Commands (rough descriptions, detailed payload bytes lower down):

* List files
  + Just the command, no other payload
* Schedule capture
  + Timestamp, name, size?
* Schedule rotate
  + Timestamp, direction (3 integers for now)
  + Need to ask if we have a format for this yet or how we should go about it
* Delete file
  + File name
* Reformat SD
  + Command only, no payload
* Update TLE values
  + For now just epoch. Do we even need anything else? Perhaps for rotation?
* Get file (CS -> GS) [This is basically just a bulk transmission from a start location]
  + File name, transmission time, start location
  + This will just be the bulk transmission, will use a different command for grabbing missing/bad frames
* Get individual frames (Both ways!) [This is used after a bulk transmission for any missing]
  + File name, list of frames
* Send file (GS -> CS) [Basically tells CS to disable interrupts, get ready to receive a file]
  + Name of file, size of file (knows how many frames), timeout
  + Sets CubeSat into receiving mode (sets flag, stores file name). When set, will read data frames received into the specified file (ignores them otherwise).
  + Should probably leave file open and write it in immediately. Frame data will be written into the file at an index that corresponds to the frame ID.
  + CRC check every frame received, add it to a list.
  + Process:
    - Gs issues command to switch to receiving mode, tells it the file name and size.
    - CS switches to receiving mode, status frame OK when ready. In this initial node, freely records data received until the EOT is issued.
    - GS begins transmitting, CS records any missing/bad frames.
    - GS issues end of transmission. CS proceeds to next part in function.
    - If all frames received fine, CS issues status frame OK. Else issues get\_individual\_frames command to GS. CS still in receiving mode.
    - GS begins transmitting missing frames. CS continues to write it into a file, and record any bad ones.
    - Again, GS issues EOT request command. CS replies status OK, or again commands for missing frames, and repeats if necessary.
* Request transmission end [Basically tells CS to reply OK, or send command for individual frames]
  + Tells the Cubesat we’re done transmitting. Just a command, no other payload. May be repeated by GS until response.
* How about another command for getting a readout of CubeSat status (not from an SD file)– e.g. get\_status
  + This will gather some sensor data, power level, voltages, SD mount status, and anything else important. Can use in the case of the SD card not working or if we just want a general status readout? No parameters?
* Reboot
  + Causes a system reset, e.g. we want to apply the software patch

**In general for file transmission:**

* CubeSat to Ground-station (telemetry, images):
  + GS issues get\_file with parameters (file name, timeout, start location in file). CS Acks, transmits that data until file finished or timeout. Status frame when done.
  + GS was recording if any missed frames, issues the get\_individual\_frames if so. Repeats if necessary.
  + Repeats over several passes if necessary; we know where the last stopped, can increment the start location in get\_file command.
* Ground-station to CubeSat (software patches):
  + GS issues command (send file) for CS to listen for data (sets receiving flag, preps new file on SD). CS Acks.
  + GS sends all its data, CS records bad/missing frames, then GS requests for transmission to end (EOT).
  + If CS is missing any frames, issues a get\_individual\_frames command to the GS.
  + GS transmits those frames, and again issues an EOT.
  + Once all frames received, CS sends an OK status frame, and transmission ends.

These files are small enough that they should be able to be transmitted in a single pass easily, so don’t need to make GS->CS work over multiple passes. If it fails to do so in one pass for some reason, just issue a command to delete the file then try again.

**Command payload parameters**

Commands in general will use the first byte to indicate the command, however the rest of the payload is used for parameters and is unique to each command.

**Schedule image capture**

* Command (1 Byte uint8\_t), Timestamp (4 Bytes uint32\_t), File name (11 Bytes char) [8.3 format]
* 11 bytes or 8 bytes for the file name? We can auto append ‘.jpg’ in the code. Maybe leave space for 11 and just ignore the last few, or go 8 and the GS can make sure its correct before sending… Ok, 11 to be consistent? Extension really doesn’t matter anyways as long as we know what is in the file.

**Schedule rotate**

* Command (1 Byte uint8\_t), Timestamp (4 Bytes uint32\_t), Direction (3x 1 Byte uint8\_t)

**Update TLE**

* Command (1 Byte uint8\_t), Epoch (4 Bytes uint32\_t), Nothing else yet

**List files**

* Command (1Byte uint8\_t)

**Delete file**

* Command (1 Byte uint8\_t), File name (11 Bytes char) [FAT16 8.3 format file name]

**Reformat SD**

* Command (1 Byte uint8\_t)

**Get file**

* Command (1 Byte uint8\_t), File name (11 Bytes char), Start location (3 Bytes uint24\_t), Timeout (2 Bytes uint16\_t)
* Timeout must be a positive value, and impose a maximum limit (e.g. max pass length)

**Get frames**

* Command (1 Byte uint8\_t), File name (11 Bytes char), List of frame IDs (n\* 3 Bytes uint24\_t)
* List of n frame with IDs that are 3 bytes each. We can store each as a uint32\_t, won’t take up too much memory? We do have 192Kb of ram but I still hate dynamic allocation. Perhaps can declare a big list to hold these beforehand but what happens if we’re missing a lot of frame s? Although it’s probably the only decent solution anyways. Just make its size 2-3x the expected amount.
* If a value encountered in the list is max value for a 3 byte int (all bits 1), then assume that’s the end of the list. Perhaps, either don’t even extend the payload that far and just end it at the end of the list, or set the rest of the bits in the payload to 1. Honestly probably easier to just make the frame shorter and don’t even deal with the extra case if we don’t need to. Make it a GS problem.
* May need to issue several of this command in sequence if there is a lot of frames missing. With a 92 byte max payload, 12 bytes for command and file name parameters, that leaves 80 bytes for the list of IDs. If each ID is 3 bytes, we can fit 26 unique IDs per get\_frames command.
* If we can squeeze a single byte out of somewhere, we have room for 27 unique IDs per command.

**Send file**

* Command (1 Byte uint8\_t), File name (11 Bytes char), File size in bytes/max ID (4 Bytes uint32\_t)

**Get status**

* Command (1 Byte uint8\_t)

**Reboot**

* Command (1 Byte uint8\_t)