Preparado - *Prepared (also subject responsible if other)*		Nr – *No.*		
Mikel Fernández Obregón		[SPE-15:6102]		
Responsable/Aprobado - *Doc respons/Approved*	Contr – *Checked*	Datum - *Date*	Rev.	File
Miguel López López		2015-05-04	PA1	owa3x, FMS, SAE J1939, Can bus

OWASYS FMS LIBRARY

Abstract

The Owasys FMS library is a pure C library implementing the SAE J1939 standard to be used on Owasys Owa3x units.

1. INTRODUCTION

The Owasys FMS library is pure C library implementing the SAE J1939 standard to be used on Owasys Owa3x units. Its main features are:

- Simple interface for quick integration on any existing application
- Implementation of all relevant SAE J1939 messages
- Transparent to the main business logic.
- No external dependencies

It is specially designed to work seamlessly with the official Owasys demo application owa3x_AN25.

1.1. Requirements

This library is only meant to work with the **Owasys Owa3x family** of devices, hardware R4 or higher, and it depends on the following Owasys proprietary libraries to work :

- RTU library
- IO library

Vehicles: Bus and trucks and any vehicle model with CAN bus interface implementing the SAE J1939 standard FMS protocol.

2. SAE J1939 DATA PROVIDED

*The sources of the data are the internal ECUs of the vehicle. The Owa3x acts only as a passive reader of the CAN bus through the FMS J1939 interface and **never interacts nor writes on the CAN bus**.*

Depending on the vehicle's manufacturer and specific model some of the information listed here may not be available even when it is part of the standard.

2.1. Common Bus and truck data

The following information is read by the library, both for buses, trucks and other vehicles:

- Bus FMS standard version supported by this interface
- Truck FMS standard version supported by this interface
- Vehicle identification (VIN) (Disabled for performance reasons)
- Engine temperature: Main engine coolant temperature in ºC
- Ambient temperature: in ºC
- Movement flag
- Air Supply Pressure: Service Brake Air Pressure Circuit #1,Service Brake Air Pressure Circuit #2 (in kPa)

- Diesel Exhaust Fluid Tank 1 level: Ratio of volume of diesel exhaust fluid to the total volume of diesel exhaust fluid storage container
- FMS Tell Tale Status: The Tell Tale Status information is derived from information displayed to the driver's dashboard.[1]
- Engine current RPM
- Actual engine: % of torque
- Engine speed in km/h
- Total engine hours[2]
- High Resolution Vehicle Distance: Odometer of the vehicle, in meters
- Fuel level: current % of the tank, fuel level in litres and high resolution fuel level (in l also but with 4 decimal positions)
- Fuel economy: Amount of fuel consumed by engine per unit of time (l/h) and Instantaneous Fuel Economy: Current fuel economy at current vehicle velocity (km/l)
- Pedal information: brake, clutch, parking brake and cruise control.
- Accelerator Pedal Position % (100% is max)
- Gear information: Selected gear and current gear

2.2. Bus specific information

- Door control information
- ...

2.3. Truck specific information

- Engine load: % engine load at current speed
- Service distance: distance that can be travelled before service is needed. In km
- ...

2.4. Tachograph data

All tachograph-related information provided by the FMS interface will be published by the library. This information is provided independently from the tacograph model. This information is not legally valid.

- Tachograph engine speed (km/h)
- Direction indicator
- Tachograph performance
- Handling information
- System events
- Vehicle motion

1 The interpretation os the status is manufacturer dependant and might be different.
2 Engine running tatus and manufacturer dependant.

- Overspeed flag
- Information about both **drivers**: Card present, time related state and working state. Driver identifications.

2.5. Events

The event are asynchronous and are evaluated by the own Owasys library based on the data read from the CAN bus. They are published to be consumed by the main application. Start and end of each event is reported.

- No connection: No communication on the CAN bus
- Not implemented: A message that is not implemented detected on the CAN bus (Disabled by default for performance reasons). It is used to detected manufacturer specific dependant extensions to the protocol.
- Generic error
- Ignition
- Idling: Time with no movement and ignition is on. The minimum time of idling engine to raise the event is configurable.
- For this event duration, fuel consumption average rate and total fuel consumed (normal and high resolution modes) are calculated.
- Movement with no throttle: Minimum duration threshold configurable
- Engine temperature over limit: Temperature threshold configurable
- Engine RPM over limit: RPM threshold configurable
- Accelerator over limit: Acclerator threshold configurable

These events are implemented in the function `int fms_EventCheck()` inside "fms.c" file and defined as an enumeration in "fms.h" and so are easily extensible.

3. INTEGRATION

The FMS code once initialized runs in a different thread detached from the main application. Given the potential speed of the CAN bus FMS interface this dedicated thread is initialized only for reading from the CAN bus and writing into memory the FMS values (`int fms_Writer(void*)`); while in the main application the `int fms_Run(void)` function should be called frequently to check the status and process the FMS events.

The FMS library uses an internal state machine with the following states:

- FMS_OFF,
- FMS_VEHICLE_STOPPED,
- FMS_VEHICLE_IDLING,
- FMS_VEHICLE_MOVING

Preparado - *Prepared (also subject responsible if other)*	Datum - *Date*	Rev.	Nr – *No.*
Mikel Fernández Obregón	2015-05-04	PA1	[SPE-15:6102]

For tracking applications these states are the most appropriate. Should you need to modify the behaviour or the procesing of events please check `int fms_Run(void)` function. This function should be called periodically to check the current state of the FMS bus and process events.

This implementation allows for synchronous/traditional or asynchronous (using external libev for example) architectures to be used in the main application.

3.1. Usage

Include the **fms.h and fms.c** files into your project. The FMS library is used calling the methods of the following `fms_t`structure from your main application code:

```
///@see lines 145 and 276 of @file fms.h
fms_t Fms = {
   &fms_Start,
   &fms_Stop,
   &fms_End,
   &fms_GetInfo,
   &fms_Run,
   0 };
```

Apart from this, declare a `fms_data_t` stucture to store the values read from the FMS CAN bus.

1. First initialize passing your configuration in a `fms_conf_t` structure. Then attach a custom event handler:

   ```
   fms_data_t myCurrentFmsData;
   Fms.Init(myFmsConf);
   Fms.EventHandler = myCustomFMS_EventHandlerFunction;
   ```

   If you don't want to treat the FMS events, simply leave the event handler set to `NULL`.

2. Start the independent FMS thread:

   ```
   if ( Fms.Start() != NO_ERROR ){
     //log error
   }
   ```

3. Call periodically (i.e 1/2 seg) the `int fms_Run(void)` function and your custom event handler function to keep the FMS state (Off, moving, stopped and idling) and events properly updated and treated:

```
    Fms.GetInfo(&myCurrentFmsData);
 Fms.Run();
```

4. To stop the FMS features call the correspoding method:

```
    Fms.End();
 Fms.EventHandler = NULL;
```

3.2. Customization

The code can be customized and extended dependeing of each application requirements, specially the state machine and the events without altering the reader thread or your main application.

Should you have more uncommon J1939 data or any manufacturer specific-messages please contact us at customer_support@owasys.com