```
#include <Adafruit_PWMServoDriver.h>
#include <LiquidCrystal.h>

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
//String lastReceivedMessage = "";
bool handShakeSuccessful = false;
int received;
int startStage = 0;
bool started = false; //once started == true then start swapping from the serial reads
bool CommandInProcess = false;
String inputString = "";
bool stringComplete = false;
byte  insertNumber = "";

const int AddOnType_MMU = 0;
const int AddOnType_BlendBox = 1;
int AddOnType = AddOnType_MMU;

const int servo_pwm_max = 2900;// 2376;
const int servo_pwm_min = 600; //484;


//servos enums
const int eePinNum = 0;
const int eeMaxAngle = 1;
const int eeCurrentAngle = 2;

//Servos
const int numOfServos = 8;
const int s_Tool_Rotate = 0; //360d //TR
const int s_Tool_Height = 1; //TH
const uint8_t s_Tool_Lock = 2; //TL
const uint8_t s_QuickSwapHotend_Lock = 3; //QL
const uint8_t s_ToolHolder_Rotate = 4; //360d //HR
const uint8_t s_Cutter_Rotate = 5; //CR
const uint8_t s_Cutter_Action = 6; //CA
const uint8_t s_WasteCup_Action = 7; //WA

int servos[numOfServos][3]; //pin #, max angle, current angle
String servos_names[numOfServos] = {"Tool_Rotate", "Tool_UpDown", "Tool_LockUnlock", 
"Cutter_Rotate", "Cutter_CutOpen", "WasteCup_DumpFill", "ToolHolder_Rotate", 
"ExtruderHotend_LockUnlock"};

//LCD
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
unsigned long tepTimer = 0;
bool setupComplete = false;
int buttonPress;                                   // variable to store the value coming 
from the analog pin
int potValue = 0;
int potMinValue = 600; //15
int potMaxValue = 1000;
int potRange = 400; //1008
int inServoTuning = false;
int potCentered = false;
int toolDegrees = 0;
int pulselength = 0;
int currentPotPosition = 0;
int currentAngle = 0;
int moveToAngle = 0;
bool servoSetModeEnabled = false;
bool buttonPressed = false;
float anglePercentOfMax = 0.0;
int potMatchValue = 0;
int maxAngle = 180.0; //360.0
int currentServoEditing = 0;
bool loopStarted = false;

//swap tools process
```

```
const int eeps_ServoNumber = 0;
const int eeps_Degrees = 1;
const int eeps_MsDelayPerDegreeMoved = 2;
const int eeps_MsDelayAfterCommandSent = 3;
const int eeps_StepType = 4; //eeps_ButtonCheck = 4;
const int eeps_IsIncludedInMMU = 5;

const int eeRegularStep = 0; //eeButtonCheck_No = 0;
const int eeButtonCheck_Empty = 1;
const int eeButtonCheck_HoldingTool = 2;
const int eeExtrude = 3;
const int eeRetract = 4;
const int eeToolHolderPrepRotate = 5;
const int eeAddHalfDegreePrecision = 6;
const int eeToolHolderPrepUNrotate = 7;

const int eeToolHolderPrepRotate_Degrees = 3;
const int eeToolHolderPrepUNrotate_Degrees = 3; //4; //1; //2; //3; //3; //1; //3; //6; //3;


const int numOfProcessSteps_LoadTool = 21; //***change this if adding or removing process steps
const int numOfProcessSteps_UnloadTool = 30; //***change this if adding or removing process steps
bool executeNextProcessStep = false;
int currentStepOfProcess = 0;
int ProcessSteps_LoadTool[numOfProcessSteps_LoadTool][5]; //servo number, degrees, msDelayPerDegree, msDelayAfterCommandSent, buttonCheck
int ProcessSteps_UnloadTool[numOfProcessSteps_UnloadTool][6]; //servo number, degrees, msDelayPerDegree, msDelayAfterCommandSent, buttonCheck, IsIncludedInMMU
int ps_currentServo = 0;
int ps_targetAngle = 0;
int ps_msDelayPerDegreeMoved = 0;
int ps_msDelayAfterCommandSent = 0;
int numberOfStepsToProcess = 0;
int msDelayAfterCommandSent_Buffer = 100; //50 //in milliseconds //extra ms delay






//menu
//menus enums
const int eeOnce = 0;
const int eeEndless = 1;
const int eeLoadTool = 0;
const int eeUnloadTool = 1;
const int eeAutomaticProcess = 0;
const int eeManualProcess = 1;

const int mm_menu_OnceEndless = 0;
const int mm_menu_LoadUnload = 1;
const int mm_menu_AutoManual = 2;
const int mm_menu_Process = 3;

bool automaticExecuteProcessSteps = false;
int menu_currentLevel = 0;
int menu_OnceEndless_position = 0; //for the Once/Endless menu
int menu_LoadUnload_position = 0; //for the load/unload menu
int menu_AutoManual_position = 0; //for the auto/manual menu
const int menu_OnceEndless_TotalPositions = 2;
const int menu_LoadUnload_TotalPositions = 2;
const int menu_AutoManual_TotatlPositions = 2;
String menu_OnceEndless_Name[menu_OnceEndless_TotalPositions] = {"Once", "Endless"};
String menu_LoadUnload_Names[menu_LoadUnload_TotalPositions] = {"Load", "Unload"};
String menu_AutoManual_Names[menu_AutoManual_TotatlPositions] = {"Auto", "Manual"};
bool Selected_LoadUnload = false;
bool Selected_Load = false;
bool Selected_Unload = false;


//error state button press
const bool ErrorCheckingEnabed = false;
const int CheckButton_Pin = 3; //0; //digital pin zero(0)
const int eeePauseAtRotation = 95;
bool InErrorState = false;
int CurrentProcessType = eeLoadTool;//load/unload
int ErrorOnProcessStep = 0;

//tool holder rotation and selection
bool firstPositionCommandGiven = false;
const int servoMinAngle = 0;
float pos_Tool_Holder_FirstTool = 15; //11.7; //11; //12; //2;
bool toolIsLoaded = false;
int CurrentTool = 0;
int eeth_maxTool = 24;

int pos_Tool_Lock_Locked;
int pos_Cutter_Rotate_Stowed;
bool LockToolPartWayThru = false;
int numMsUntilLock = 50; //100; //200; //10ms per degree currently

void printWithParity(String message) {
  Serial.println(message + String(checkParity(message)));
}

void updateLCD(String message1, String message2) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(message1);
  lcd.setCursor(0, 1);
  lcd.print(message2);
}

void updateLCD_line1(String message) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(message);
}

void updateLCD_line2(String message) {
  lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print(message);
}


//**** Holder Rotate (HR) ****
//Servo 4
int Adjustment_HolderRotate = 0;

void ToolHolder_AlignToThisTool(int SelectThisTool)
{
int localPulseLength = 0;
int msDelayPerToolPostionToCompleteMovement = 50;
int msDelayPadding = 50;
int msDelayUntilRotationComplete = 0; //total ms to delay for the current tool holder rotation
float degreesPerTool = 14.4; //14.72; //computed angle is 14.4d per spline calced as 25T splines 360/25=14.4 //14.72; //last 14.80 //this works for 25T's but barely: 14.85; //19:14.95 slightly too much;
float degreesPositionOfSelectedTool = (float)SelectThisTool * degreesPerTool;

//apply adjustment from EEPROM
   pos_Tool_Holder_FirstTool = pos_Tool_Holder_FirstTool + Adjustment_HolderRotate;

```

```
    servos[s_ToolHolder_Rotate][eeCurrentAngle] = degreesPositionOfSelectedTool + pos_Tool_Holder_FirstTool;

    localPulseLength = fMap(degreesPositionOfSelectedTool + pos_Tool_Holder_FirstTool, servoMinAngle, servos[s_ToolHolder_Rotate][eeMaxAngle], servo_pwm_min, servo_pwm_max);

    pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, localPulseLength);

    msDelayUntilRotationComplete = abs(CurrentTool - SelectThisTool) * msDelayPerToolPostionToCompleteMovement + msDelayPadding;

    delay(msDelayUntilRotationComplete);

    CurrentTool = SelectThisTool;
}

int fMap(float desiredAngle, int MinAngle, int MaxAngle, int minPWM, int maxPWM)
{
  int angleRange = MaxAngle - MinAngle;
  int pwmRange = maxPWM - minPWM;

  float desiredAnglePercentOfRange = desiredAngle / angleRange;
  float pwmByDesiredAngle = float(pwmRange) * float(desiredAnglePercentOfRange) + float(minPWM);

  return pwmByDesiredAngle;
}

void setup()
{
    //position variables
    //Servo 0
    //**** Tool Rotate (TR) ****
    int Adjustment_Tool_Rotate = 0; //1; //-1; //0; //1; //2; //changed out servo all should be off by same angle over last servo
    //next line is starting first 1st position
    int pos_Tool_Rotate_ButtingTheToolToTheLeftOfNext = 104 + Adjustment_Tool_Rotate; //103 //102; //104; //103;
    int pos_Tool_Rotate_LeftOfToolInHolder = 98 + Adjustment_Tool_Rotate; //97 //98; //99; //101;//101 to try and deal with the single nozzle load failure //100; //102; //101;//103;//95; //SetupMode
    int pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar = 96 + Adjustment_Tool_Rotate; //95 //80, 85; //90; //83;//76 = 11
    int pos_Tool_Rotate_UnderToolHolder_CenteredUnderCurrentTool = 98 + Adjustment_Tool_Rotate; //97 //98, 97; //95;
    int pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar_NoPressure = pos_Tool_Rotate_UnderToolHolder_CenteredUnderCurrentTool; //95; //97; //98; //96; //91; //86 = 5
    int pos_Tool_Rotate_ReleaseFromHotendUnderToolHolder = 109 + Adjustment_Tool_Rotate; //108 //106 //104;
    int pos_Tool_Rotate_BetweenBothNozzles = pos_Tool_Rotate_ReleaseFromHotendUnderToolHolder - 7 + Adjustment_Tool_Rotate;
    int pos_Tool_Rotate_ButtonToolCheck = 75 + Adjustment_Tool_Rotate; //74, 72, 75, 70 //72; //74; //75; //68;
    int pos_Tool_Rotate_UnderExtruder_JerkConnectWithNozzleCollar = 281 + Adjustment_Tool_Rotate; //280 //275, 282; //283; //284; //265; //270; //274;
    int pos_Tool_Rotate_UnderExtruder_ConnectWithNozzleCollar = 286 + Adjustment_Tool_Rotate; //285 //284; 286; //Why did this change???!?!??? 285; //283; //284; //283; // 282; //283; //284; //285; //283; //287; //285; //278;
    int pos_Tool_Rotate_UnderExtruder_JerkReleaseFromNozzleCollar = 293 + Adjustment_Tool_Rotate; //292 //293 291; // 310; //305; //297;
    int pos_Tool_Rotate_UnderExtruder_ReleasedFromNozzleCollar = 291 + Adjustment_Tool_Rotate; //290 //291, 293 291; //285;
    int pos_Tool_Rotate_WaitingForUnloadCommand = 148 + Adjustment_Tool_Rotate; //147 //140;
    int pos_Tool_Rotate_PastWasteCup = 259 + Adjustment_Tool_Rotate; //258 //251;

    //**** Tool Height (TH) ****
```

```
        //Servo 1
        int Adjustment_Tool_Height = 0; //4; //14;//1 //-1; increasing this will lower all up/down moves. decreasing this will raise all up/down moves
        //next line is starting first 1st position
        int pos_Tool_Height_LowestLevel = 129 + Adjustment_Tool_Height; //125 //121 //Servo change 126; // below 126 it causes the servo to stall.  127; //126; //119; //117
        int pos_Tool_Height_ButtingTheToolToTheLeftOfNext = 44 + Adjustment_Tool_Height; //40 //37, Servo change 42;//we want to be at the height of the hex on the right nozzle //39; //40; //41;
        int pos_Tool_Height_NozzleCollarLevel = 37 + Adjustment_Tool_Height; //33 //30, Servo change 35; //38; //36; //29;
        int pos_Tool_Height_ToolLoweredButStillInHolder = 59 + Adjustment_Tool_Height; //54 //Servo change 49; //42;
        int pos_Tool_Height_ToolFullyInsertedInHolder = 31 + Adjustment_Tool_Height; //27 //25, 22, Servo change 27; //29; //20; //13;
        int pos_Tool_Height_ToolFullyInsertedInHolder_NoPressure = 38 + Adjustment_Tool_Height; //34 //35, 32, Servo change 37; //36; //40; //29; //SetupMode
        int pos_Tool_Height_ToolLoweredButStillInExtruder = 52 + Adjustment_Tool_Height; //48 //Servo change 53; //53 moves up until the hotend tapper is past the edge of the inner bore of the heater block//56; //49;
        int pos_Tool_Height_ToolFullyInsertedIntoExtruder = 41 + Adjustment_Tool_Height; //37 //39, 40, 39, 33, Servo change 38; //40; //42; //35; //28;
        int pos_Tool_Height_ToolFullyInsertedIntoExtruder_ScrappingHotendMildPressure = 44 + Adjustment_Tool_Height; //40 //Servo change 43; //42; //41; //42; //40; //42; //clunking? maybe 41?
        int pos_Tool_Height_ToolFullyInsertedIntoExtruder_NoPressure = 42 + Adjustment_Tool_Height; //38 //39; 38; //Servo change 43; //44; //42; // 45; // 42; //35;
        int pos_Tool_Height_ToolLowered_CuttingHeight = 122 + Adjustment_Tool_Height; //118 //117 //125 //113;//111, 108, 109; 107; //higher number moves down away from cutter leaving longer filament strand sticking out. //108; //Servo change 112;//works with new cutting sequence //110; //108; at 108 there was a single instance of cutting the heatbreak copper and it was ruined. //107; //99;
        int pos_Tool_Height_ToolLowered_BelowCutterJaws = 116 + Adjustment_Tool_Height; //112 //Servo change 117; //110;

        //**** Tool Lock (TL) (micro 280d servo) ****
        //Servo 2
        int Adjustment_Tool_Lock = 0;
        //next line is starting first 1st position
        int pos_Tool_Lock_Unlocked = 195 + Adjustment_Tool_Lock; //180
        pos_Tool_Lock_Locked = 112 + Adjustment_Tool_Lock; //8 //7; //8; //standard move172degrees. 8;//now precision move +.5 //9;//8;//9; //8; //9; 13;

        //**** QuickSwap- Hotend Lock (QL) ****
        //Servo 3
        int Adjustment_QuickSwapHotend_Lock = 0;
        //next line is 1st starting first 1st position
        int pos_QuickSwapHotend_Lock_Locked = 70 + Adjustment_QuickSwapHotend_Lock; //0
        int pos_QuickSwapHotend_Lock_Unlocked = 104 + Adjustment_QuickSwapHotend_Lock; //34 //32; //33; //34; //35; //29;

        //**** Cutter Rotate (CR) ****
        //Servo 5
        int Adjustment_Cutter_Rotate = 0;
        //next line is starting first 1st position
        pos_Cutter_Rotate_Stowed = 27 + Adjustment_Cutter_Rotate; //25;//1;//must be greater than 0. 0 causes major jittering. Something about 25 works better than 26. 26 had lots of jitter.
        int pos_Cutter_Rotate_Cutting = 123 + Adjustment_Cutter_Rotate; //122; //125; //126, 124, 122; //121; //122; //121; //122; //120; //124; //126; //127; //128;//get closer but lower the cutting height a little //126; //127; 127 is too close //126; //or maybe 127? //121; //99;

        //**** Cutter Action (CA) ****
        //Servo 6
        int Adjustment_Cutter_Action = 0;
        //next line is starting first 1st position
        int pos_Cutter_Action_Open = 175 + Adjustment_Cutter_Action; //160
```

```
    int pos_Cutter_Action_Cut = 15 + Adjustment_Cutter_Action; //0 //20; //40; //6; //7;
    //21;

    //**** Waste Cup Action (WA) (micro 280d servo) ****
    //Servo 7
    int Adjustment_WasteCup_Action = 0;
    //next line is starting first 1st position
    int pos_WasteCup_Action_Fill = 110 + Adjustment_WasteCup_Action; //0
    int pos_WasteCup_Action_Dump = 99 + Adjustment_WasteCup_Action; //107




    Serial.begin(9600);
    // Serial.println("Swapper3D Start!");


    //this ensures that the serial buffer is empty
    //so that no aberant tool changes are performed
    ClearSerialBuffer();


    lcd.begin(16, 2);
    lcd.setCursor(0,0);
    lcd.print("Ready to Swap!");
    lcd.setCursor(0,1);
    lcd.print("Empty");

    pwm.begin();
    pwm.setOscillatorFrequency(27000000);
    pwm.setPWMFreq(300);  // Digtal servos run at 300Hz updates


    //initialize the pin for the end effector insert full/empty checks
    // pinMode(CheckButton_Pin, INPUT_PULLUP);
    pinMode(A3, INPUT_PULLUP);

    delay(10);


    //pin #, max angle, start angle, current angle
    servos[s_Tool_Rotate][eePinNum] = 15;
    servos[s_Tool_Rotate][eeMaxAngle] = 360;
    servos[s_Tool_Rotate][eeCurrentAngle] = pos_Tool_Rotate_ButtingTheToolToTheLeftOfNext
    ;
    servos[s_Tool_Height][eePinNum] = 14; //using this pwm servo port on the servo
    shield causes random bytes on the serial lines
    servos[s_Tool_Height][eeMaxAngle] = 180;
    servos[s_Tool_Height][eeCurrentAngle] = pos_Tool_Height_LowestLevel;
    servos[s_Tool_Lock][eePinNum] = 13;
    servos[s_Tool_Lock][eeMaxAngle] = 280;//micro
    servos[s_Tool_Lock][eeCurrentAngle] = pos_Tool_Lock_Unlocked;
    servos[s_QuickSwapHotend_Lock][eePinNum] = 12;//s_QuickSwapHotend_Lock
    servos[s_QuickSwapHotend_Lock][eeMaxAngle] = 180;//
    servos[s_QuickSwapHotend_Lock][eeCurrentAngle] = pos_QuickSwapHotend_Lock_Locked;//
    servos[s_ToolHolder_Rotate][eePinNum] = 11;//s_ToolHolder_Rotate
    servos[s_ToolHolder_Rotate][eeMaxAngle] = 360;//
    servos[s_ToolHolder_Rotate][eeCurrentAngle] = pos_Tool_Holder_FirstTool;//
    servos[s_Cutter_Rotate][eePinNum] = 10;//s_Cutter_Rotate
    servos[s_Cutter_Rotate][eeMaxAngle] = 180;//
    servos[s_Cutter_Rotate][eeCurrentAngle] = pos_Cutter_Rotate_Stowed;//
    servos[s_Cutter_Action][eePinNum] = 9;//s_Cutter_Action
    servos[s_Cutter_Action][eeMaxAngle] = 180;//
    servos[s_Cutter_Action][eeCurrentAngle] = pos_Cutter_Action_Open;//
    servos[s_WasteCup_Action][eePinNum] = 8;//s_WasteCup_Action
    servos[s_WasteCup_Action][eeMaxAngle] = 280;//micro
    servos[s_WasteCup_Action][eeCurrentAngle] = pos_WasteCup_Action_Fill;//




    for(int i; i < 8; i++)
    {
        pulselength = map(servos[i][eeCurrentAngle], 0, servos[i][eeMaxAngle], servo_pwm_min, servo_pwm_max);
        pwm.setPWM(servos[i][eePinNum], 0, pulselength);
        delay(100);
    }

    //align to the first tool
    ToolHolder_AlignToThisTool(0);


    //store process steps
    ProcessSteps_LoadTool[0][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[1][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[2][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[3][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[4][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[5][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[6][eeps_ServoNumber] = s_Tool_Lock;
    ProcessSteps_LoadTool[7][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[8][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[9][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[10][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
    ProcessSteps_LoadTool[11][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[12][eeps_ServoNumber] = s_Tool_Lock;
    ProcessSteps_LoadTool[13][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[14][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
    ProcessSteps_LoadTool[15][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[16][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[17][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[18][eeps_ServoNumber] = s_Tool_Height;
    ProcessSteps_LoadTool[19][eeps_ServoNumber] = s_Tool_Rotate;
    ProcessSteps_LoadTool[20][eeps_ServoNumber] = s_Tool_Rotate;

    ProcessSteps_LoadTool[0][eeps_Degrees] = pos_Tool_Height_ButtingTheToolToTheLeftOfNext;
    ProcessSteps_LoadTool[1][eeps_Degrees] = pos_Tool_Rotate_LeftOfToolInHolder;
    ProcessSteps_LoadTool[2][eeps_Degrees] = pos_Tool_Height_NozzleCollarLevel;
    ProcessSteps_LoadTool[3][eeps_Degrees] = pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar;
    ProcessSteps_LoadTool[4][eeps_Degrees] = pos_Tool_Rotate_UnderToolHolder_ConnectWithNozzleCollar_NoPressure;
    ProcessSteps_LoadTool[5][eeps_Degrees] = pos_Tool_Height_ToolLoweredButStillInHolder;
    ProcessSteps_LoadTool[6][eeps_Degrees] = pos_Tool_Lock_Locked;
    ProcessSteps_LoadTool[7][eeps_Degrees] = pos_Tool_Height_LowestLevel;
    ProcessSteps_LoadTool[8][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
    ProcessSteps_LoadTool[9][eeps_Degrees] = pos_Tool_Rotate_UnderExtruder_ConnectWithNozzleCollar;
    ProcessSteps_LoadTool[10][eeps_Degrees] = pos_QuickSwapHotend_Lock_Unlocked;
    ProcessSteps_LoadTool[11][eeps_Degrees] = pos_Tool_Height_ToolLoweredButStillInExtruder;
    ProcessSteps_LoadTool[12][eeps_Degrees] = pos_Tool_Lock_Unlocked;
    ProcessSteps_LoadTool[13][eeps_Degrees] = pos_Tool_Height_ToolFullyInsertedIntoExtruder;
    ProcessSteps_LoadTool[14][eeps_Degrees] = pos_QuickSwapHotend_Lock_Locked;
    ProcessSteps_LoadTool[15][eeps_Degrees] = pos_Tool_Height_ToolFullyInsertedIntoExtruder_NoPressure;
    ProcessSteps_LoadTool[16][eeps_Degrees] = pos_Tool_Rotate_UnderExtruder_JerkReleaseFromNozzleCollar;
    ProcessSteps_LoadTool[17][eeps_Degrees] = pos_Tool_Rotate_UnderExtruder_ReleasedFromNozzleCollar;
    ProcessSteps_LoadTool[18][eeps_Degrees] = pos_Tool_Height_LowestLevel;
    ProcessSteps_LoadTool[19][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
    ProcessSteps_LoadTool[20][eeps_Degrees] = pos_Tool_Rotate_WaitingForUnloadCommand;


```

```
        ProcessSteps_LoadTool[0][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[1][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[2][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[3][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[4][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[5][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[6][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[7][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[8][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[9][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[10][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[11][eeps_MsDelayPerDegreeMoved] = 20; //60;//6;
        ProcessSteps_LoadTool[12][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[13][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[14][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[15][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[16][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[17][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[18][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[19][eeps_MsDelayPerDegreeMoved] = 0;
        ProcessSteps_LoadTool[20][eeps_MsDelayPerDegreeMoved] = 0;

        ProcessSteps_LoadTool[0][eeps_MsDelayAfterCommandSent] = 450; //160;
        ProcessSteps_LoadTool[1][eeps_MsDelayAfterCommandSent] = 250; //150; //160;
        ProcessSteps_LoadTool[2][eeps_MsDelayAfterCommandSent] = 250; //150; //160;
        ProcessSteps_LoadTool[3][eeps_MsDelayAfterCommandSent] = 350; //150; //80;
        ProcessSteps_LoadTool[4][eeps_MsDelayAfterCommandSent] = 100;
        ProcessSteps_LoadTool[5][eeps_MsDelayAfterCommandSent] = 150;
        ProcessSteps_LoadTool[6][eeps_MsDelayAfterCommandSent] = 30; //130;
        ProcessSteps_LoadTool[7][eeps_MsDelayAfterCommandSent] = 120;
        ProcessSteps_LoadTool[8][eeps_MsDelayAfterCommandSent] = 300; //200; //90;
        ProcessSteps_LoadTool[9][eeps_MsDelayAfterCommandSent] = 500; //260;//rotate to
        under extruder
        ProcessSteps_LoadTool[10][eeps_MsDelayAfterCommandSent] = 300; //110;//give the
        hotend time to stabilize before moving up to heaterblock bore
        ProcessSteps_LoadTool[11][eeps_MsDelayAfterCommandSent] = 300; //220;
        ProcessSteps_LoadTool[12][eeps_MsDelayAfterCommandSent] = 130;//cannot unlock
        without hitting the cooling shround. Must have delay. //30; //130;
        ProcessSteps_LoadTool[13][eeps_MsDelayAfterCommandSent] = 230; //130; //fully
        inserted into extruder
        ProcessSteps_LoadTool[14][eeps_MsDelayAfterCommandSent] = 300; //200; //110; //Lock
        the hotend into extruder
        ProcessSteps_LoadTool[15][eeps_MsDelayAfterCommandSent] = 90;
        ProcessSteps_LoadTool[16][eeps_MsDelayAfterCommandSent] = 200; //110; //jerk nozzle
        release from hotend collar
        ProcessSteps_LoadTool[17][eeps_MsDelayAfterCommandSent] = 90;
        ProcessSteps_LoadTool[18][eeps_MsDelayAfterCommandSent] = 180;
        ProcessSteps_LoadTool[19][eeps_MsDelayAfterCommandSent] = 1000; //700;
        ProcessSteps_LoadTool[20][eeps_MsDelayAfterCommandSent] = 160;

        ProcessSteps_LoadTool[0][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[1][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[2][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[3][eeps_StepType] = eeToolHolderPrepUNrotate; //eeRegularStep;
        //rotate connect with nozzle collar
        ProcessSteps_LoadTool[4][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[5][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[6][eeps_StepType] = eeRegularStep; //eeAddHalfDegreePrecision;
        //eeRegularStep;
        ProcessSteps_LoadTool[7][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[8][eeps_StepType] = eeButtonCheck_HoldingTool;
        ProcessSteps_LoadTool[9][eeps_StepType] = eeRegularStep; //eeAddHalfDegreePrecision;
        //eeRegularStep; //rotate centered under heater block bore
        ProcessSteps_LoadTool[10][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[11][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[12][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[13][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[14][eeps_StepType] = eeRegularStep;
        ProcessSteps_LoadTool[15][eeps_StepType] = eeRegularStep;
```

```
ProcessSteps_LoadTool[16][eeps_StepType] = eeRegularStep;
ProcessSteps_LoadTool[17][eeps_StepType] = eeRegularStep;
ProcessSteps_LoadTool[18][eeps_StepType] = eeRegularStep;
ProcessSteps_LoadTool[19][eeps_StepType] = eeButtonCheck_Empty;
ProcessSteps_LoadTool[20][eeps_StepType] = eeRegularStep;


ProcessSteps_UnloadTool[0][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[1][eeps_ServoNumber] = s_Tool_Height;
ProcessSteps_UnloadTool[2][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[3][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[4][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
ProcessSteps_UnloadTool[5][eeps_ServoNumber] = s_Tool_Height; //down to cutting height
ProcessSteps_UnloadTool[6][eeps_ServoNumber] = s_Tool_Lock;
ProcessSteps_UnloadTool[7][eeps_ServoNumber] = s_QuickSwapHotend_Lock;
ProcessSteps_UnloadTool[8][eeps_ServoNumber] = s_Cutter_Rotate;
ProcessSteps_UnloadTool[9][eeps_ServoNumber] = s_Cutter_Action;
ProcessSteps_UnloadTool[10][eeps_ServoNumber] = s_Cutter_Action; //here
ProcessSteps_UnloadTool[11][eeps_ServoNumber] = s_Tool_Height;
ProcessSteps_UnloadTool[12][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[13][eeps_ServoNumber] = s_Cutter_Action;
ProcessSteps_UnloadTool[14][eeps_ServoNumber] = s_Cutter_Action;
ProcessSteps_UnloadTool[15][eeps_ServoNumber] = s_Cutter_Rotate;
ProcessSteps_UnloadTool[16][eeps_ServoNumber] = s_Tool_Height; //lowest height
ProcessSteps_UnloadTool[17][eeps_ServoNumber] = s_Tool_Rotate; //to check button
ProcessSteps_UnloadTool[18][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[19][eeps_ServoNumber] = s_Tool_Height;
ProcessSteps_UnloadTool[20][eeps_ServoNumber] = s_Tool_Lock;
ProcessSteps_UnloadTool[21][eeps_ServoNumber] = s_Tool_Height;
ProcessSteps_UnloadTool[22][eeps_ServoNumber] = s_Tool_Height;
ProcessSteps_UnloadTool[23][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[24][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[25][eeps_ServoNumber] = s_Tool_Height;
ProcessSteps_UnloadTool[26][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[27][eeps_ServoNumber] = s_Tool_Rotate;
ProcessSteps_UnloadTool[28][eeps_ServoNumber] = s_WasteCup_Action;
ProcessSteps_UnloadTool[29][eeps_ServoNumber] = s_WasteCup_Action;

ProcessSteps_UnloadTool[0][eeps_Degrees] = pos_Tool_Rotate_UnderExtruder_ReleasedFromNozzleCollar;
ProcessSteps_UnloadTool[1][eeps_Degrees] = pos_Tool_Height_ToolFullyInsertedIntoExtruder_NoPressure; //pos_Tool_Height_ToolFullyInsertedIntoExtruder_ScrappingHotendMildPressure; //pos_Tool_Height_ToolFullyInsertedIntoExtruder_ScrappingHotendMildPressure;
ProcessSteps_UnloadTool[2][eeps_Degrees] = pos_Tool_Rotate_UnderExtruder_JerkConnectWithNozzleCollar;
ProcessSteps_UnloadTool[3][eeps_Degrees] = pos_Tool_Rotate_UnderExtruder_ConnectWithNozzleCollar;
ProcessSteps_UnloadTool[4][eeps_Degrees] = pos_QuickSwapHotend_Lock_Unlocked;
ProcessSteps_UnloadTool[5][eeps_Degrees] = pos_Tool_Height_ToolLowered_CuttingHeight;
ProcessSteps_UnloadTool[6][eeps_Degrees] = pos_Tool_Lock_Locked;
ProcessSteps_UnloadTool[7][eeps_Degrees] = pos_QuickSwapHotend_Lock_Locked;
ProcessSteps_UnloadTool[8][eeps_Degrees] = pos_Cutter_Rotate_Cutting;
ProcessSteps_UnloadTool[9][eeps_Degrees] = pos_Cutter_Action_Cut;
ProcessSteps_UnloadTool[10][eeps_Degrees] = pos_Cutter_Action_Open; //here
ProcessSteps_UnloadTool[11][eeps_Degrees] = pos_Tool_Height_ToolLowered_BelowCutterJaws;
ProcessSteps_UnloadTool[12][eeps_Degrees] = pos_Tool_Rotate_PastWasteCup;
ProcessSteps_UnloadTool[13][eeps_Degrees] = pos_Cutter_Action_Cut;
ProcessSteps_UnloadTool[14][eeps_Degrees] = pos_Cutter_Action_Open;
ProcessSteps_UnloadTool[15][eeps_Degrees] = pos_Cutter_Rotate_Stowed;
ProcessSteps_UnloadTool[16][eeps_Degrees] = pos_Tool_Height_LowestLevel;
ProcessSteps_UnloadTool[17][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck; //should have tool
ProcessSteps_UnloadTool[18][eeps_Degrees] = pos_Tool_Rotate_UnderToolHolder_CenteredUnderCurrentTool; //ready to lift into position
ProcessSteps_UnloadTool[19][eeps_Degrees] =
pos_Tool_Height_ToolLoweredButStillInHolder;
ProcessSteps_UnloadTool[20][eeps_Degrees] = pos_Tool_Lock_Unlocked;
ProcessSteps_UnloadTool[21][eeps_Degrees] = pos_Tool_Height_ToolFullyInsertedInHolder;
ProcessSteps_UnloadTool[22][eeps_Degrees] = pos_Tool_Height_ToolFullyInsertedInHolder_NoPressure;
ProcessSteps_UnloadTool[23][eeps_Degrees] = pos_Tool_Rotate_ReleaseFromHotendUnderToolHolder;
ProcessSteps_UnloadTool[24][eeps_Degrees] = pos_Tool_Rotate_BetweenBothNozzles; //pos_Tool_Rotate_LeftOfToolInHolder;
ProcessSteps_UnloadTool[25][eeps_Degrees] = pos_Tool_Height_LowestLevel;
ProcessSteps_UnloadTool[26][eeps_Degrees] = pos_Tool_Rotate_ButtonToolCheck;
ProcessSteps_UnloadTool[27][eeps_Degrees] = pos_Tool_Rotate_ButtingTheToolToTheLeftOfNext;
ProcessSteps_UnloadTool[28][eeps_Degrees] = pos_WasteCup_Action_Dump;
ProcessSteps_UnloadTool[29][eeps_Degrees] = pos_WasteCup_Action_Fill;

ProcessSteps_UnloadTool[0][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[1][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[2][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[3][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[4][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[5][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[6][eeps_IsIncludedInMMU] = false; //needs to be completely removed it's not used in either...
ProcessSteps_UnloadTool[7][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[8][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[9][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[10][eeps_IsIncludedInMMU] = true;//here
ProcessSteps_UnloadTool[11][eeps_IsIncludedInMMU] = false; //tool down almost all the way
ProcessSteps_UnloadTool[12][eeps_IsIncludedInMMU] = false; //tool rotate
ProcessSteps_UnloadTool[13][eeps_IsIncludedInMMU] = false; //pos_Cutter_Action_Cut;
ProcessSteps_UnloadTool[14][eeps_IsIncludedInMMU] = false; //pos_Cutter_Action_Open;
ProcessSteps_UnloadTool[15][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[16][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[17][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[18][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[19][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[20][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[21][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[22][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[23][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[24][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[25][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[26][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[27][eeps_IsIncludedInMMU] = true;
ProcessSteps_UnloadTool[28][eeps_IsIncludedInMMU] = false; //pos_WasteCup_Action_Dump;
ProcessSteps_UnloadTool[29][eeps_IsIncludedInMMU] = false; //pos_WasteCup_Action_Fill;

ProcessSteps_UnloadTool[0][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[1][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[2][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[3][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[4][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[5][eeps_MsDelayPerDegreeMoved] = 10; //lower to cutting height
ProcessSteps_UnloadTool[6][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[7][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[8][eeps_MsDelayPerDegreeMoved] = 6;//this makes the end position more repeatable than allowing the servo to control it's deceleration //6; //0; //cutter rotate
ProcessSteps_UnloadTool[9][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[10][eeps_MsDelayPerDegreeMoved] = 0;//here
ProcessSteps_UnloadTool[11][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[12][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[13][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[14][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[15][eeps_MsDelayPerDegreeMoved] = 0;//go full speed so that
the tool can be stowed symultaneously //6 slow to keep the servo from dying //0;
//cutter rotate
ProcessSteps_UnloadTool[16][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[17][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[18][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[19][eeps_MsDelayPerDegreeMoved] = 0; //60;
ProcessSteps_UnloadTool[20][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[21][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[22][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[23][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[24][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[25][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[26][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[27][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[28][eeps_MsDelayPerDegreeMoved] = 0;
ProcessSteps_UnloadTool[29][eeps_MsDelayPerDegreeMoved] = 0;

ProcessSteps_UnloadTool[0][eeps_MsDelayAfterCommandSent] = 550; //350;
ProcessSteps_UnloadTool[1][eeps_MsDelayAfterCommandSent] = 630; //430; //330; //230;
//up to nozzle collar level
ProcessSteps_UnloadTool[2][eeps_MsDelayAfterCommandSent] = 400; //200; //150; //90;
ProcessSteps_UnloadTool[3][eeps_MsDelayAfterCommandSent] = 90;
ProcessSteps_UnloadTool[4][eeps_MsDelayAfterCommandSent] = 110;
ProcessSteps_UnloadTool[5][eeps_MsDelayAfterCommandSent] = 0;
ProcessSteps_UnloadTool[6][eeps_MsDelayAfterCommandSent] = 0; //110; //lock tool.
lock moved to SetServoPosition()
ProcessSteps_UnloadTool[7][eeps_MsDelayAfterCommandSent] = 0;
ProcessSteps_UnloadTool[8][eeps_MsDelayAfterCommandSent] = 550; //650; //550; //500;
//190; //cutter rotate
ProcessSteps_UnloadTool[9][eeps_MsDelayAfterCommandSent] = 600; //550; //370; //130;
//cut
ProcessSteps_UnloadTool[10][eeps_MsDelayAfterCommandSent] = 200; //250; //370;//130;
//open //here
ProcessSteps_UnloadTool[11][eeps_MsDelayAfterCommandSent] = 0; //50; //uncomment for
Palette
ProcessSteps_UnloadTool[12][eeps_MsDelayAfterCommandSent] = 0; //90; //uncomment for
Palette
ProcessSteps_UnloadTool[13][eeps_MsDelayAfterCommandSent] = 0; //370; //130; //cut
//uncomment for Palette
ProcessSteps_UnloadTool[14][eeps_MsDelayAfterCommandSent] = 0; //370; //130; //open
//uncomment for Palette
ProcessSteps_UnloadTool[15][eeps_MsDelayAfterCommandSent] = 0; //cutter rotate
stowed //75; //50; //100; //200; //100; //50;//need slight delay just for the cutter
to rotate a little away from the filament and break the strand //130;s_Cutter_Rotate
no delay needed when stowing the cutter
ProcessSteps_UnloadTool[16][eeps_MsDelayAfterCommandSent] = 70;
ProcessSteps_UnloadTool[17][eeps_MsDelayAfterCommandSent] = 1000; //700; //200;
//button check should have tool
ProcessSteps_UnloadTool[18][eeps_MsDelayAfterCommandSent] = 80;
ProcessSteps_UnloadTool[19][eeps_MsDelayAfterCommandSent] = 180;
ProcessSteps_UnloadTool[20][eeps_MsDelayAfterCommandSent] = 0; //130;
ProcessSteps_UnloadTool[21][eeps_MsDelayAfterCommandSent] = 80;
ProcessSteps_UnloadTool[22][eeps_MsDelayAfterCommandSent] = 70;
ProcessSteps_UnloadTool[23][eeps_MsDelayAfterCommandSent] = 300; //400; //50;
//release from hotend which is now stowed
ProcessSteps_UnloadTool[24][eeps_MsDelayAfterCommandSent] = 50;
ProcessSteps_UnloadTool[25][eeps_MsDelayAfterCommandSent] = 350; //150; //lower to
lowest level
ProcessSteps_UnloadTool[26][eeps_MsDelayAfterCommandSent] = 400; //300; //180;
button check should be empty
ProcessSteps_UnloadTool[27][eeps_MsDelayAfterCommandSent] = 100;
ProcessSteps_UnloadTool[28][eeps_MsDelayAfterCommandSent] = 180;
ProcessSteps_UnloadTool[29][eeps_MsDelayAfterCommandSent] = 100;

ProcessSteps_UnloadTool[0][eeps_StepType] = eeRegularStep;
ProcessSteps_UnloadTool[1][eeps_StepType] = eeRegularStep;
ProcessSteps_UnloadTool[2][eeps_StepType] = eeRegularStep;
```

```
    ProcessSteps_UnloadTool[3][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[4][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[5][eeps_StepType] = eeExtrude; //lower to cutting height. extrude stage 2
    ProcessSteps_UnloadTool[6][eeps_StepType] = eeAddHalfDegreePrecision; //eeRegularStep; //eeAddHalfDegreePrecision; //eeRegularStep; //locking the nozzle-hotend into the end effector
    ProcessSteps_UnloadTool[7][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[8][eeps_StepType] = eeRegularStep; //eeAddHalfDegreePrecision; //eeRegularStep; //rotate to cutting position
    ProcessSteps_UnloadTool[9][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[10][eeps_StepType] = eeRegularStep; //open cutters //here
    ProcessSteps_UnloadTool[11][eeps_StepType] = eeRegularStep; //eeRetract;Not anymore that these steps are only for the palette //after cutters are open. retract
    ProcessSteps_UnloadTool[12][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[13][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[14][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[15][eeps_StepType] = eeRetract; //eeRegularStep;
    ProcessSteps_UnloadTool[16][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[17][eeps_StepType] = eeButtonCheck_HoldingTool;
    ProcessSteps_UnloadTool[18][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[19][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[20][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[21][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[22][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[23][eeps_StepType] = eeToolHolderPrepRotate;
    ProcessSteps_UnloadTool[24][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[25][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[26][eeps_StepType] = eeButtonCheck_Empty;
    ProcessSteps_UnloadTool[27][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[28][eeps_StepType] = eeRegularStep;
    ProcessSteps_UnloadTool[29][eeps_StepType] = eeRegularStep;


    ClearSerialBuffer();

    setupComplete = true;
    loopStarted = true;

    inputString.reserve(15);
    updateLCD("Ready to Swap!", "Insert: Empty");
}


int checkParity(String message) {
  int count = 0;
  for (int i = 0; i < message.length(); i++) {
    int value = message[i];
    while (value) {
      count++;
      value = value & (value - 1);
    }
  }
  return ~count & 1; // returns 1 for odd parity, 0 for even parity
}


void loop() {
  if (stringComplete) {
    int inputParity = inputString.charAt(inputString.length() - 1) - '0';
    String inputMessage = inputString.substring(0, inputString.length() - 1);
    String inputMessage_TextPart = inputMessage;
    int inputMessage_NumberPart = 0;

    // Find the index where the number starts
    int numIndex = -1;
    for (int i = 0; i < inputMessage.length(); i++) {
      if (isdigit(inputMessage.charAt(i))) {
        numIndex = i;
```

```
          break;
        }
      }

      // Extract the text part and number part if a number is found
      if (numIndex != -1) {
        inputMessage_TextPart = inputMessage.substring(0, numIndex);
        inputMessage_NumberPart = inputMessage.substring(numIndex).toInt();
      }

      Serial.print("number index: ");
      Serial.println(numIndex);
      Serial.print("inputMessage: ");
      Serial.println(inputMessage);
      Serial.print("inputMessage_TextPart: ");
      Serial.println(inputMessage_TextPart);
      Serial.print("inputMessage_NumberPart: ");
      Serial.println(inputMessage_NumberPart);

      if (checkParity(inputMessage) == inputParity) {
        if (inputMessage_TextPart == "octoprint") {
          printWithParity("swapper");
        } else if (inputMessage_TextPart == "load_insert") {
          insertNumber = inputMessage_NumberPart;
          load_insert(insertNumber);
          printWithParity("ok");
          delay(3000);
          updateLCD("Ready to Swap!", "Insert: " + String(insertNumber));
        } else if (inputMessage_TextPart == "unload_connect") {
          updateLCD_line1("Connect");
          unload_connect();
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "unload_pulldown") {
          updateLCD_line1("Pulldown");
          unload_pulldown();
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "unload_deploycutter") {
          updateLCD_line1("Deploy cutter");
          unload_deploycutter();
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "unload_cut") {
          updateLCD_line1("Cut");
          unload_cut();
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "unload_stowcutter") {
          updateLCD_line1("Stow cutter");
          unload_stowcutter();
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "unload_dumpwaste") {
          updateLCD_line1("Dump waste");
          unload_dumpwaste();
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "unloaded_message") {
          insertNumber = 0;
          updateLCD("Ready to Swap!", "Insert: Empty");
          printWithParity("ok");
          delay(3000);
        } else if (inputMessage_TextPart == "swap_message") {
          if (insertNumber == 0) {
            updateLCD_line1("Swapping -> " + String(inputMessage_NumberPart));
          } else {
            updateLCD_line1("Swapping " + String(insertNumber) + " -> " + String(
            inputMessage_NumberPart));
          }
          printWithParity("ok");
        } else if (inputMessage_TextPart == "wiper_deploy") {
          updateLCD_line1("Deploy wiper");
          wiper_deploy();
          printWithParity("ok");
          delay(3000);
          updateLCD_line1("Wiper deployed");
        } else if (inputMessage_TextPart == "wiper_stow") {
          updateLCD_line1("Stow wiper");
          wiper_stow();
          printWithParity("ok");
          delay(3000);
          updateLCD("Ready to Swap!", "Insert: " + insertNumber);
        } else {
          // Handle command not found
          printWithParity("Command not found");
        }
      } else {
        Serial.println("Parity check failed");
      }

      inputString = "";
      stringComplete = false;
    }
  }




  // function that continuously reads incoming serial data,
  //appending each character to a string until it encounters a newline character,
  //which signals the end of a message.
  void serialEvent() {
    while (Serial.available()) {
      char inChar = (char)Serial.read();
      if (inChar != '\n' && inChar != '\r') {
        inputString += inChar;
      }
      if (inChar == '\n') {
        stringComplete = true;
      }
    }
  }

  void ClearSerialBuffer()
  {
    while (Serial.available())
    {
          Serial.read();
    }
  }



  void load_insert(int toolToLoad)
  {

      toolIsLoaded = true;
      bool errorResume = false;
      //return; //js

      currentStepOfProcess = 0;
      ToolHolder_AlignToThisTool(toolToLoad);


      numberOfStepsToProcess = numOfProcessSteps_LoadTool;
      CurrentProcessType = eeLoadTool;

      while (currentStepOfProcess < numberOfStepsToProcess)
      {
          // RefreshPositionServoInfo();  //moved to the ProcessStep method
          // SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);//moved to the ProcessStep method
          // delay(ps_msDelayAfterCommandSent); //moved to the ProcessStep method
          // Serial.println("process step");
          ProcessStep(); //***this delays the next step, checks the buttons, and runs special extrudes on the printer

          if(!InErrorState)
          {
              currentStepOfProcess++;
          }
          //if in error button check
          //unwind to waiting for user input
          else
          {
              // Serial.println("park the tool actuator.");
              SetServoPosition(s_Tool_Rotate, 95, 0);
              //unlock the tool holder
              pulselength = map(180, 0, servos[s_Tool_Lock][eeMaxAngle], servo_pwm_min, servo_pwm_max);
              pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
              delay(200);

              do
              {
                  buttonPress = analogRead(0);
                  // Serial.println(buttonPress);
              }while (buttonPress < 820 || buttonPress > 830);


              //return the tool lock to the pre-error state
              pulselength = map(servos[s_Tool_Lock][eeCurrentAngle], 0, servos[s_Tool_Lock][eeMaxAngle], servo_pwm_min, servo_pwm_max);
              pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
              delay(200);

              Serial.println("Error was reset");
              lcd.clear();
              lcd.setCursor(0,0);
              lcd.print("Loading");
              InErrorState = false;
          }
      }


      // toolIsLoaded = true; //should be at the top so that it works when SetupMode
    currentStepOfProcess = 0;

      lcd.setCursor(0,1);
      lcd.print("Heating nozzle");
      //delay(30000);//no delay needed because in PS there is an M109 wait for temp. //25000); //30000); //must delay some time so the nozzle-hotend can heatup
      lcd.setCursor(0,1);
      lcd.print("                ");

    ClearSerialBuffer();
  }

  void unload_connect(){

  }


  void unload_pulldown(){

  }
```

```
void unload_deploycutter(){

}

void unload_cut(){

}

void unload_stowcutter(){

}

void unload_dumpwaste(){

}

void wiper_deploy(){

}

void wiper_stow(){

}


void Unload()
{
    if (!toolIsLoaded) return;

    toolIsLoaded = false;
    //return; //js

    currentStepOfProcess = 0;
    numberOfStepsToProcess = numOfProcessSteps_UnloadTool;
    CurrentProcessType = eeUnloadTool;
    //already aligned to the current tool. The currentTool is set in
    ToolHolder_AlignToThisTool()


    while (currentStepOfProcess < numberOfStepsToProcess)
    {
        ProcessStep(); //***this delays the next step, checks the buttons, and runs
        special extrudes on the printer

        if(!InErrorState)
        {
            do
            {
                currentStepOfProcess++;
            } while (ProcessSteps_UnloadTool[currentStepOfProcess][eeps_IsIncludedInMMU]
            == false);
        }
        //if in error button check
        //unwind to waiting for user input
        else
        {
            SetServoPosition(s_Tool_Rotate, 95, 0);
            //unlock the tool holder
            pulselength = map(180, 0, servos[s_Tool_Lock][eeMaxAngle], servo_pwm_min,
            servo_pwm_max);
            pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
            delay(200);

            do
            {
                buttonPress = analogRead(0);
                // Serial.println(buttonPress);
            }while (buttonPress < 820 || buttonPress > 830);

            //return the tool lock to the pre-error state
            pulselength = map(servos[s_Tool_Lock][eeCurrentAngle], 0, servos[s_Tool_Lock][eeMaxAngle], servo_pwm_min, servo_pwm_max);
            pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
            delay(200);

            Serial.println("Error was reset");
            lcd.clear();
            lcd.setCursor(0,0);
            lcd.print("Unloading");
            InErrorState = false;
        }
    }
  currentStepOfProcess = 0;
}

void RefreshPositionServoInfo()
{
    switch(CurrentProcessType)
    {
        case eeLoadTool: //Load process
            ps_currentServo = ProcessSteps_LoadTool[currentStepOfProcess][eeps_ServoNumber];
            ps_targetAngle = ProcessSteps_LoadTool[currentStepOfProcess][eeps_Degrees];
            ps_msDelayPerDegreeMoved = ProcessSteps_LoadTool[currentStepOfProcess][eeps_MsDelayPerDegreeMoved];
            ps_msDelayAfterCommandSent = ProcessSteps_LoadTool[currentStepOfProcess][eeps_MsDelayAfterCommandSent] + msDelayAfterCommandSent_Buffer;
            break;
        case eeUnloadTool: //unload process
            ps_currentServo = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_ServoNumber];
            ps_targetAngle = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_Degrees];
            ps_msDelayPerDegreeMoved = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_MsDelayPerDegreeMoved];
            ps_msDelayAfterCommandSent = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_MsDelayAfterCommandSent] + msDelayAfterCommandSent_Buffer;
            break;
    }
}

bool CheckButton_Pressed()
{
    delay(10);

    // if(digitalRead(CheckButton_Pin)==1)
    if(analogRead(CheckButton_Pin) > 1020)
    {
        return true;
    }
    else
    {
        return false;
    }
}

void ProcessStep()
{
    int stepType = 0;

    switch(CurrentProcessType)
    {
        case eeLoadTool: //Load process
            stepType = ProcessSteps_LoadTool[currentStepOfProcess][eeps_StepType];
            break;
        case eeUnloadTool: //unload process
            stepType = ProcessSteps_UnloadTool[currentStepOfProcess][eeps_StepType];
            break;
    }


    switch(stepType)
    {
        case eeButtonCheck_Empty:
            RefreshPositionServoInfo();
            SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
            delay(ps_msDelayAfterCommandSent);  //delay first, then check button.
            otherwise the button cannot ever be pressed

            if(CheckButton_Pressed() && ErrorCheckingEnabed)
            {
                Serial.println("ERROR 1");
                lcd.clear();
                lcd.setCursor(0,0);
                lcd.print("ERROR->Not Empty");
                lcd.setCursor(0,1);
                lcd.print("S to retry");
                InErrorState = true;
            }
            else
            {
                if(CheckButton_Pressed())
                {
                  Serial.println("Button pressed.");
                }
                else
                {
                  Serial.println("Button NOT pressed.");
                }

                InErrorState = false;
            }
            break;
        case eeButtonCheck_HoldingTool:
            RefreshPositionServoInfo();
            SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
            delay(ps_msDelayAfterCommandSent);  //delay first, then check button.
            otherwise the button cannot ever be pressed

            if(!CheckButton_Pressed() && ErrorCheckingEnabed)
            {
                Serial.println("ERROR 2");
                lcd.clear();
                lcd.setCursor(0,0);
                lcd.print("ERROR->Empty");
                lcd.setCursor(0,1);
                lcd.print("S to retry");
                InErrorState = true;
            }
            else
            {
                if(CheckButton_Pressed())
                {
                  Serial.println("Button pressed.");
                }
                else
                {
                  Serial.println("Button NOT pressed.");
                }

                InErrorState = false;
            }
            break;
        case eeExtrude:
            LockToolPartWayThru = true;

            // Serial.println("Extrude");
```

```
                Serial.write(90);
                Serial.write(91);
                Serial.write(93);
                Serial.write(94);
                Serial.write(1); //direction 1=extrude
                Serial.write(55); //53); //53 length
                Serial.write(66); //5); //80, 75, 72, 70,67, 65 //feedrate

                delay(97);//delay before servo movement to allow the extrude to begin on the
                printer

                RefreshPositionServoInfo();
                SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);

                delay(ps_msDelayAfterCommandSent);
                break;
            case eeRetract:
                // Serial.println("Retract");
                Serial.write(90);
                Serial.write(91);
                Serial.write(93);
                Serial.write(94);
                Serial.write(2); //direction 2=retract
                Serial.write(70); //56); //pass:56); Fail:53 //retract a little too much,
                then add back after load heat up IF this is a 'same color nozzle size
                switch' otherwise the tool change will restore the difference//53);
                //54);//63); //54); //53); //length
                Serial.write(65); //120); //3900//feedrate

                RefreshPositionServoInfo();
                SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
                Serial.print("eeRetract ms delay:");
                Serial.println(ps_msDelayPerDegreeMoved);
                delay(ps_msDelayAfterCommandSent);
                break;
            case eeToolHolderPrepRotate://rotate the tool holder slightly to account for the
            pull of the end effector when releasing the nozzle collar
                servos[s_ToolHolder_Rotate][eeCurrentAngle] = servos[s_ToolHolder_Rotate][
                eeCurrentAngle] + eeToolHolderPrepRotate_Degrees;
                pulselength = map(servos[s_ToolHolder_Rotate][eeCurrentAngle], servoMinAngle,
                 servos[s_ToolHolder_Rotate][eeMaxAngle], servo_pwm_min, servo_pwm_max);
                pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, pulselength);

                RefreshPositionServoInfo();
                SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
                delay(ps_msDelayAfterCommandSent);
                break;
            case eeToolHolderPrepUNrotate://rotate the tool holder slightly to account for
            the pull of the end effector when releasing the nozzle collar
                //UNrotate it
                RefreshPositionServoInfo();

                SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
                //rotate tool actuator
                delay(70); //60); //55); //65); //80); //120); //60); //50); //this delay
                always the tool actuator to begin moving, then the tool holder rotates at
                the same time and when the optimal position is acheived the end effector
                slips onto the nozzles collar.

                //Begin rotate TOOL HOLDER
                pulselength = map(servos[s_ToolHolder_Rotate][eeCurrentAngle] -
                eeToolHolderPrepUNrotate_Degrees, servoMinAngle, servos[s_ToolHolder_Rotate][
                eeMaxAngle], servo_pwm_min, servo_pwm_max);
                pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, pulselength);
                //End rotate TOOL HOLDER

                delay(ps_msDelayAfterCommandSent);

                //rotate it back
                pulselength = map(servos[s_ToolHolder_Rotate][eeCurrentAngle], servoMinAngle, servos[s_ToolHolder_Rotate][eeMaxAngle], servo_pwm_min, servo_pwm_max);
                pwm.setPWM(servos[s_ToolHolder_Rotate][eePinNum], 0, pulselength);
                break;
            case eeRegularStep:
                RefreshPositionServoInfo();
                SetServoPosition(ps_currentServo, ps_targetAngle, ps_msDelayPerDegreeMoved);
                delay(ps_msDelayAfterCommandSent);
                break;
            case eeAddHalfDegreePrecision:
                int precisionPulseLength = 0;
                RefreshPositionServoInfo();
                precisionPulseLength = fMap((float)ps_targetAngle + (float)0.5, 0, servos[ps_currentServo][eeMaxAngle], servo_pwm_min, servo_pwm_max);

                pwm.setPWM(servos[ps_currentServo][eePinNum], 0, precisionPulseLength);
                delay(ps_msDelayAfterCommandSent);
                break;
        }
    }

    void SetServoPosition(int ServoNum, int TargetAngle, int msDelay)
    {
        int currentAngle = servos[ServoNum][eeCurrentAngle];
        int angleDifference = TargetAngle - currentAngle;

        int msCountedBeforeLock = 0;

      //if the msDelay is zero then don't use a loop
      if(msDelay == 0)
      {
          //Serial.println("zero delay");
          servos[ServoNum][eeCurrentAngle] = TargetAngle;
          pulselength = map(servos[ServoNum][eeCurrentAngle], 0, servos[ServoNum][eeMaxAngle], servo_pwm_min, servo_pwm_max);
          pwm.setPWM(servos[ServoNum][eePinNum], 0, pulselength);
      }
      //else use a loop to inject the delay and fake accel
      else
      {
        //Serial.println("ms delay");
        if (angleDifference > 0)
        {
            for(int i = currentAngle; i <= TargetAngle; i++)
            {
                servos[ServoNum][eeCurrentAngle] = i;
                pulselength = map(servos[ServoNum][eeCurrentAngle], 0, servos[ServoNum][eeMaxAngle], servo_pwm_min, servo_pwm_max);
                pwm.setPWM(servos[ServoNum][eePinNum], 0, pulselength);
                delay(msDelay);


                //deploy the tool lock part way thru the extrude
                if (LockToolPartWayThru)
                {
                    msCountedBeforeLock += msDelay;

                    //lock the tool
                    if(msCountedBeforeLock >= numMsUntilLock)
                    {
                        LockToolPartWayThru = false;
                        servos[s_Tool_Lock][eeCurrentAngle] = pos_Tool_Lock_Locked;
                        pulselength = map(servos[s_Tool_Lock][eeCurrentAngle], 0, servos[ServoNum][eeMaxAngle], servo_pwm_min, servo_pwm_max);
                        pwm.setPWM(servos[s_Tool_Lock][eePinNum], 0, pulselength);
                    }
                }
            }
        }
        else
        {
            for(int i = currentAngle; i >= TargetAngle; i--)
            {
                servos[ServoNum][eeCurrentAngle] = i;
                pulselength = map(servos[ServoNum][eeCurrentAngle], 0, servos[ServoNum][
                eeMaxAngle], servo_pwm_min, servo_pwm_max);
                pwm.setPWM(servos[ServoNum][eePinNum], 0, pulselength);
                delay(msDelay);
            }
        }
      }
    }
```