CAN_TxHeaderTypeDef TxHeader; - used for storing header information that will be sent through CAN.
CAN_RxHeaderTypeDef TxHeader; - used for storing header information that will be received through CAN.
CAN_FilterTypeDef canfilterconfig; - used to store the filters for receiving messages.

TxHeader.IDE - used to define what id we are using, standard or extended.
TxHeader.StdId - used to define the id of the transmitter.
TxHeader.RTR - indicates whether we are sending a data frame or remote frame.
TxHeader.DLC - indicates the datalength we will be sending

uint8_t TxData[8]; -used to store the data that we are sending through CAN.
uint8_t RxData[8]; -used to store the data that we are receiveing through CAN.
uint32_t TxMailbox - a mailbox used to carry data over CAN.

canfilterconfig.FilterActivation - used to activate or deactivate
canfilterconfig.FilterBank - for which filter bank to use from the assigned ones
canfilterconfig.FilterFIFOAssignment - specifies which out of the 2 FIFO's we are going to use to receive the message(FIFO0 or FIFO1)

canfilterconfig.FilterIdHigh - for comparing the higher 16 bits of the ID register.
canfilterconfig.FilterIdLow - for comparing the lower 16 bits of the ID register.
canfilterconfig.FilterMaskIdHigh - for comparing the higher 16 bits of the MASK register.
canfilterconfig.FilterMaskIdLow - for comparing the lower 16 bits of the MASK register.
canfilterconfig.FilterMode - specifies the filter mode to be initialized.
canfilterconfig.FilterScale - specifies the filter scale.
canfilterconfig.SlaveStartFilterBank - the number of filter banks assigned.

HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox); - a function used to transmit a CAN message, simplified by making SendCANMessage function which takes the id of the transmitting CAN device, the data that we will be sending and the length of that data.

HAL_CAN_GetRxMessage(&hcan1, CAN_RX_FIFO0, &RxHeader, RxData); - a function used to receive a CAN message, simplified by using the ReceiveChargerCANMessage function which takes the ID of the device that is sending the message to it and save the message data in RxData;

GetChargingState function is used to compare the soc data, temperature data, charging current data and charging voltage data, it then compares them to the maximum allowed parameters and if anything is over the threshold returns CHARGING_STATE_OFF;(1), and if everything is within the working parameters it returns CHARGING_STATE_ON;(0);

EncodeDataForCAN function receives the data that needs to be encoded to be sent through CAN.

DecodeChargerMessage function is used to decode the data from received from the charger.

ChargingStateAlgorithm() function is used to send a message to the charger through can on whether it should power off or stay on. First we get the data that the charger sent through the ReceiveChargerCANMessage function, then we use the DecodeChargerMessage function to decode data so we can compare it with the GetChargingState function and we save the data it returns into an integer.Then we compare the value of charging state so that we know what message to send. After that we prepare the data for sending through CAN by using the function EncodeDataForCAN and finally send it using the SendCANMessage function.