Yrkes Akademin
Vi hjälper dig att lyckas!

# Group Assignment

Yrkes Akademin
Vi hjälper dig att lyckas!

# Description

The Group project consists of a group of small personal assignments. Let's imagine, that a customer asked your team to create an abstract product, which can perform a few different features listed below:

- It is written on C++ language and has Code Coverage > 50%
- The project is modular and easy expandabable ( by adding a folder or git submodule).
- GoogleTest should be used as a UT test framework.

We are quite happy, because we as a team can discuss with the customer the final product details and the content. The idea of a project: we are developing Proof Of a Concept (PoC) of a middle layer for an embedded device. We don't need to use real drivers and interfaces, but we should use "dummy" interfaces instead (we will read and write to files or in a console instead of using i2c, UART, socket and etc.)

The skeleton of a PoC should be an application, which is calling a set of functions provided by modules. Key Modules are:

- Component "File IO", which contains functions to read a file and write into a file.
- Component "CAN Messages", which contains a json file (with list of CAN messages ID, names and other input data) and python script to generate from given json file a resulting header file with structures representing CAN message.
- Main Component, which is calling functions from the "File IO" component with input data generated from the component "CAN Messages".
- UT and Component tests should be used for most of the components ( depends on a situation).
- Component tests should be used for the Main Component.
- It is fine to have a mono-repository for components. But if a team would like to have a few separate repositories, it is welcome, but then the team should consider CI pipelines and their flows.

Yrkes Akademin
Vi hjälper dig att lyckas!

# Tests

Yrkes Akademin
Vi hjälper dig att lyckas!

# Unit & Components Tests

## Unit Tests

For the C++, the GoogleTest framework should be used. A customer really likes GoogleTest! But for the python scripts "pytest" should be used (if possible).

## Components Tests

For a component, which is using input and output files, a component test written on Python should be used ( UT for python component test is not needed, it's good! ). A customer is quite strange, would like to see component test implemented on Python with classes. (Nothing fancy, a class with 2 fields (input,output) should be enough).

## IMPORTANT Note:

You should design the architecture to be able to mock functions!

Yrkes Akademin
Vi hjälper dig att lyckas!

# Components

Yrkes Akademin
Vi hjälper dig att lyckas!

# Component "File IO"

A customer really wants to have a component to work with files. And due to PoC, he is agreed to have for now a limitations to work with std::string type for now. The component should have the following functions:

- read from a file a line of string.
- read from a file a container of strings.
- write to a file a string, which contains exactly 1 line.
- write to a file a container of strings.

To make our life a bit easier, we can choose what container for string to use ( std::list, std::vector, etc.). And a customer told us the rule: there is only one command on the line.

- The component should be implemented as a library but also have a dummy application, which is working as "echo" ( meaning, if an user called dummy App with a file, the app should call functions to read the file and call functions to write an output file with the same content).
- The component should have UT for the functionality (so plan ahead the code structure to be able to mock specific functions 😉 ).
- The component should also have a component test written on Python. The idea is to create a temporary file with predefined input, call a dummy app, compare the generated output file with predefined data.).

Yrkes Akademin
Vi hjälper dig att lyckas!

# Component "CAN Messages"

The main goal of this component is to generate from given json file an output C++ class containing methods which are returning correspond string for converted CAN message. For example for the given JSON document containing:

```
//let's imagine, the filename is "min_signals.json"
{
    "id":"0x100",
    "signals":[
        {"name":"temperature", "type":"float", "length":10, "comment":"the ambient temperature"},
        {"name":"humidity", "type":"uint8_t", "length":7, "comment":"the ambient humidity percentage"},
    ]
}
```

Yrkes Akademin
Vi hjälper dig att lyckas!

# Component "CAN Messages". part2

Should generate a C++ class:

```
//header file with the name of json file
//for the example above: min_signals.h
#ifndef HEADER_MIN_SIGNALS_H
#define HEADER_MIN_SIGNALS_H

class CAN_min_signals {
    public:
        CAN_min_signals();
        /*
        get the ambient temperature
        */
        std::string get_temperature();
        /*
        set the ambient temperature
        */
        std::string set_temperature(float newValue);

        /*
        get the ambient humidity percentage
        */
        std::string get_humidity();

        // continued on the next page
```

Yrkes Akademin
Vi hjälper dig att lyckas!

## Component "CAN Messages". part3

```
        /*
        set the ambient humidity percentage
        */
        std::string set_humidity(uint8_t newValue);
    private:
        uint8_t m_startMsgId;
        uint8_t m_temperatureGetMsgId;
        uint8_t m_temperatureSetMsgId;
        uint8_t m_humidityGetMsgId;
        uint8_t m_humiditySetMsgId;
};

#endif //HEADER_MIN_SIGNALS_H
```

Yrkes Akademin
Vi hjälper dig att lyckas!

# Component "CAN Messages". part4

```
//source file, you can choose the extension ( .cc, .cxx, .cpp).
//for the example above: min_signals.cpp
#include "can_messages/min_signals.h"
#include <sstring>

CAN_min_signals::CAN_min_signals() {
    m_startMsgId = 0x100;
    m_temperatureGetMsgId = m_startMsgId + 2 ;
    m_temperatureSetMsgId = m_startMsgId + 2 + 1;
    m_humidityGetMsgId = m_startMsgId + 4;
    m_humiditySetMsgId = m_startMsgId + 4 + 1;
}

std::string CAN_min_signals::get_temperature() {
    std::stringstream sstream;
    sstream << "{\"ID\": " << m_temperatureGetMsgId
            << ", \"length\":0 "
            << ", \"value\": \"\" }";
    return sstream.str();
}

      // continued on the next page
```

Yrkes Akademin
Vi hjälper dig att lyckas!

# Component "CAN Messages". part5

```
std::string CAN_min_signals::set_temperature(float newValue) {
    std::stringstream sstream;
    sstream << "{\"ID\": " << m_temperatureSetMsgId
            << ", \"length\":10 "
            << ", \"value\": \"
            << newValue
            << \" }";
    return sstream.str();
}
std::string CAN_min_signals::get_humidity() {
    std::stringstream sstream;
    sstream << "{\"ID\": " << m_humidityGetMsgId
            << ", \"length\":0 "
            << ", \"value\": \"\" }";
    return sstream.str();
}
std::string CAN_min_signals::set_humidity(uint8_t newValue) {
    std::stringstream sstream;
    sstream << "{\"ID\": " << m_humiditySetMsgId
            << ", \"length\":8 "
            << ", \"value\": \"
            << newValue
            << \" }";
    return sstream.str();
}
```

Yrkes Akademin
Vi hjälper dig att lyckas!

## Important Note

The generated files should be placed in a specified directory with the following structure:

```
<specified folder for output>
|- include
|    |- can_messages
|          |- <name of JSON document>.h
|- src
    |- <name of JSON document>.cpp

P.S. the extension for the source file might be different (.cpp , .cxx, .cc )
P.P.S. the id for each signal should follow by pattern : startId + signal Number + (get/set flag).
```

## Requirements and Tests

- The component should use a correct formed JSON file as the input.
- The component should generate both header and source files.
- The component should be written on Python language.
- The component might have UT ( unit tests) written with using pytest.
- The component might have a component test written again on Python script ( as a separate file), which provides predefined JSON file as an input and checks generated files with predefined examples.

Yrkes Akademin
Vi hjälper dig att lyckas!

# Main Component

The main component is dependending on other components: "CAN Messages" and "File IO". It is fine to start the development process without them, but later adaptation might be required. After some discussions with a customer (it was tough!), we have concluded, that the main component should perform the following:

- the component should read a file, where each line contains only 1 command. For example:

```
set volume 100
get humidity
get volume
set stop_signal_light enabled
```

- For each of a command, the component should look for a list of messages (from component CAN Messages) to resolve message ID, name and etc.
- the component should generate an output JSON file with correspond CAN messages. Each messages on a separate line, for an abstract example:

```
{
    [
        {"ID": 0x03, "length":8, "value": 100 },
        {"ID": 0x06, "length":0, "value": "" },
        {"ID": 0x02, "length":0, "value": "" },
        {"ID": 0x11, "length":1, "value": 1 },
    ]
}
```

- the component should have a component test written on Python with predefined fixed input file and compare the output with predefined text.
- the component should have an Unit test used GoogleTest to test read and write functionality.

Yrkes Akademin

Vi hjälper dig att lyckas!