The purpose of this document is to describe the logic behind the HAST Arduino code. Each function is briefly documented, and any problem points or potentially confusing logic will be expanded upon further.

**Code basics:**

1. Uploading Code: Uploading code is standard and instuctions can be found in the documentation for the Arduino IDE. However, you **MUST** disconnect the RockBLOCK from pins 0 and 1 for the upload to work. For simplicity, switches are attached to both pins that, when pressed, allow code to be uploaded. This process is time-dependent, so if testing the RockBLOCK is desired, you must press the switched during the upload and release them as soon as possible once the upload is complete.
2. Libraries: The code includes multiple open-source libraries for utilizing the SD card, reading from the sensors, and interacting with the RockBLOCK. Make sure to install these libraries before editing/uploading the code.
3. Serial Logging: Multiple serial baud rates are used for logging. 38400 is used for standard logging and is useful up until a message is sent to the RockBLOCK. 19200 is used for RockBLOCK logging and logs anything after a message is sent to the RockBLOCK.

**Function: void setup()**

Starts by initializing the SD card for storing readings. Once the SD card is initialized, data is collected from the sensors and stored on the SD card and the EEPROM is updated to reflect a new measurement. Finally, if there have been [sendInterval] measurements since data was last sent, the function initializes the RockBLOCK and attempts to send the most recent set of measurements.

Potential points of confusion: EEPROM is a small block of internal storage and is used to track the number of measurements since the last message was sent.

**Function: void clearFiles()**

Clears both the data file and the position file. Only used for testing.

Potential points of confusion: ‘0’ is written to the position file to indicate that no data has been sent.

**Function: void initializeRB()**

Initializes the RockBLOCK modem. Taken from the IridiumSBD library.

**Function: bool sendMessage(const char\* msg)**

Sends a message through the RockBLOCK modem. Returns true if message was successfully sent, false otherwise. Taken from the IridiumSBD library.

**Function: void sendData()**

Starts by getting the current position in the data file and moving to that position in the data file. Next, while we still have data to send and we haven’t failed to send a message (tracked by trySend), we create character strings with [daysPerMsg] measurements and attempt to send these measurements through the RockBLOCK. If successful, we mark [changed] to be true to signal that the position in the file needs to be changed. If unsuccessful, we stop attempting to send data. Finally, we check whether all of our data has been successfully sent. If so, we reset the number of days since the last successful message and set the position to the end of the file. If we still have data but were able to send some, we update the position file to hold the position after the last successful transmission.

Potential points of confusion: [lastPos] tracks the position of the last successful transmission in case some, but not all, of the new data is successfully sent. Also, the message string needs to be converted to a character array to fit the parameter used by sendMessage().

**Function: void collectSensorData()**

Starts by creating a measurement string that holds the current date. Then for every sensor attached to the box, it collects both a temperature (Celsius) and a humidity reading. This data is added to the measurement string. Once every sensor has taken its readings, the completed measurement is saved to the SD card.

Potential points of confusion: Newlines aren’t used when storing the measurements as the RockBLOCK treats newlines as an end of message. There may be a workaround for this, but we haven’t found one yet.

**Function: String getDate()**

Returns the current date obtained from the Rocketscream. Currently not implemented.

Current plan of implementation: Uno will query Rocketscream through a pin connection for a Unix timestamp. Once it gets the timestamp, it will convert it to a human readable timestamp (accurate to at least the nearest hour) and return that timestamp as a string.

**Function: int getPos()**

Gets the current position in the data file.

**Function: void setPos(int newPos)**

Sets the current position in the data file.

Potential points of confusion: ‘\0’ is appended to the end of the string for finding the end of the position during parsing.