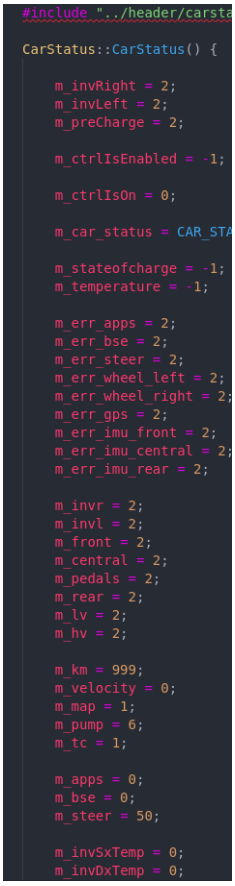
General refactor of "carstatus.cpp"

2019 November 15

1 Introduction

The SteeringWheel UI needs a lot of variables in order to works properly. In the past those variables were all putted toghether in one place, "Carstatus". This class was created to store them and to collect methods associated to them. To date "Carstatus" contains over 50 variables. At this point there is a lot of confusion and the code is difficult to be read.



```
#include "../header/carsta

CarStatus::CarStatus() {

    m_invRight = 2;
    m_invLeft = 2;
    m_preCharge = 2;

    m_ctrlIsEnabled = -1;

    m_ctrlIsOn = 0;

    m_car_status = CAR_STA

    m_stateofcharge = -1;
    m_temperature = -1;

    m_err_apps = 2;
    m_err_bse = 2;
    m_err_steer = 2;
    m_err_wheel_left = 2;
    m_err_wheel_right = 2;
    m_err_gps = 2;
    m_err_imu_front = 2;
    m_err_imu_central = 2;
    m_err_imu_rear = 2;

    m_invr = 2;
    m_invl = 2;
    m_front = 2;
    m_central = 2;
    m_pedals = 2;
    m_rear = 2;
    m_lv = 2;
    m_hv = 2;

    m_km = 999;
    m_velocity = 0;
    m_map = 1;
    m_pump = 6;
    m_tc = 1;

    m_apps = 0;
    m_bse = 0;
    m_steer = 50;

    m_invSxTemp = 0;
    m_invDxTemp = 0;
```

The ideal solution is to store variables and methods associated in different classes and use them from "Carstatus" using a "has a" for each class.
——————————————INSERT CLASS DIAGRAM HERE————————————————-

2 Implementation issues

These changes will make code finally readable but some methods require to remain into "Carstatus". These methods are thoose that deal with sending changed data after emits.
The only possible solution is to make some getters inside the new classes and call them from "Carstatus". Therefore there will still be functions in this class but there will be much less.
Another side effect of this implementation is the needs to create a ".h" (and ".cpp") file for each new class. There will be more or less 10 new classes, that means 10 new header and source files.

3 Refactor

3.1 Inverter

```
int m_invRight;
int m_invLeft;
int m_preCharge;
int m_invSxTemp;
int m_invDxTemp;
```

These 5 variables has been moved into a new Class called Inverters. Now Carstatus' functions use getter and setter of this class in order to maintain an object oriented paradigm. Variables' names are also changed to keep a certain standard.
So, the class now looks like this:

```
public:
    Inverters();
    void setLeftInverter(int);
    void setRightInverter(int);
    void setPreCharge(int);
    void setLeftInverterTemperature(int, int);
    void setRightInverterTemperature(int, int);
    int getLeftInverter() const;
    int getRightInverter() const;
    int getPreCharge() const;
```

```
    int getLeftInverterTemperature() const;
    int getRightInverterTemperature() const;
private:
    int invLeft;
    int invRight;
    int preCharge;
    int invLeftTemp;
    int invRightTemp;
```

3.2 Control

```
int m_ctrlIsEnabled;
int m_ctrlIsOn;
int m_car_status;
```

These 3 has been moved into Control class with respective getters and setters

```
public:
    Control();
    void setCtrlIsEnabled(bool);
    void setCtrlIsOn(bool);
    bool setCarStatus(int);
    bool getCtrlIsEnabled() const;
    bool getCtrlIsOn() const;
    int getCarStatus() const;
private:
    bool ctrlIsEnabled;
    bool ctrlIsOn;
    int carStatus;
```

As you could notice

`ctrlIsEnabled`

and

`ctrlIsOn`

now are boolean. This decision was taken because these variables rappresent boolean values and two integers were a waste of resources.

3.3 Race

```
int m_speed;
int m_velocity;
int m_km;
```

These 3 have been moved into the new Race class with respective getters and setters.
The new Race class looks like:

```
public:
    Race();
    void setSpeed(int, int);
    void setVelocity(int);
    void setKm(int, int);
    int getSpeed() const;
    int getVelocity() const;
    int getKm() const;
private:
    int speed;
    int velocity;
    int km;
```

In order to maintain the new standard the formulas to calcolate speed and km has been inserted into setSpeed and setKm functions, this could not be done for setVelocity function because in carStatus.cpp there are 2 different formulas to calcolate the velocity, so it is easier to maintain that formulas into carStatus.

3.4 Warning

```
int m_invr;
int m_invl;
int m_front;
int m_rear;
int m_lv;
int m_hv;
int m_central;
int m_pedals;
int m_warning;
```

These old variables have been moved to Warning, the new class; as always the new class has all getters and setters required.
The new Warning class looks like:

```
public:
    Warning();
    void setInvRight(int);
    void setInvLeft(int);
    void setFront(int);
    void setRear(int);
    void setLv(int);
    void setHv(int);
    void setCentral(int);
    void setPedals(int);
    void setWarning(int);
    int getInvRight() const;
    int getInvLeft() const;
    int getFront() const;
    int getRear() const;
    int getLv() const;
    int getHv() const;
    int getCentral() const;
    int getPedals() const;
    int getWarning() const;
private:
    int invRight;
    int invLeft;
    int front;
    int rear;
    int lv;
    int hv;
    int central;
    int pedals;
    int warning;
```

The **Q_PROPERTY** called warning in carStatus.h has been modified to getWarning (and valWarning for the variable name) in order to free the warning instace and to give it to the new Warning class instace. This means that all the references in qml to the older warning **Q_PROPERTY** has

been modified to getWarning or valWarning. The file in question is StatusLED.qml. In this way it has been possible to maintain our new standard. Schematically:

- Older **Q_PROPERTY** in carStatus:

```
Q_PROPERTY(int warning READ warning NOTIFY warningChanged);
```

- New **Q_PROPERTY** in carStatus:

```
Q_PROPERTY(int varWarning READ getWarning NOTIFY warningChanged);
```

- Older **StatusLED.qml**:

```
property var warning : CarStatus.warning;
...
warningVal = CarStatus.warning;
```

- New **StatusLED.qml**:

```
property var warning : CarStatus.getWarning;
...
warningVal = CarStatus.varWarning;
```

3.5 Manettini

```
int m_map;
int m_tc;
int m_pump;
```

These 3 variables belong to Steering Wheel's physical manettini which control respectively car's power delivery, traction control and cooling. These are now moved into Manettini class.

```
public:
    Manettini();
    void setMap(int);
    void setTc(int);
    void setPump(int);
    int getMap() const;
    int getTc() const;
    int getPump() const;
    void incMap(const int);
    void incPump(const int);
    void incTc(const int);
private:
    int map;
    int tc;
    int pump;
```

As can be seen besides getters, setters and the class costructor, it has been also added

`incMap incPump incTc.`

Their task is to increment the value of each manettino by one, if this value exceeds a certain amount it runs a modulus in order to reset the variable.