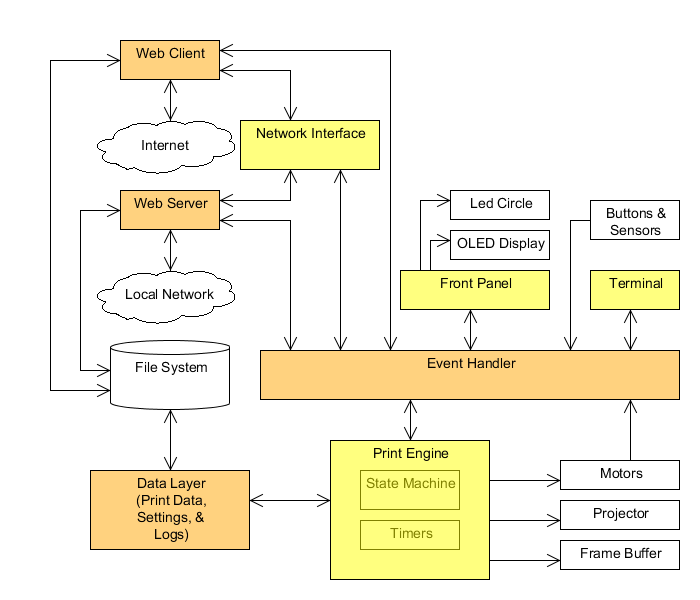
**Ember Firmware Architecture**   
Version 2.3

By Richard Greene & Jason Lefley

**Overview**   
The Ember 3D printer firmware (codenamed smith) runs on a BeagleBone Black (BBB) clone running Debian. The diagram below shows the major firmware components in color, with associated hardware in white. The arrows indicate the primary direction of data flow.   
  
  
**Event Handler**   
The event handler implements the main loop and detects interrupts from front panel buttons and the motor controller, timer expirations, and external input, e.g. from the net. It notifies via callbacks all components that have subscribed to such events, for further processing. In the case of hardware interrupts from the motor and front panel controllers (implemented in separate AVR firmware, not described in this document), the event handler reads the I2C status register of the controller that generated the interrupt (providing the reason for the interrupt) and includes that value in the callbacks. Components may also raise events, e.g. to broadcast status or send commands. Named pipes are used to send commands to the event handler, and to report status to the web client. The components shown above in yellow do not communicate directly with each other, but only through the event handler (and in some cases, the file system).   
The event handler is based on epoll, which responds efficiently to any inputs that can be monitored via file descriptors, i.e. virtually anything. It blocks, yielding all its time to the operating system, until there is actually new input available.

**Data Layer**   
The data layer provides interfaces for components that need to store and/or retrieve data. In particular it manages the storage of print data.  Data for only one print at a time may be present in the printer's file system, and a new print job may only be downloaded when no print is in progress. That enables users to decide if they would like to re-print existing data before downloading a new print job. It also avoids the possibility that data download would interfere with the timing of the printing process.

The data layer also gives all components access to a logger and a settings singleton. It is responsible for reading and writing individual settings, stored as name/value pairs in a JSON text file, and provides the ability to reset them to their default values. The logger subscribes to event handler events in order to log them. Any event may be logged, though not all events are currently logged. The logger also provides methods for any component to log errors, without depending on the event handler. Logging operations are performed using syslog.

**Print Engine**   
The print engine is responsible for controlling the printing process itself. It is driven by a hierarchical state machine (implemented with the Boost Statechart library) that responds to user input, motor interrupts, and timed events that it schedules (e.g. end of exposure of current layer). It also fires a status update event whenever its status changes. The components below subscribe to that event in order to refresh their UI. The print engine defines the actions to be taken when buttons are pressed, which depends on its current state.  
The print engine drives the frame buffer using the SDL library to display slice images on the projector when exposing, and an all-black screen at all other times. The slices are rasterized externally and downloaded to the BBB as PNG files. The print engine also turns the projector's UV LED on and off, to eliminate the stray light that would otherwise be present when showing an all-black screen.

**Front Panel**   
This component subscribes to the print engine's status update events and uses those to appropriately update the circle of LEDs and the text on the OLED dot matrix display.

**Terminal**   
For development purposes, a terminal component subscribes to status updates, for display using stdout. It also accepts commands entered from a keyboard via stdin.

**Network Interface**   
The network interface communicates with external applications that provide the remote UI for the printer. This communication is facilitated by writing printer status messages to a named pipe in response to status update events.

**Web Server**   
The printer hosts a web application and server that provide a set of HTTP endpoints accessible from the local network (Wi-Fi, USB, or Ethernet). The application is written in Ruby and leverages the Sinatra web framework. HTML endpoints exist for accessing pages to load print data, upgrade firmware, and download log archives. JSON compatible endpoints also exist to allow other applications to load print data and issue commands. To handle incoming requests, the application may send commands to the event handler and/or invoke additional functionality in the Ruby code.

**Web Client**   
An event driven web client provides communication with web applications that interact with the printer. The client is written in Ruby and uses the Faye web socket and EventMachine libraries to enable near real-time, bi-directional communication with a cloud based backend service. The client handles incoming requests, including print data loading, firmware upgrades, log retrieval, registration, and printer control, by dispatching commands to the event handler and/or by invoking additional functionality in the Ruby code. The client listens for status updates from the network interface. The client makes HTTP requests to the backend to communicate these status updates and other outgoing information.

**USB Interface**   
USB hosts can use the network over USB interface to control the printer, without the need for custom USB drivers. The event handler also responds to USB drive insertion/removal events, so that print data may be loaded directly from drives plugged into the printer.

**Process View**   
Most of the firmware runs as a single thread in a single user-mode process. The major exceptions are the webserver and web client, which each run in their own separate processes. Because the OLED display and the temperature sensor are both relatively slow, a separate thread is spawned to draw each screen and for each temperature reading. A separate thread is also used for any image processing that may be needed, e.g. to improve XY dimensional accuracy.   
The firmware's main() routine is responsible for initializing the system, and configuring each of the components. The event handler is aware of each of the possible types of events and provides a method to enable components to subscribe to specific events. The main() routine then starts the print engine state machine and the event handler, which run until smith is terminated. Components use timed events (via timerfd\_xxx functions) instead of sleep calls to minimize their own consumption of CPU cycles.

**Safety & Security**   
The projector shows a black screen and turns off the UV LED whenever the door is opened or fatal errors (those that would prevent printing) occur, to avoid exposing users to UV light.  The motors are also paused if the door is opened while they are in motion.  
Data being sent to and from the printer is not encrypted. It is not yet clear what steps, if any, may need to be taken to secure the printer's network interface.

**Localization**   
All text messages used for status and error reporting, including those displayed on the front panel, are defined as symbolic constants in a small number of header files. Therefore only those files would need to be translated (and the firmware rebuilt) in order to customize the printer for non-English-speaking locales.