VMSENS Software Development Kit

User Manual

VMSENS Inertial Technologies (BeiJing) CO.LTD
TEL: +86-10-85376216
Email: contact@vmsens.com
Internet: www.vmsens.com

Terms, abbreviations and references

Term	Description
Quaternion	A non-commutative extension of complex numbers. A unit length quaternion is a convenient parameterization of rotations.
Euler Angles	Representation of the spatial orientation of any frame of the space as a composition of rotations from a reference frame.

Abbreviation	Description
DOF	Degree of freedom
DSP	Digital signal process
MCU	Micro controller unit
IMU	Inertial Measurement Unit
AHRS	Attitude and heading reference system
SDK	Software development kit
MT	Motion tracking
EKF	Extended Kalman Filter
Dps	Degree per second
DCM	Direction cosine matrix

1 Introduction

This document provided many different ways of interfacing with your VMSENS devices (included VM-i, VM-OEM, VM-BT and other VMSENS provide devices). User can find a good choice on the system design for your own system design.

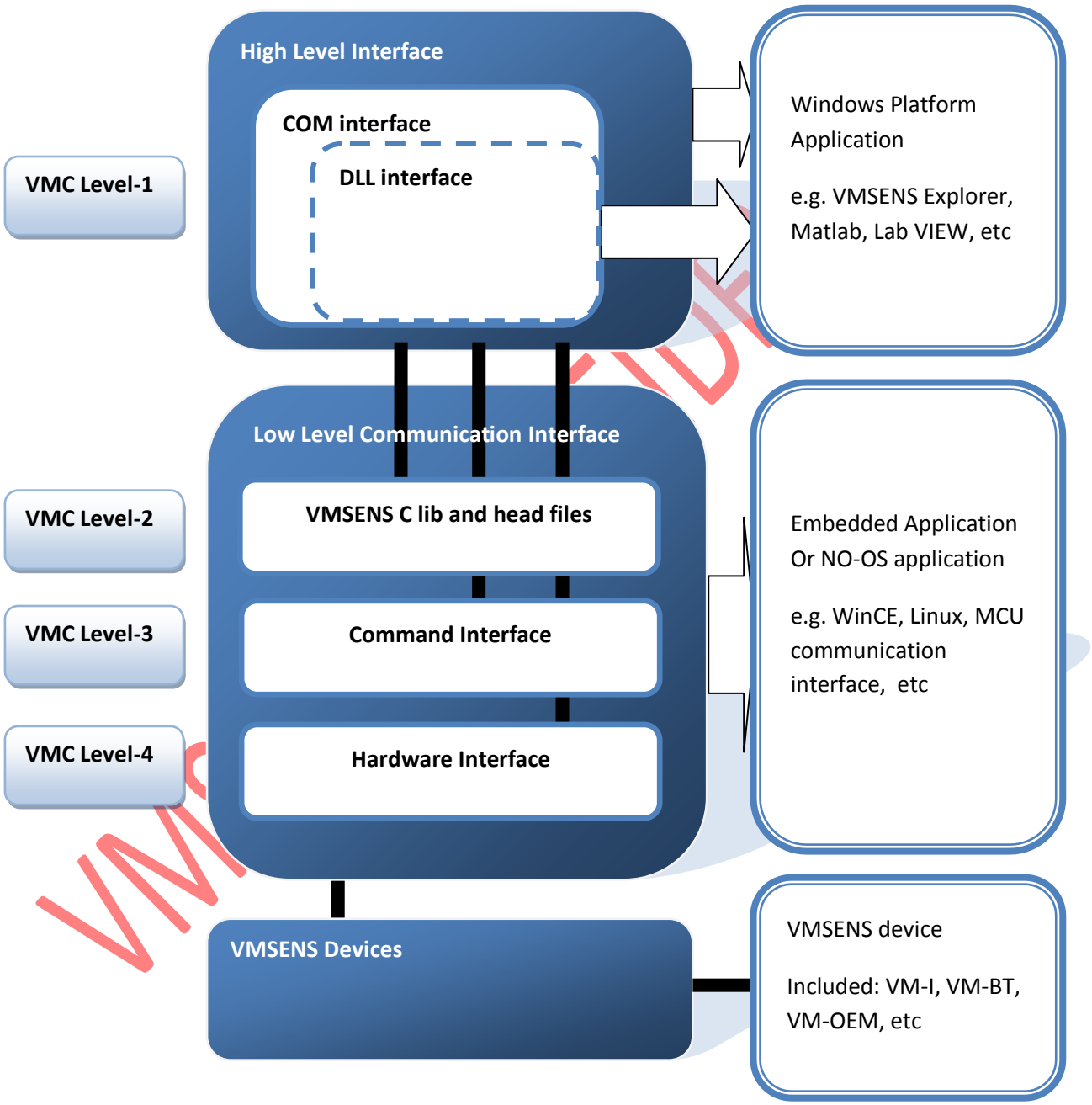
In this document we will discuss the VMSENS Explorer Communication software and its architecture. This software should be able to save a large amount of time in interfacing in a reliable way with the VMSENS devices. There are several different levels on which you can interface using the CMT. These options and how to implement and use the CMT are discussed in this document.

Low-level communication interfacing on the serial COM port (RS-232) can be use by users; VMSENS provide such interface for the user who want to full control and hard real-time of VMSENS devices. This is not discussed detail in this document; please refer to other documentation for the detail

The software development architecture is discussed in section 2,user can have a good concept on design a system base on VMSENS devices

1.1 SDK Software architecture

This section describes the VMSENS software development interface architecture, you can have a concept on VMSENS SDK architecture, and next chapter will present the top level interface.



1.2 Interface through COM-object API

A COM-object is a DLL that is registered on the operating system (Windows), so if properly installed you can access the functions of the COM-object in all Windows applications that support COM.

If you want to develop a Windows software application that uses the VMSENS devices, you can consider using the COM-object API . In particular if you are developing your application within another application such as MATLAB, LabVIEW, Excel, etc. the COM-object is the preferred interface. The COM-object provides easy to use function calls to obtain data from the sensor or to change settings.

The COM-object handle all of hardware communication interfacing and it is an easy way to get real-time performance(not in hardware real-time). Typically this method is preferred when you want to access the VMSENS devices directly in application software such as MATLAB, LabVIEW, Excel (Visual Basic), etc.

1.3 Interface through DLL API

VMSENS provided a standard C dynamic linked library interface method; but this method of interfacing (the function calls) is similar to the COM object, by using a DLL API, there is no need to register the DLL on the operating system, the functions are accessed directly in your source code by linking the DLL.

The programmer who want to develop Windows application software on VMSENS Devices by using a programming language (such as C, C++,C#, etc.) can use a DLL API. you will find that the DLL interface provides easier support for structured data, and this is also the recommended method.

1.4 Communication over command interface

For the user who want to full-control and hard real-time of VMSENS devices, VMSESN provide a low-level communication by RS-232 (UART) interface.

The VMSENS's low power processor performs all the calculations/calibration, you just retrieve the data from the serial port by using the VMSENS communication protocol

VMSENS provide a C lib / head files and command Interface for the user who want to use Low-level communication with VMSENS Device

Please refer to the VMSENS Low-level communication documentation and the other documentation for more information on this topic.

2 DLL / LIB API documentation

2.1 Data structures and defines:

- **Structure**

```
struct DeviceConfiguration {
      uint32_t      m_deviceId;              //device ID
      uint16_t      m_samplingFreq;          //output data polling rate
      uint16_t      m_outputMode;            //Output Mode configuration
      uint32_t      m_outputSettings;        //Output setting configuration
      uint8_t       m_fwRevMajor/Minor; // fw verion id
      float         m_gyro_param;            //gyro parameter data for the algorithm
      float         m_accel_param;           //accelerator parameter data for algorithm
      //float   null                         //not used
};

struct VmsVector {
  double m_Data[3];
};

struct VmsCalData {
  VmsVector       m_acc,m_gyr,m_pos;
};

struct VmsQuat {
  double m_Data[4];
};

struct VmsEuler {
  double m_roll;
  double m_pitch;
  double m_yaw;
};

struct VmsMatrix {
  double m_Data[3][3];
};
```

- **Output Mode Bit Field defines**

```
#define VMS_OUTPUTMODE_SENSORDATA                0x0002
```

#define VMS_OUTPUTMODE_ORIENT 0x0004

- **Output Setting Bit Field defines**

#define VMS_OUTPUTSETTINGS_ORIENT_QUATERNION 0x00000000

#define VMS_OUTPUTSETTINGS_ORIENT_EULER 0x00000004

#define VMS_OUTPUTSETTINGS_ORIENT_MATRIX 0x00000008

#define VMS_OUTPUTSETTINGS_SENSORDATA_ACC 0x00000010

#define VMS_OUTPUTSETTINGS_SENSORDATA_GYR 0x00000020

#define VMS_OUTPUTSETTINGS_SENSORDATA_POS 0x00000040

#define VMS_OUTPUTSETTINGS_SENSORDATA_MASK 0x0000000C

2.2 Function Definitions:

2.2.1 Initialize and de-initialize functions

VMSENSE_API VmsRet VmsCreateInstance(

uint32_t* Instance

);

Description: Create a communication instance.

Parameters:

uint32_t* Instance	[IN]Handle to the communication instance

VMSENSE_API VmsRet VmsClose (

Const uint32_t instance

);

Description: Close the communication instance.

Parameters:

Const int32_t instance	[IN]Handle to the communication instance

VMSENSE_API VmsRet VmsOpenPort(

const int32_t instance,

const uint16_t portNumber,

const uint32_t baudRate = CMT_DEFAULT_BAUD_RATE

);

Description: Open a COM port and associate it with a communication instance.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
const uint16_t portNumber	[IN]COM port number for connecting
const uint32_t baudRate	[IN]baudrate,fixed to 115200 at the present

VMSENSE_API VmsRet VmsClosePort (

const int32_t instance,

const uint16_t portNr

);

Description: Close the com port.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
const uint16_t portNr	[IN]Com port number

2.2.2 State switching functions

VMSENSE_API VmsRet VmsGotoConfig (

const int32_t instance

);

Description:

Parameters:

const int32_t instance	[IN]Handle to the communication instance

VMSENSE_API VmsRet VmsGotoMeasurement (

const int32_t instance

);

Description:

Parameters:

const int32_t instance	[IN]Handle to the communication instance

VMSENSE_API VmsRet VmsSetCurBid(

const int32_t instance,

uint8_t bid

);

Description: Set the module ID to communicate with.

(For multi wireless module communications)

Parameters:

const int32_t instance	[IN]Handle to the communication instance
uint8_t bid	[IN]Current bid to be set

2.2.3 Functions for getting/setting parameters

VMSENSE_API VmsRet VmsGetConfiguration (

const int32_t instance,

DeviceConfiguration* configuration

);

Description: Get the device configuration.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
DeviceConfiguration* configuration	[OUT]The device configuration structure (Refer to 2.1)

VMSENSE_API VmsRet VmsGetDeviceMode (

const int32_t instance,

VmsOutputMode* mode,

VmsOutputSettings* settings,

);

Description: Get the output mode and output settings.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsOutputMode* mode	[OUT]Buffer holding the output mode (Refer to 2.1)
VmsOutputSettings* settings	[OUT]Buffer holding the output setting(Refer to 2.1)

VMSENSE_API VmsRet VmsSetDeviceMode (

const int32_t instance,

const VmsOutputMode mode,

const VmsOutputSettings settings,

);

Description: Setting the device output mode, which include the output mode, output setting.

(The bit field definition please refer to 2.1)

Parameters:

const int32_t instance	[IN]Handle to the communication instance
const VmsOutputMode mode	[IN] The output mode setting
const VmsOutputSettings settings	[IN]The output setting
const uint16_t frequency	[IN]The output frequency

VMSENSE_API VmsRet VmsSetSampleFrequency (

const int32_t instance,

uint16_t freq,

);

Description: Set the update frequency.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
uint16_t*freq	[IN]The frequency parameter to set

VMSENSE_API VmsRet VmsSetGyroRateRange (

const int32_t instance,

uint16_t range,

);

Description: Set the gyro rate range.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
uint16_t range	[IN]The gyro range parameter to set (0:500 dps 1:1000 dps 2:2000 dps)

VMSENSE_API VmsRet VmsSetProcessVar(

const int32_t instance,

float var,

const VmsDeviceId dev=0

);

Description: Set the algorithm parameter for gyro.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
float var	[IN]The covariance to be set to the device
const VmsDeviceId dev	[IN]The device ID

VMSENSE_API VmsRet VmsSetAccVar (

const int32_t instance,

float var,

);

Description: Set the algorithm parameter for accelerometer.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
float var	[IN]The covariance to be set to the device

2.2.4 Data handling functions

VMSENSE_API VmsRet VmsRequestDataPacket (

const int32_t instance

);

Description: Request a data packet.

Parameters:

const int32_t instance	[IN]Handle to the communication instance

const VmsDeviceId dev	[IN]The device ID

VMSENSE_API VmsRet VmsWaitForDataPacket (

const int32_t instance

);

Description: Wait for a data packet.

Parameters:

const int32_t instance	[IN]Handle to the communication instance

VMSENSE_API VmsRet VmsDataGetCalAcc(

const int32_t instance,

VmsVector* data,

);

Description:

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsVector* data	[OUT]Pointer to data buffer(accelerator data)

VMSENSE_API VmsRet VmsDataGetCalGyr(

const int32_t instance,

VmsVector* data,

);

Description: Get the Gyro sensor data for from the recent received data packet.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsVector* data	[OUT]Pointer to data buffer(gyro data)

VMSENSE_API VmsRet VmsDataGetCalPos (

const int32_t instance,

VmsVector* data,

);

Description: Get the Calibrated magnetic sensor data from the recent received data packet.

Parameters:

Return Value:

const int32_t instance	[IN]Handle to the communication instance
VmsVector* data	[OUT]Pointer to Data buffer (position data)

VMSENSE_API VmsRet VmsDataGetCalData (

const int32_t instance,

VmsCalData* data,

);

Description: Get the Calibrated sensor data from the recent received data packet. (Include rate of turn, acceleration along three axis and also the normalized magnetic data)

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsCalData* data	[OUT]Pointer to Data buffer

VMSENSE_API VmsRet VmsDataGetOriQuat(

const int32_t instance,

VmsQuat* data,

);

Description: Get the Orientation data in the form of unit quaternion from the recent received data packet.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsQuat* data	[OUT]Pointer to data buffer

VMSENSE_API VmsRet VmsDataGetOriEuler (

const int32_t instance,

VmsEuler* data,

);

Description: Get the Orientation data in the form of Euler angles from the recent received data packet.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsEuler* data	[OUT]Pointer to data buffer

VMSENSE_API VmsRet VmsDataGetOriMatrix (

const int32_t instance,

VmsMatrix* data,

);

Description: Get the Orientation data in the form of Directional Cosine Matrix from the recent received data packet.

Parameters:

const int32_t instance	[IN]Handle to the communication instance
VmsMatrix* data	[OUT]Pointer to data buffer

2.2.5 Misc functions:

VMSENSE_API VmsRet VmsGetDllVersion (

uint16_t* version

);

Description: Get current DLL version.

Parameters:

uint32_t* Instance	[OUT]buffer to hold the DLL version

VMSENSE_API VmsRet VmsWriteToFlash (

const int32_t instance,

);

Description: Write the current configurations to flash.

Parameters:

const int32_t instance	[IN]Handle to the communication instance

3 Customer Support

We are glad to help you with any questions you may have about VMSENS products, or about the use of the technology for your application. Please contact our customer support:

E-mail: support@vmsens.com
TEL: +86 10 85376216

To be able to help you, please mention your hardware model and software version in your email.

VMSENS Inertial Technologies (BeiJing) CO.,LTD
www.vmsens.com