokes.

Control-axis-screen

Triggered by the control-axis action, allows to jog a given axis by a given increment.

Edit value screen

Allows the edition of a value, either in the config file, or via a gcode.
Is provided with the config value, or the gcode, as well as minimum and maximum values, or a list of possible values, etc.

Example menu structure

An example implementation of a menu structure can be seen here :
https://github.com/arthurwolf/Panel-Menu (work on-going).
Mostly working on it so I can figure out what needs to be added to this specification while doing it.

Special “Playing” menu

If Smoothie is “playing” a file from the SD card, we can not read files from the SD card, which means we can not read the menu files.
This means that whenever Smoothie is playing, the panel, instead of displaying the normal “menus”, should display a special “playing” menu, which shows the current print progress, and allows users to do a few simple ajdustments of speed, temperature and extrusion ratio.