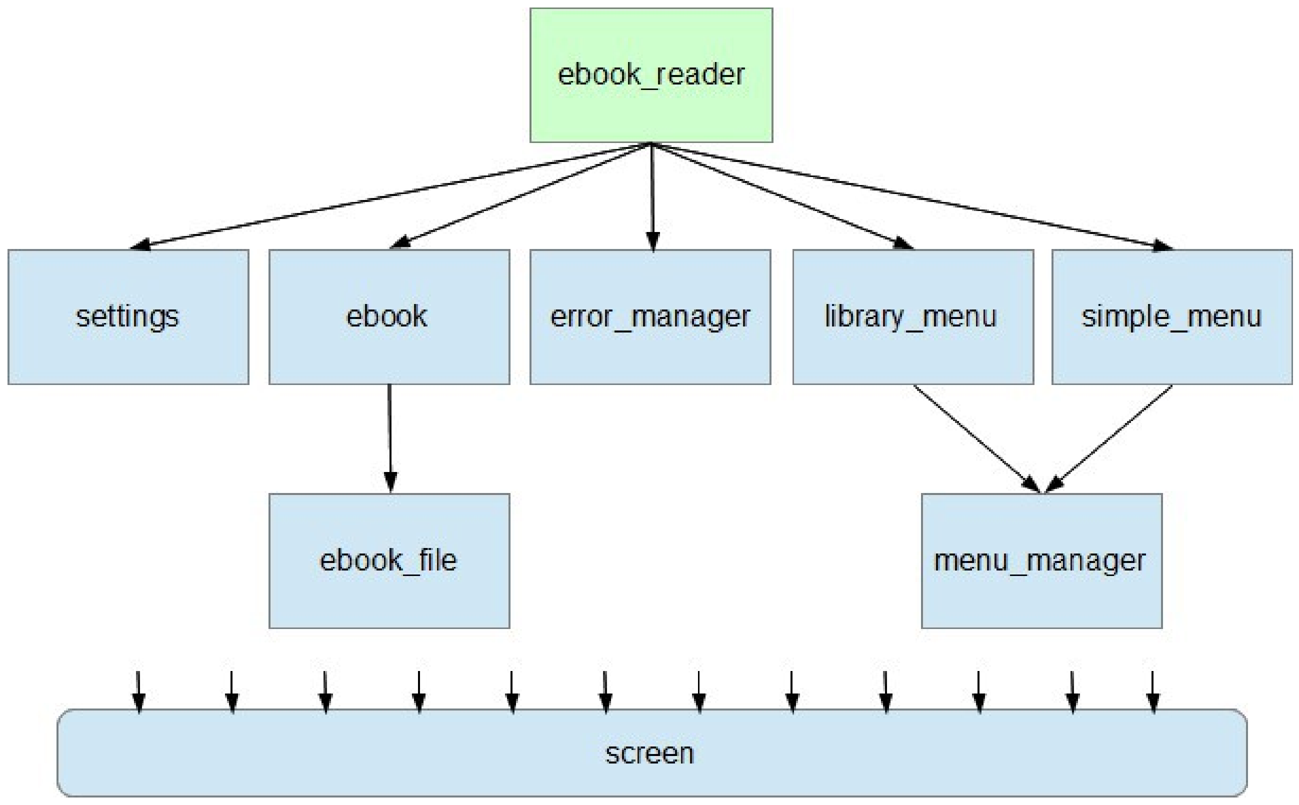
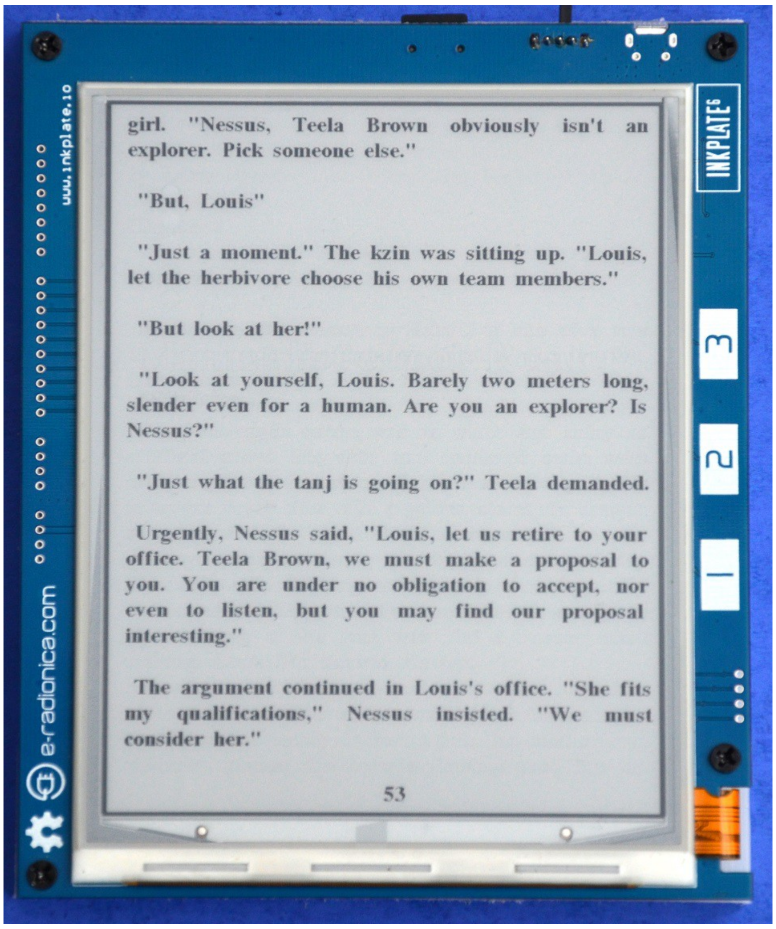
# Minimalist eBook Reader for InkPlate 6



## Contents

# Minimalist eBook Reader for InkPlate 6



girl. "Nessus, Teela Brown obviously isn't an explorer. Pick someone else."

"But, Louis"

"Just a moment." The kzin was sitting up. "Louis, let the herbivore choose his own team members."

"But look at her!"

"Look at yourself, Louis. Barely two meters long, slender even for a human. Are you an explorer? Is Nessus?"

"Just what the tanj is going on?" Teela demanded.

Urgently, Nessus said, "Louis, let us retire to your office. Teela Brown, we must make a proposal to you. You are under no obligation to accept, nor even to listen, but you may find our proposal interesting."

The argument continued in Louis's office. "She fits my qualifications," Nessus insisted. "We must consider her."

53

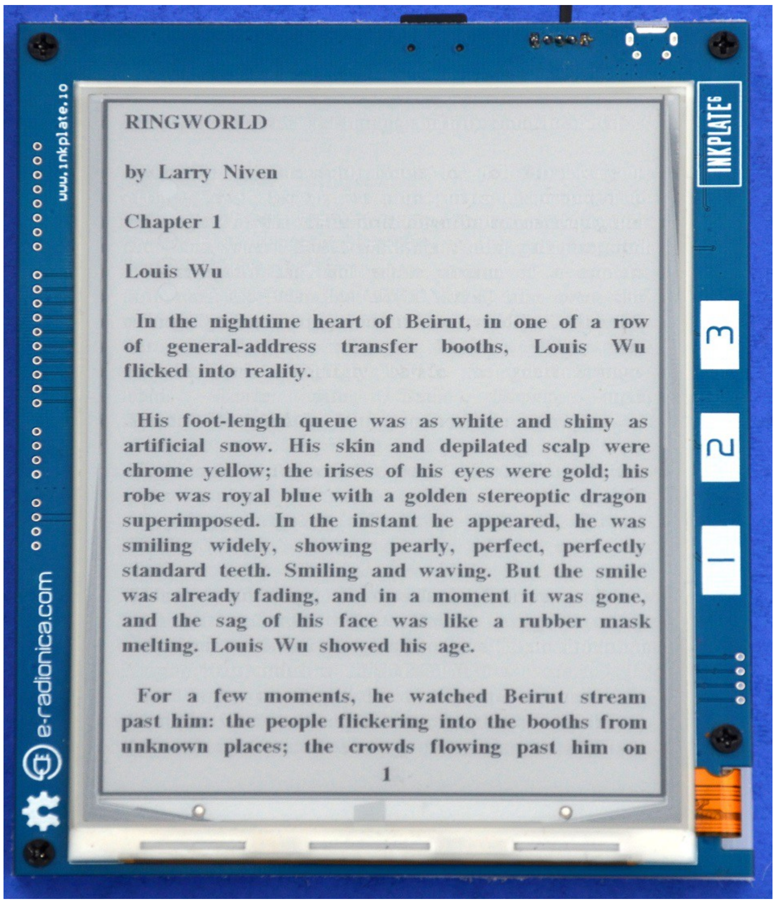
InkPlate 6 is a 600 x 800 pixel e-ink screen from a recycled Kindle, mounted on a board with an ESP32 module. The board also includes an SD card socket, a USB connector and a charging system for a lithium battery. Although the screen does not provide a touch interface, three separate touch pads are mounted along one edge of the screen. The aim of this project was to produce a minimalist eBook reader using this hardware.

eBook readers, e.g. Kindle, Kobo and PocketBook, allow eBooks in various formats to be read, but typically support other functions such as web browsing, buying eBooks online and even playing chess. For the purposes of this project such extra functions are considered superfluous – after all most people have other computers to perform such functions, even if only the toy ones they keep in their pockets. This minimalist eBook reader will just provide the functionality required to read the text of books stored on an SD card. It could be the basis for more advanced projects.

The most common open eBook format is EPUB. An EPUB book file essentially contains a description of a website written in HTML and CSS, which an eBook reader can divide up into pages. This allows books to be formatted in a wide variety of ways and includes support for images, diagrams, links and tables. It is much more complex than is required to simply present the text of a book. For this reason, the Minimalist eBook Reader uses eBooks formatted as simple text files.

The format used is simply a sequence of paragraphs. Each paragraph is a sequence of words separated by spaces and terminated by a single newline character. It is striaghtforward to produce such text files from EPUB or other eBook formats using the free Calibre eBook management software. This can convert any format it supports into a simple text file which can be further simplified into the form required here by a few global replacements using any capable text editor, e.g. Notepad++. The process will be described in more detail later in this document.

## Using the eBook Reader



RINGWORLD

by Larry Niven

Chapter 1

Louis Wu

In the nighttime heart of Beirut, in one of a row of general-address transfer booths, Louis Wu flicked into reality.

His foot-length queue was as white and shiny as artificial snow. His skin and depilated scalp were chrome yellow; the irises of his eyes were gold; his robe was royal blue with a golden stereoptic dragon superimposed. In the instant he appeared, he was smiling widely, showing pearly, perfect, perfectly standard teeth. Smiling and waving. But the smile was already fading, and in a moment it was gone, and the sag of his face was like a rubber mask melting. Louis Wu showed his age.

For a few moments, he watched Beirut stream past him: the people flickering into the booths from unknown places; the crowds flowing past him on

1

The InkPlate 6 provides three touch pads. In normal reading mode two of these provide forwards and backwards page turning and the third is used to display a simple menu. The menu allows the reader to return to the first page, or to move the furthest page read. It also provides access to the library of eBooks stored on the SD card, and to a system information screen which displays details such as the current battery voltage.

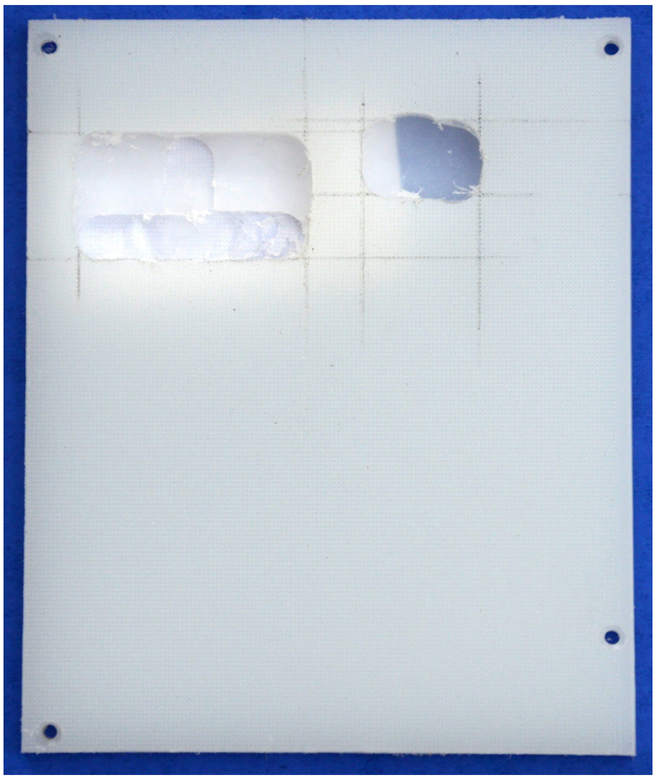
The library display shows the names of all the eBook files stored on the SD card and allows a new book to be selected.

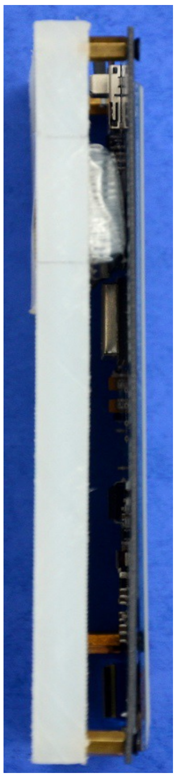
When the reader is switched off, it stores the details of the current book, current page and an index of pages between the first page and the furthest page read. When the reader is switched on, it restores these details and reselects the last displayed page.

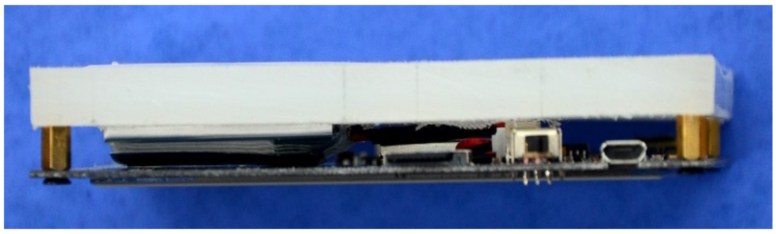
In the spirit of minimalism, there are no functions for managing the eBook library beyond being able to select a book to read. Any computer able to read and write SD cards can be used to add books to the library or to remove them.

## Hardware

The hardware is consistent with the minimalist approach, and consists of the InkPlate 6 board fixed to a simple back plate made from a sheet of 8mm plastic (nylon chopping board). This has two cavities roughly milled into it to house a 1000ma/hr lithium battery and to provide clearance for the battery socket. There are four 3mm holes to accommodate countersunk M3 screws that fit the four hexagonal spacers supplied with the board. These holes are drilled out on the PCB side to 5.5mm, to a depth of 4mm, to locate the spacers securely. No designers employed by Apple were involved in the design of this "case". The following images show the back plate, with and without the InkPlate attached.



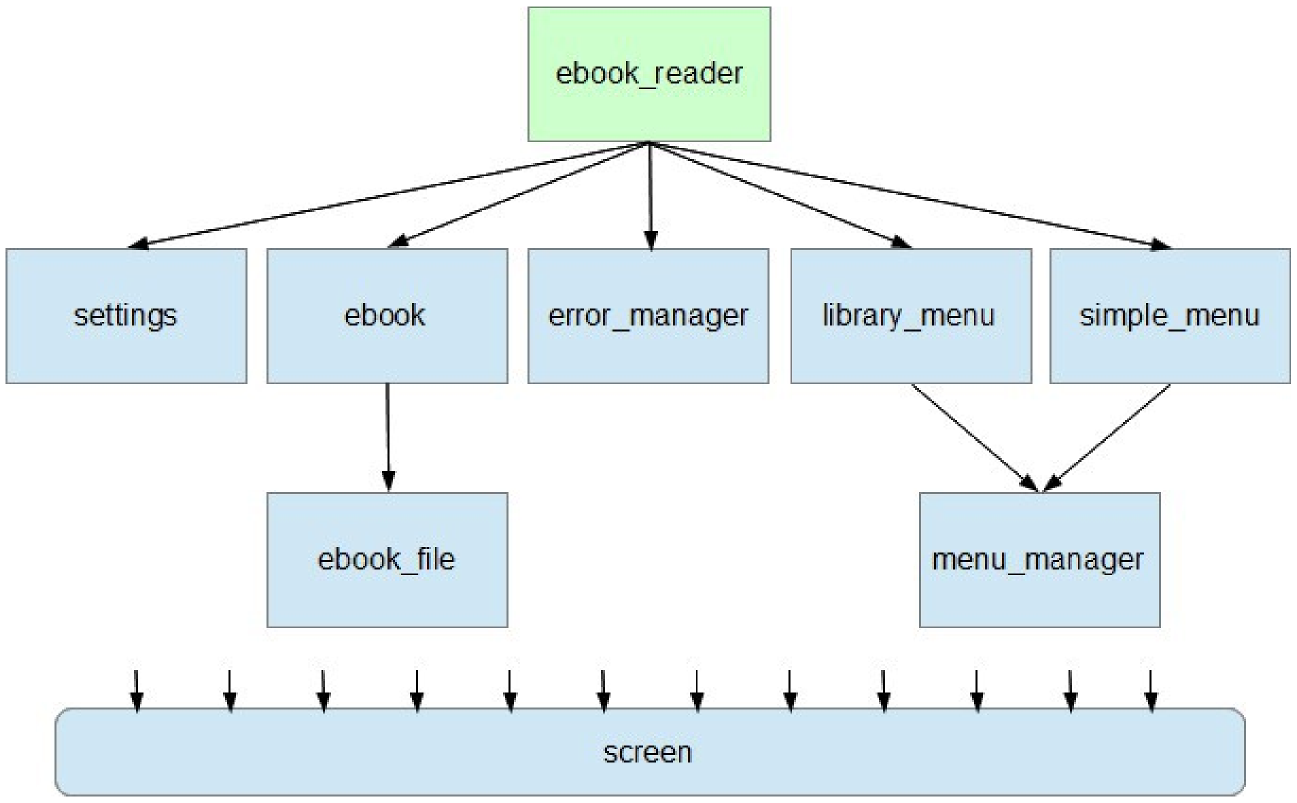




## Software Structure

The reader software is written in C++ and can be built and uploaded to the device using the Arduino IDE. It relies on the InkPlate library provided by e-radionica.com.

The main program, or sketch to use the Arduino terminology, manages the user interface, turning pages, displaying a menu, etc. This is supported by classes which represent the eBook being read, a menu system, a library menu, error management and stored settings management. In addition there are some extra functions to simplify access to the display and the touch pads. The relationship between these is shown in the following diagram.



## Settings

The **settings** class uses the standard Preferences library to store reading related settings in the EPS32's non-volatile memory. It has the following interface.

```
void get_current_book(char* current_book, int max_length);
```

Retrieve the currently open book file path.

```
void set_current_book(const char* current_book);
```

Store the currently open book file path.

```
void set_book_details(int current_page, int pages, long page_index[]);
```

For the currently open book store the current page, number of pages read and the index of page starts within the current  book file.

```
void get_book_details(int& current_page, int& pages, long page_index[]);
```

For the currently open book retrieve the current page, number of pages read and the index of page starts within the current  book file.

## eBook

The **ebook** class is responsible for paginating and displaying selected pages within the current book. It does this by reading words from the selected book file, one character at a time, until the end of paragraph marker (a newline character) is reached, or the bottom of the page is reached, or the end of the file is reached. As each word is read, it's width in pixels is calculated. If the word can be added to the current line without overflowing the right hand margin, the word is added to the line and the total width of the line is updated. When enough words have been read to fill a line, or the end of paragraph is reached, the line is output to the display. If a word if read that will not fit on the end of the line without it overflowing, the file is rewound to the beginning of the word so that it can be included in the next line. The ebook class uses the **book_file** class to read individual characters. This provides a rewindable character stream.

Words are not displayed as they are read in, but instead are stored in a character array with an index which stores the starting position of each word in the array. This allows the line display function to compute the amount of padding required between words to produce right aligned text before it is displayed.

A further index is created to store the file positions of the initial characters of each page. The page index is updated every time the user advances to a page that has not yet been read. This allows the user to move backwards and forwards between the first page and the furthest page read without having to paginate the entire book when it is opened, i.e. pagination is performed incrementally as the user advances through the book.

The ebook class implements the following interface.

```
void set_font(const GFXfont* book_font);
```

Set the font used to display the book. If **nullptr** is passed for the **book_font** the InkPlate's default font is used.

```
void new_book(const char* path, bool from_beginning);
```

Opens a book using the file at the indicated path. If the **from_beginning** parameter is true, the first page of the book is displayed. If it is false, the current page and the page index are retrieved from the settings and the current page is displayed.

```
void display_page();
```

Redisplays the current page, e.g. after a menu has has been displayed.

```
void first_page();
```

Sets the current page to the first page and displays it.

```
void furthest_read_page();
```

Sets the current page to the furthest read page and displays it.

```
void next_page();
```

Sets the current page to the next page and displays it.

```
void previous_page();
```

Sets the current page to the previous page and displays it.

```
int get_current_page();
```

Gets the current page number.

```
int get_furthest_read_page();
```

Gets the number of pages read.

## Ebook File

The **ebook_file** class provides rewindable access to character sequences from text files stored on an SD card plugged into the InkPad. Internally this uses the SdFat library provided along with the main InkPlate library. The **ebook_file** class implements the following interface.

```
bool open_file(const char* path);
```

Opens the file at the indicated path. It returns true if successful and false otherwise.

```
void close_file();
```

Closes the file.

```
bool end_of_file();
```

Returns true if the last character in the file has been read and false otherwise.

```
char get_ch();
```

Gets the next character from the file. If the end of the file has been reached it returns the null character.

```
long file_position();
```

Returns the zero based position of the next available character in the file.

```
void set_position(long new_position);
```

Set the position in the file of the next available character.

## Error Manager

The error_manager class stores the current error state of the system using to the following set of values.

```
enum error_state {no_error, no_sd_card, no_library, no_book};
```

```
no_error
```

The default state of the system when nothing has gone wrong.

```
no_sd_card
```

The system has been unable to initialise the SD card, which may mean that the SD card is not present, or that it is damaged or incorrectly formatted. It is often necessary to power cycle the system after inserting a valid card to resolve this.

```
no_library
```

The  system cannot find a directory called “Library” at the top level of the SD card.

```
no_book
```

The system was unable to open an ebook_file at a given path.

In addition to defining the **error_state** values, the **error_manager** class implements the following interface.

```
error_state get_error_state();
```

Returns the current error state.

`void set_error_state(error_state error);`

Sets the current error state.

`void show_error_status();`

Displays a message based on the current error state and waits for the user to tap a touch pad to indicate it has been read.

`void show_error(error_state error);`

Sets the current error state and then displays it immediately.

## Menu Manager

The **menu_manage** class provides the basis for menus use by the reader. It is intended to produce menus consisting of a series of options displayed vertically with circular markers to the left of each. The selected option is indicated by a filled marker the rest are unfilled. The selection can be moved up or down by taping one of the lower two touch pads. The third touch pad is used to activate the selected option. Note: it is up to the user interface to monitor the touch pads and to call methods of this class to move the selection marker.

The class can be supplied with a simple array of character arrays to define the text of the options, and a corresponding array of functions to call when a particular option is activated. A virtual function is provided for displaying the options. This can be overridden to allow the provided character arrays to be formatted in any desired fashion. Alternatively it may be overridden to acquire and display the text of the options independently, e.g. by iterating through the names of files stored in a directory on the SD card.

The display and updating of the currently selected option is managed independently of how the option text is displayed.

The menu_manager class implements the following interface.

`void set_menu_options(char* options[], int option_count);`

Sets text for the options to be displayed.

`void set_menu_actions(menu_action actions[], int action_count);`

Sets actions corresponding to each of the displayed options. The type **menu_action** denotes a void function with no parameters.

`void virtual show_menu_options(int initial_selection);`

Causes the menu options to be displayed with the indicated option selected initially, i.e. the marker to the left of it will be filled. This function is virtual and must be overridden in a sub-class or nothing with be displayed.

`bool virtual do_menu_action();`

Causes the action corresponding to the selected option to be executed. This returns true if the action is executed successfully and false otherwise. This function may be overridden to allow some action specific for a particular kind of menu to be performed. If the default behaviour is used, it is permissible for there to be less actions than displayed options, in which case the last action specified will be executed for all options without corresponding actions, e.g. if only one action is defined using the **set_menu_actions** function, it will be used for every option.

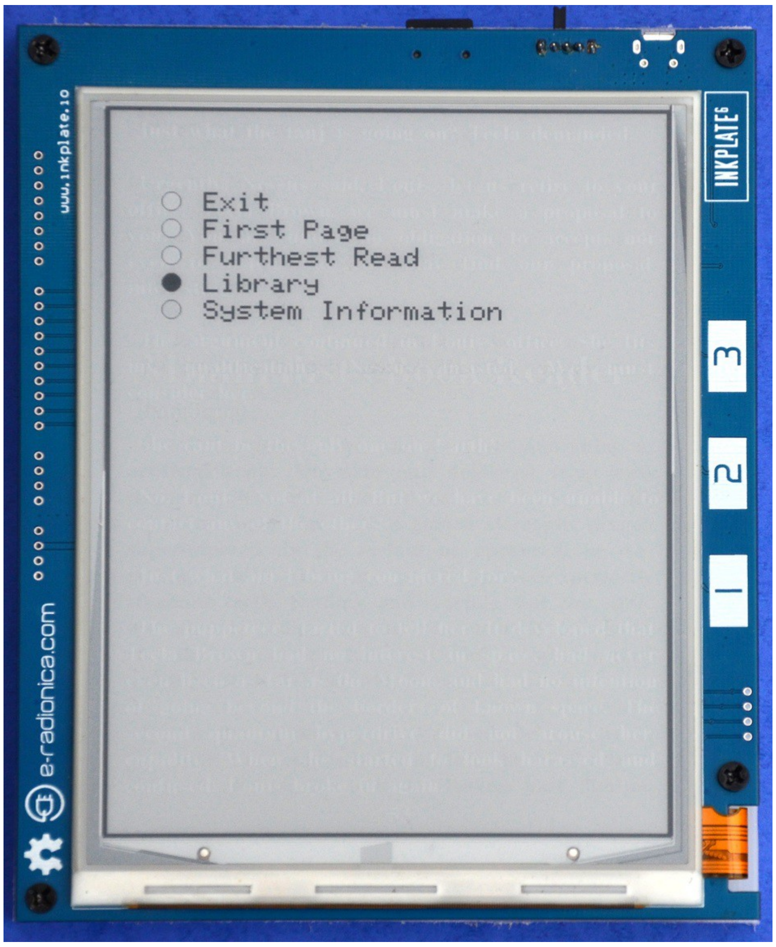
`void select_previous_option();`

```
void select_next_option();
```

These two functions are used for updating the current selection by moving it forwards or backwards and redrawing the markers to the left of the displayed options. The selection is moved in a circular fashion, so that it can wrap around at either end of the menu.

## Simple Menu

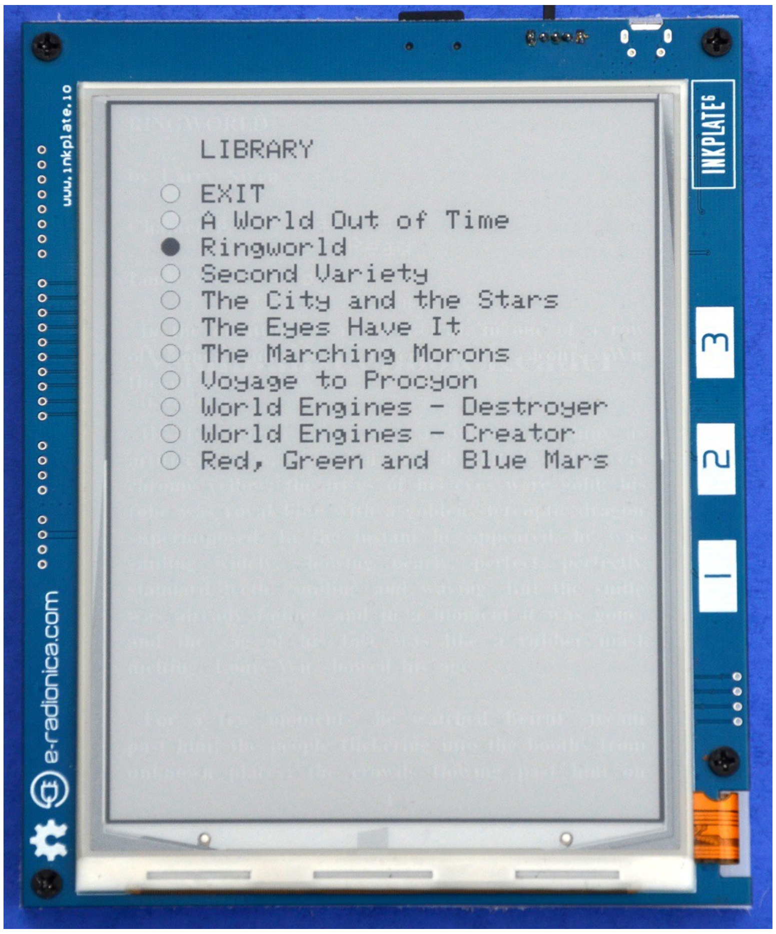
The **simple_menu** class subclasses the **menu_manager** class and overrides the **show_menu_options** function to allow simple textual menus to be displayed. No other functionality is added. The eBook reader uses this class to implement the main menu, which will be displayed when the top touch pad is tapped while reading a page. Because there are only three touch pads, an "EXIT" option must be provided to allow the user to exit from the menu without activating any of the options.



## Library Menu

The **library_menu** class also subclasses the **menu_manager** class to display a selectable list of book titles corresponding to files store on the SD card. To do this it overrides the **show_menu_options** function with a new version, which displays options corresponding to eBook file names from a directory called "Library" at the top level of the SD card. It also overrides the **do_menu_action** function so that when an option is selected, the corresponding book file is opened and the first page of the book is displayed. No attempt is made to sort the book titles. The order in which they are displayed will depend of the way the files are stored on the SD card.

Again, an "EXIT" option is provided so that the user can exit from the menu without opening a new book.



## Screen

The screen module provides extended support for accessing the display and the touch interface. Other components of the system access the functionality it provides by including "screen.h". This provides a global reference to a single instance of the InkPlate library class called **display**, through which all graphics, and interaction is managed.

The module provides the following definitions.

```
const int milliseconds = 1;
const int seconds      = 1000 * milliseconds;
```

Used to allow explicate units to be specified in delays.

```
enum orientation {portrait          = 1, landscape          = 2,
                  inverted_portrait = 3, inverted_landscape = 4
                 };
```

Provides names for the possible screen orientations.

```
enum button {no_button, menu_button, previous_button, next_button};
```

Provides names for the touch pads as used by the user interface independently of the orientation of the display.

```
extern Inkplate*   display;
```

Provides global access to the functionality of the InkPlate library.

```
void   init_screen();
```

Creates an instance of the InkPlate class and assigns a reference to it to the global **display** variable. It then sets the display orientation and various other display parameters.

```
void   set_orientation(orientation new_orientation);
```

Sets the display orientation and the mapping between the physical touch pads and the user interface functions represented by the **button** constants **menu_button**, **previous_button** and **next_button**, e.g. if the orientation is changed from **portrait** to **inverted_portrait**, the top most touch pad will still behave as the **menu_button**.

```
int    font_height(const GFXfont* font);
```

Returns the line height in pixels of text displayed in the indicated font.

```
int    string_width(const char* s);
```

Returns the width in pixels of the characters in the char array **s** when displayed in the current font.

```
void   redisplay();
```

Causes newly defined graphics to be displayed. The eInk screen has two display update modes. One called partial update simply adds graphics to the screen without completely reinitialising it. This is a flicker-free update which does not erase existing graphics. There is also a full update mode which completely reinitialises the screen before displaying new graphics. This causes flicker. However if too many partial updates are applied sequentially, the quality of the displayed image will deteriorate. The redisplay function manages this by counting the number of partial updates it performs and only performs a full update when the count reaches a threshold defined by a constant called **max_partial_updates**.

```
button get_button();
```

Checks if the user is touching a touch pad. If so, it waits for the user to stop touching that touch pad. It then returns the appropriate **button** constant. If the user is not touching a pad the **no_button** value is returned immediately.

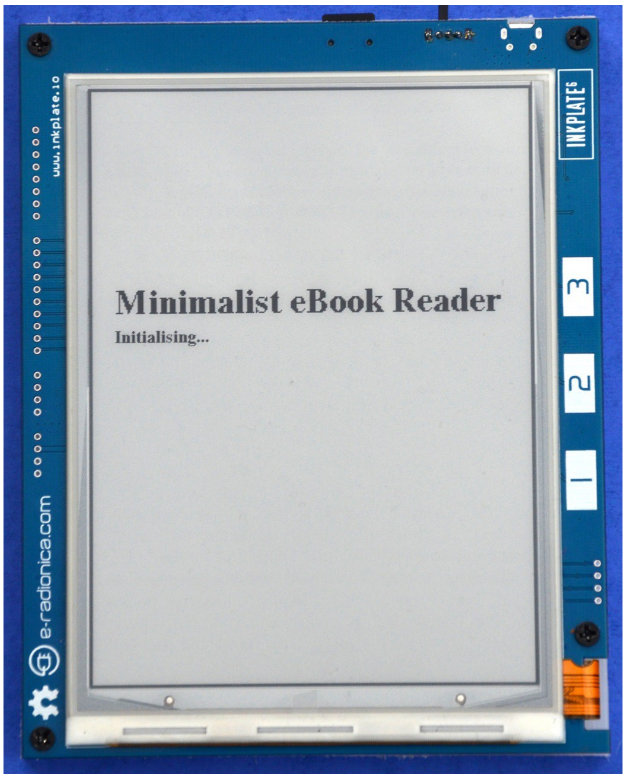
```
void reduce_power(long interval);
```

Removes power from the display (current image is still displayed) and puts the InkPlate into low power mode for the indicated interval in milliseconds. If the interval is too long, response to the touch pads will be impacted.

## The User Interface

The main ebook reader module implements the user interface using the usual Arduino structure with a **setup** function which is called once followed by repeated calls of a **loop** function.

The **setup** function creates instances of the components described above and uses their constructors to connect them as illustrated in the system structure diagram (The components are created on the heap because they are never deallocated so heap fragmentation is not an issue). A start up screen is then displayed.



After this the file path to the current eBook is retrieved from the settings and a new **book** is defined using current page information from the settings. This will cause the last page the user had selected to be displayed. However, the first time the reader is used, no book will have been selected and an error message will be displayed instead. The user can then use the library menu to select a book to read.

The **loop** function drives the rest of the user interface. It checks to see if a **button** has been tapped and, if so, calls a handler function. After this it puts the hardware into a low power state for 1 second. As the time taken to check for a button activation is very small, the system will spend most of its time in a low power state, thus prolonging battery life.

The operation of the button handler function depends on the current state of the system. There are four major states defined by the **display_state** type.

```
enum display_state {reading_state, main_menu_state, library_state, system_info_state};
```

When a page of an eBook is displayed the system is in **reading_state**. If the main menu is displayed it is in **main_menu_state**. If the library menu is displayed it is in **library_state**, and if the system information page is displayed, it is in **system_info_state**. The function of the touch pad buttons depends or the current state, so each state has its own button handler.

In **reading_state** tapping the **menu_button** causes the main menu to be displayed. Tapping the **previous_button** causes the previous page to be displayed, unless the current page number is 1. Tapping th**e next_button** causes the next page to be displayed, unless the last page in the book has been reached.

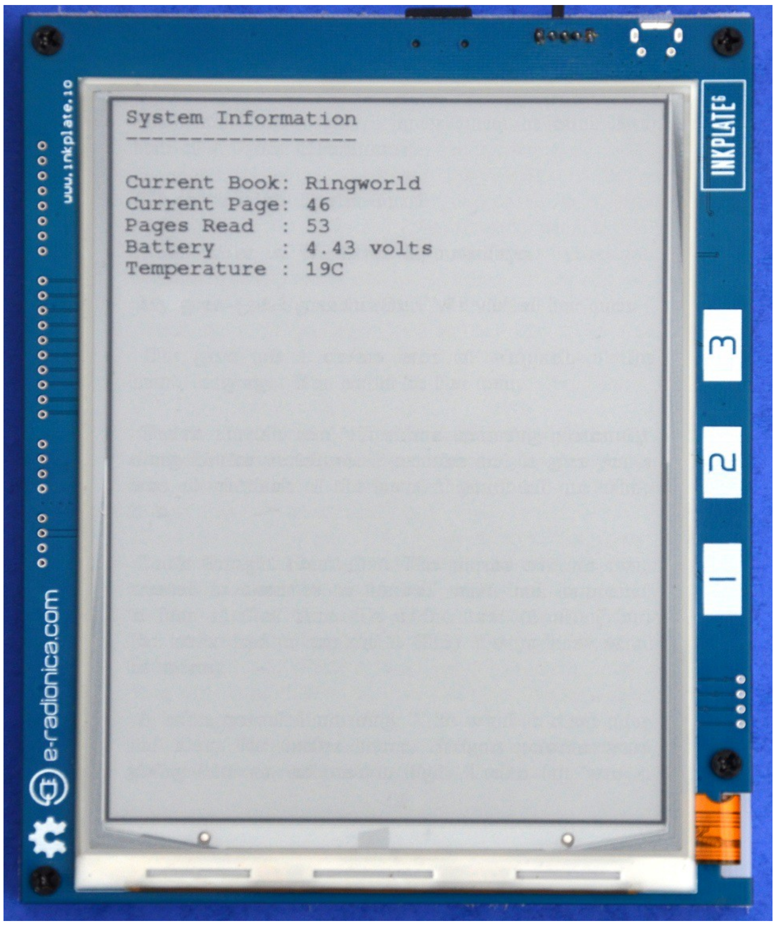
In **main_menu_state** the next and previous buttons provide menu navigation and the **menu_button** activates an option. Activating the “EXIT” option causes the system to return to r**eading_state** at the current page.

The main menu also provides access to the first page of the book from any page, and the furthest read page from any page. Selecting either of these options will return the system to **reading_state** with the appropriate page displayed.

Selecting the library option from the main menu causes the library menu to be displayed. This displays a list of books stored on the SD card in terms of their file names. The list of options is truncated if it would overflow the bottom on the display.

Selecting a book causes the system to return to **reading_mode** displaying the first page of the selected book. All the settings associated with any previous book are reset.

The final option on the reading menu allows a system information page to be displayed as shown in the following image. When this is displayed, taping any button causes the system to return to the main menu.



```
System Information
------------------

Current Book: Ringworld
Current Page: 46
Pages Read  : 53
Battery     : 4.43 volts
Temperature : 19C
```

## SD Card Organisation

The system expects the SD card to be formatted as FAT32 and to contain a directory called ”Library” at the top level. Other files and folders may exist on the card but will be ignored. The library directory should only contain files representing eBooks. The names of these files should be the titles of the books without extensions. The files names will be used directly to display the book titles in the library menu. The contents of each book file should be the text of the book divided into paragraphs by single newline characters, i.e. a linefeed, ASCII character 10 decimal.

## Converting eBooks into the Required Format

Books in formats such as EPUB, MOBI etc can be converted to simple text using the Calibre eBook management application, which available free for Windows, Linux and MacOS (other ebook managers are available). The text file produced will need a little further processing to achieve the required format, but this can be done with any capable text editor.

To convert a book using Calibre follow these steps.

1. Add the book to the Calibre library.
2. Select it and click the “Convert books” button in the menu bar.
3. When the conversion window appears, click on the “Output format” drop-down at the top right of the window and select “TXT” in the drop-down list.
4. Click the “OK” button at the bottom of the window and the conversion will begin.
5. When the conversion is complete, open the folder in which the eBook is stored by clicking “Click to open” next to the word “Path” in the right-hand column of the main Calibre window.
6. Copy the newly created text file to a convenient location for further processing.

At this stage the new text file will contain line breaks which do not end paragraphs. Paragraphs will always be ended by at least two consecutive line breaks. The single line breaks need to be replaced by single spaces so that they still separate words. Actual paragraph breaks can then be replaced by single newline characters. Using Notepad++ (other text editors are available) this is achieved as follows. In what follows it is assumed that the text file contains Windows style line endings, i.e. carriage return followed by linefeed.

Open the book file.

1. Select “Replace” in the “Search” menu.
2. Make sure that the “Wrap around” checkbox is ticked and “Extended” is selected under “Search Mode”.
3. Type “\r\n\r\n” (without the quotes - this represents a double carriage return, linefeed sequence), into the “Find what” box.
4. Type “$$$”, or some other sequence that does not occur in the text into the “Replace with” box and click the “Replace All” button.
5. Type “\r\n” into the “Find what” box.
6. Type “ ” (a single space), into the “Replace with” box and click the “Replace All” button.
7. Type “$$$” into the “Find what” box.
8. Type “\n” (a single newline), into the “Replace with” box and click the “Replace All” button.

If the file has UNIX style line endings then substitute “\n\n” for “\r\n\r\n” in step 3 and “\n” for “\r\n” in step 5. Alternatively use the “EOL Conversion” option under the “Edit” menu to force a particular line ending style before you start. Of course, another possibility is to write a simple program to do this for you. See the next section for other possible formatting changes.

At this point the file should be in the correct format and can be transferred to the library directory on the SD card to be used with the eBook reader.

## Notes

- This system was build using Arduino IDE 1.8.13 under macOS Sierra, but has also been built under Linux Mint 20. Both these system were running on a 2010 Macbook Pro.
- The e-radionica.com Inkplate library version was 2.0.0.
- A 32Gb SanDisk Ultra SD card was used because it was available, but a significantly smaller card would suffice. The largest book installed has over 1,500 pages in hardback form, and had a file size of just over 4Mb, therefore 250 such books would fit on a 1Gb card.
- eBook format conversions were performed using Calibre and Notepad++ running under Windows 7 Pro. The Geany text editor also has the required functionality to be used for formatting purposes under MacOS, Linux and Windows.
- eBooks already in text format can be obtained, free of charge, directly from the Gutenberg Project, but will still need the final stage of formatting using a text editor to be performed.
- The fonts provided with the graphics library are limited to ASCII characters. Any Unicode characters in a book file will therefore fail to display correctly. The most common problems involve speech quotes (“”) and apostrophes ('). Where they are an issue, both opening (“) and closing (”) double quote characters can be replaced by ASCII double quote characters using a text editor. Apostrophes can be replaced by the ASCII single quote character. The image at the beginning of this document shows a displayed page from a book which has been reformatted in this way. A simple reformatting program could automate a whole range of such substitutions to suit personal preferences.