```
#include <SPI.h>
#include <mcp2515.h>

struct can_frame BmsLimits;
struct can_frame BmsSOC;
struct can_frame BmsStatus1;
struct can_frame BmsErrors;
struct can_frame ACCharging;

struct can_frame canMsg;

MCP2515 mcp2515(10);

float current = 0;
uint8_t soc = 0;
uint16_t voltage = 0;
uint16_t maxACCurrent = 0;

// use millis() to periodically send the can frames
unsigned long previousMillis = 0;
const long interval = 100;

void setup() {
  //  while (!Serial); // wait for serial monitor to start
  Serial.begin(115200); // start serial monitoring at 115200 baud

  // start the CAN bus
  mcp2515.reset();
  mcp2515.setBitrate(CAN_500KBPS, MCP_8MHZ); // MCP_20MHZ, MCP_16MHZ, MCP_8MHZ
  mcp2515.setNormalMode();

  Serial.println("Done setting up, starting transmitting can frames");
}

void loop() {
  unsigned long currentMillis = millis(); // get the current time

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    uint16_t ChargeVoltageLimit = 393 * 10;
    uint16_t ChargeCurrentLimit = 50 * 10;
    uint16_t DischargeVoltageLimit = 307 * 10;
    uint16_t DischargeCurrentLimit = 50 * 10;

    uint16_t HvBatterySOC = soc; // implemented

    uint16_t HvBatteryVoltage = voltage / 10; // should work
    int16_t HvBatteryCurrent = current * 10; // this 2 should work
    int16_t HvBatteryTemp = 8 * 10; //donk probibly needs to be activated b4 final instalation

    uint16_t AClimit = maxACCurrent * 10; //amp
    uint16_t ACvolt = 230 * 10; // we r in europa so 230 is the standard


    // set all the data of the can frames
```

```
    BmsLimits.can_id  = 0x351; // 849
    BmsLimits.can_dlc = 8;
    BmsLimits.data[0] = (ChargeVoltageLimit & 0xff);
    BmsLimits.data[1] = (ChargeVoltageLimit >> 8);
    BmsLimits.data[2] = (ChargeCurrentLimit & 0xff);
    BmsLimits.data[3] = (ChargeCurrentLimit >> 8);
    BmsLimits.data[4] = (DischargeVoltageLimit & 0xff);
    BmsLimits.data[5] = (DischargeVoltageLimit >> 8);
    BmsLimits.data[6] = (DischargeCurrentLimit & 0xff);
    BmsLimits.data[7] = (DischargeCurrentLimit >> 8);

    BmsErrors.can_id  = 0x35A; // 858
    BmsErrors.can_dlc = 4;
    BmsErrors.data[0] = 0x00;
    BmsErrors.data[1] = 0x00;
    BmsErrors.data[2] = 0x00;
    BmsErrors.data[3] = 0x00; //bit 4 2 1 2 enable charging

    BmsSOC.can_id  = 0x355; // 853
    BmsSOC.can_dlc = 6;
    BmsSOC.data[0] = (HvBatterySOC & 0xff);
    BmsSOC.data[1] = (HvBatterySOC >> 8);
    BmsSOC.data[2] = 0xff;
    BmsSOC.data[3] = 0xff;
    BmsSOC.data[4] = 0xff;
    BmsSOC.data[5] = 0xff;

    BmsStatus1.can_id  = 0x356; // 854
    BmsStatus1.can_dlc = 6;
    BmsStatus1.data[0] = (HvBatteryVoltage & 0xff);
    BmsStatus1.data[1] = (HvBatteryVoltage >> 8);
    BmsStatus1.data[2] = (HvBatteryCurrent & 0xff);
    BmsStatus1.data[3] = (HvBatteryCurrent >> 8);
    BmsStatus1.data[4] = (HvBatteryTemp & 0xff);
    BmsStatus1.data[5] = (HvBatteryTemp >> 8);

    ACCharging.can_id = 0x19B50407;
    ACCharging.can_dlc = 8;
    ACCharging.data[0] = 0xff;
    ACCharging.data[1] = 0xff;
    ACCharging.data[2] = (AClimit >> 8);
    ACCharging.data[3] = (AClimit & 0xff);
    ACCharging.data[4] = (ACvolt >> 8);
    ACCharging.data[5] = (ACvolt & 0xff);
    ACCharging.data[6] = 0xff;
    ACCharging.data[7] = 0xff;

    // send the can frames

    if (voltage != 0) {

      mcp2515.sendMessage(&BmsLimits);  // implemented with test data
      mcp2515.sendMessage(&BmsErrors);  // implemented for testing only, will
never give errors
      mcp2515.sendMessage(&BmsSOC);     // implemented with test data
      mcp2515.sendMessage(&BmsStatus1); // implemented with test data
      mcp2515.sendMessage(&ACCharging);

      Serial.println("all frames sent, small delay and repeat");
```

```
    }
    else {
      Serial.println("not sending can to ccs yet. emus still booting?");
    }
  }

  // read the can bus
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    unsigned long id = canMsg.can_id;
    if (id == 874) { // 874 is the id of a can frame that is sent from the ccs controller with the errors
      if (canMsg.data[0] & (1<<0)) {Serial.print("HVPreChargeFault ");}
      if (canMsg.data[0] & (1<<1)) {Serial.print("CCSContactorFault ");}
      if (canMsg.data[0] & (1<<2)) {Serial.print("HVILFault ");}
      if (canMsg.data[0] & (1<<3)) {Serial.print("BMSCommsFault ");}
      if (canMsg.data[0] & (1<<4)) {Serial.print("BMSFault ");}
      if (canMsg.data[0] & (1<<5)) {Serial.print("CCSECUFault ");}
      if (canMsg.data[0] & (1<<6)) {Serial.print("PTCTempFault ");}
      if (canMsg.data[0] & (1<<7)) {Serial.print("ChargeProtocolFault ");}
      if (canMsg.data[1] & (1<<0)) {Serial.print("IncompatibleCCSCharger ");}
      if (canMsg.data[1] & (1<<1)) {Serial.print("ChargeMode ");}
      if (canMsg.data[1] & (1<<2)) {Serial.print("PlugPresent ");}
      if (canMsg.data[1] & (1<<3)) {Serial.print("InletMotor ");}
      if (canMsg.data[1] & (1<<4)) {Serial.print("CCSContactorStatus ");}
      if (canMsg.data[1] & (1<<5)) {Serial.print("HVPresent ");}
      if (canMsg.data[1] & (1<<6)) {Serial.print("InletFault ");}
      if (canMsg.data[1] & (1<<7)) {Serial.print("StopchargeSwitch ");}

      Serial.print("BatteryVoltageSense:");
      Serial.print(((canMsg.data[2]<<8)|canMsg.data[3])/10.0);
      Serial.print(" CCSVoltageSense:");
      Serial.print(((canMsg.data[4]<<8)|canMsg.data[5])/10.0);

      Serial.println();
    }
    if ( id == 855 ) { // 855 is the id of the can frame that is sent from the ccs controller with the ac current limit
      maxACCurrent = canMsg.data[1];
      Serial.print("max ac current: "); //maxACCurrent
      Serial.println(maxACCurrent);
    }
    if (id == 0x99B50500) {
      current = canMsg.data[1] / 10.0;
      Serial.print("amps: ");
      Serial.println(current);
      soc = canMsg.data[6];
      Serial.print("usoc: ");
      Serial.println(soc);
    }
    else if (id == 0x99B50001) {
      uint8_t voltage1 = canMsg.data[3];
      uint8_t voltage2 = canMsg.data[4];
      Serial.print("volt: ");
      voltage = voltage1 << 8 | voltage2;
      Serial.println(voltage / 100.0);
    }
  }
}
```