2026-2027

- Mechanical
  - CAD Guide
- Electrical
  - Electrical Requirements
  - Electrical Architecture
  - Serial Specification
  - Firmware - StormLib
- Software
  - Software Requirements

Mechanical

Mechanical

CAD Guide

The STORM CAD is laid out in a specific way.

	Fly wheel Prototype Main	1:48 PM Today	Elizabeth Knight
	Intake Prototyping Main	10:46 AM Today	me
	Size Compare Main	10:43 AM Today	me
	Mounts Main	10:13 AM Today	me
	Test Chassis Main	6:36 PM Sep 25	me
	Summer Design Main	6:07 PM Sep 25	Elizabeth Knight
	Brackets & Gussets Main	12:50 PM Sep 14	me
	Shaft, Collars, Hubs, ... Main	12:39 PM Jul 3	me
	Simplified Strafer Cha... Main	12:15 PM Jul 3	me
	imported strafer chas... Main	12:06 PM Jul 3	me
	Tubing & Extrusion Main	11:34 AM Jul 3	me

Mounts

Is for mounts that attach things to other things. Things like motor mounts, bearing mounts, camera mounts, etc. *Most of the parts you design will probably go in here.*

Brackets & Gussets

Contains brackets and gussets. I'm not actually sure what the real difference is, but for our purposes, gussets are flat and attach things on the same plane, while brackets are not flat and attach opposing planes together.

Shaft, Collars, Hubs, & Bearings

This will be mostly CAD imports, and contains hex shaft, shaft collars (which keep things on a shaft), hubs (which mount things to a shaft / transfer rotational force), and bearings (which hold shaft and allow it to rotate).

If you have a CAD import of a hub or shaft collar or something like that put it here!

Tubing & Extrusion

This pretty much only holds the 1 piece of extrusion we will be using, the 1 in. square hollow tube aluminum extrusion. You shouldn't have to edit this document directly, but can import a custom length of extrusion.

Simplified Strafer Chassis

Contains some stripped-down goBILDA parts so the documents don't take forever to load. Notably, includes a goBILDA YellowJacket motor.

Might need to make some more documents for motors, servos, gears, etc.

Electrical

Electrical Requirements

Electrical Architecture

Electrical

Serial Specification

Electrical

Firmware - StormLib

What is StormLib?

To make it easier to send and receive the messages defined in the STORM 2027 Serial Specification , we created StormLib. StormLib also includes some abstractions of hardware (motors, encoders, etc.) taken from RobotLib. StormLib is subject to change once the 2027 rules release, and in future version afterwards.

Getting Started

1. Install the Arduino IDE
2. Install Earl's Arduino Pico library from Arduino IDE's Library Manager
3. Install StormLib (TODO)
4. Copy the firmware template (TODO) and change it to only care about the messages you care about (the ones your board will receive and send)

How to Add / Edit a Message

To add your own custom message, or edit/update an existing message, you will need to make changes in several locations. First, ensure your message is added/updated in the Serial Specification before writing any code. Then follow these steps:

1. Add/update the message ID in the `Message` enum
2. Below that, add/update the typedef struct for the message. Example:

```
// ID 21
typedef struct DriveVelocityCmd {
    int16_t front_left_velocity;
    int16_t front_right_velocity;
    int16_t back_left_velocity;
    int16_t back_right_velocity;
} DriveVelocityCmd;
```

Note: if you message is a command (the computer telling a PCB to do something), then

add the "Cmd" suffix to the name. If it's a response (the PCB telling the computer something it asked for), then add the "Rsp" suffix to the name. Otherwise, add the "Msg" suffix to the name.

3. Below that, in the `StormMessage` class, make a function that returns the struct you defined in the previous step. Example:

```
TaskResponseRsp AsTaskResponse() {
    TaskResponseRsp rsp;

    rsp.intake_velocity = byte_to_int16(DATA);
    rsp.arm_position = static_cast<uint8_t>(DATA[2]);
    rsp.climber_position = byte_to_uint16(DATA + 3);
    rsp.charging_wheel_velocity = static_cast<uint8_t>(DATA[5]);

    return rsp;
}
```

   Note: `DATA` is the `char` (byte) array that stores the data portion of the message. You will need to cast those bytes into the data type of the variables your struct defines. This can be done using one of the functions already defined (like `byte_to_int16`) or using `static_cast<T>()`. Also note the indexing: DATA is stored as a pointer (because that's how C arrays work), so to access subsequent bytes, just add to it, as shown in the example above.

And you're done! Now you can work with the struct you defined to do things.

```
// if there is a message addressed to us
   if (bus.HasMessage()) {
       // get the message
       msg = bus.GetMessage();

       // and do something diffrent based on what message it is
       switch (msg.type) {
           case DriveVelocity: {
               auto converted = msg.AsDriveVelocity();

               //TODO

               break;
           }
           default:
```

```
                // message doesn't concern us, don't do anything
                break;
        }
    }
```

Software

Software Requirements

TODO: insert link