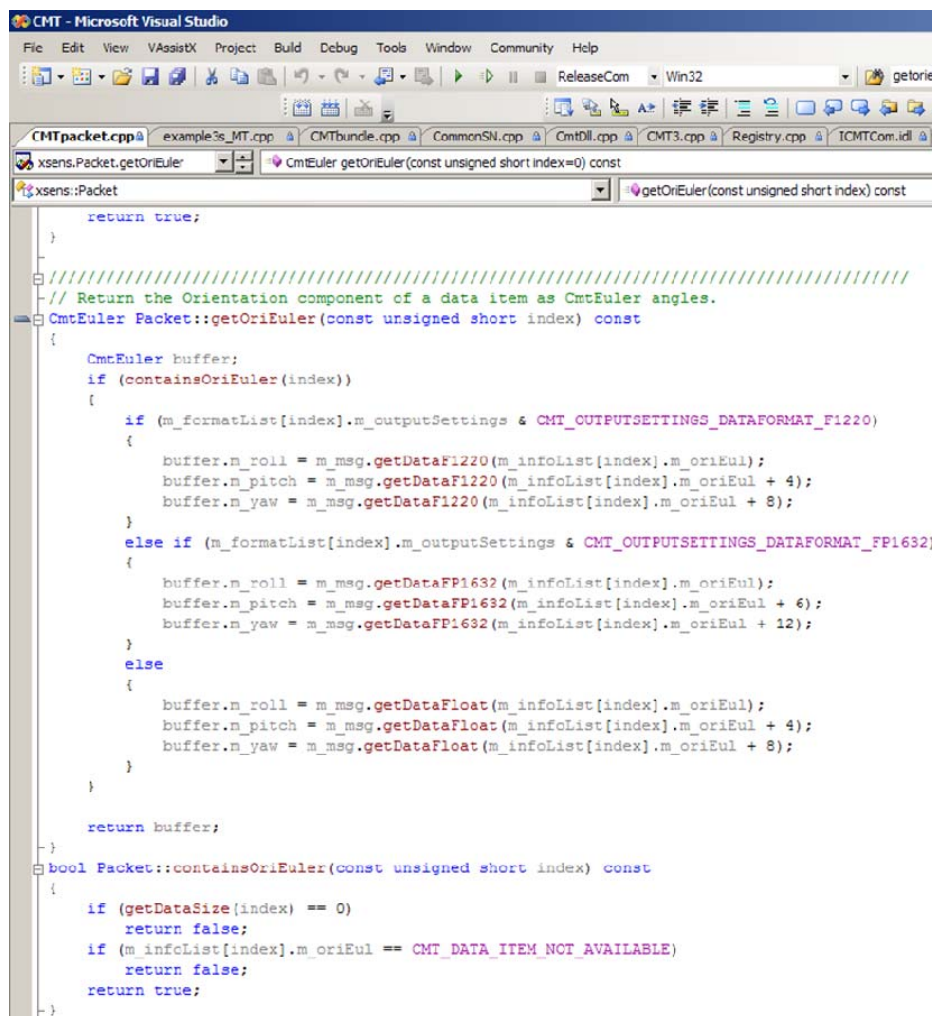


MT Software Development Kit Documentation

Document MT0200P, Revision K, 15 Oct 2010



```
CMT - Microsoft Visual Studio
File Edit View VAssistX Project Build Debug Tools Window Community Help
ReleaseCom Win32
CMTpacket.cpp example3s_MT.cpp CMTbundle.cpp CommonSN.cpp CmtDll.cpp CMT3.cpp Registry.cpp ICMTCom.idl
xsens.Packet.getOriEuler CmtEuler getOriEuler(const unsigned short index=0) const
xsens::Packet getOriEuler(const unsigned short index) const

return true;
}

// Return the Orientation component of a data item as CmtEuler angles.
CmtEuler Packet::getOriEuler(const unsigned short index) const
{
    CmtEuler buffer;
    if (containsOriEuler(index))
    {
        if (m_formatList[index].m_outputSettings & CMT_OUTPUTSETTINGS_DATAFORMAT_F1220)
        {
            buffer.m_roll = m_msg.getDataF1220(m_infoList[index].m_oriEul);
            buffer.m_pitch = m_msg.getDataF1220(m_infoList[index].m_oriEul + 4);
            buffer.m_yaw = m_msg.getDataF1220(m_infoList[index].m_oriEul + 8);
        }
        else if (m_formatList[index].m_outputSettings & CMT_OUTPUTSETTINGS_DATAFORMAT_FP1632)
        {
            buffer.m_roll = m_msg.getDataFP1632(m_infoList[index].m_oriEul);
            buffer.m_pitch = m_msg.getDataFP1632(m_infoList[index].m_oriEul + 6);
            buffer.m_yaw = m_msg.getDataFP1632(m_infoList[index].m_oriEul + 12);
        }
        else
        {
            buffer.m_roll = m_msg.getDataFloat(m_infoList[index].m_oriEul);
            buffer.m_pitch = m_msg.getDataFloat(m_infoList[index].m_oriEul + 4);
            buffer.m_yaw = m_msg.getDataFloat(m_infoList[index].m_oriEul + 8);
        }
    }

    return buffer;
}

bool Packet::containsOriEuler(const unsigned short index) const
{
    if (getDataSize(index) == 0)
        return false;
    if (m_infoList[index].m_oriEul == CMT_DATA_ITEM_NOT_AVAILABLE)
        return false;
    return true;
}
```

Xsens Technologies B.V.
phone +31 88 97367 00
fax +31 88 97367 01
email info@xsens.com
internet www.xsens.com



Revisions

Revision	Date	By	Changes
A	June 2, 2005	PRI	First version
B	June 20, 2005	SSM	Changed range of heading setting Updated the Xbus functions
C	August 8, 2005	SSM	Updated Xbus class chapter for class rev 1.1
D	February 1, 2006	PSL	Removed reference to Linux shared object, updated Linux support in example code using Xbus class. Added reference to Neverball game.
E	March 2, 2006	SSM	Changed name Xbus class to MTComm
F	February 21, 2007	PSL	Minor updates to section 2.2 Added support of Windows Mobile 5 (PocketPC) in MT Comm Class Added support of baud rates up to 921600 bps (921k6) Updated address information of Xsens
			NOTE: 1. MT SDK 3.0+ not compatible with MT9-B 2. MT SDK 3.0+ not compatible with MT's with firmware lower than 2.0
G	November 1, 2007	PSL	MT SDK 3.0 release RC1 Full re-write of SDK to CMT, compatible only with MT firmware 2.0 and higher.
H	April 1, 2008	JMU	Updates for 3.0.1 release
I	October 31, 2008	RZA	Updates for 3.1 release
J	May 27, 2009	MHA JMU	New corporate design Added 64 bit COM object interface information Added error codes
K	Oct 15, 2010	MHA	Added references to other documents

© 2010, Xsens Technologies B.V. All rights reserved. Information in this document is subject to change without notice. Xsens, MTi, MTi-G and MTx are registered trademarks or trademarks of Xsens Technologies B.V. and/or its parent, subsidiaries and/or affiliates in The Netherlands, the USA and/or other countries. All other trademarks are the property of their respective owners.

Table of Contents

1	INTRODUCTION	1
1.1.1	<i>Getting Started with the MT Manager.....</i>	1
1.1.2	<i>Interface through COM-object API.....</i>	1
1.1.3	<i>Interface through DLL API</i>	2
1.1.4	<i>Direct low-level communication with MT.....</i>	2
2	SOFTWARE ARCHITECTURE	3
2.1	CMT LEVEL 1 – HARDWARE INTERFACE	4
2.2	CMT LEVEL 2 – MESSAGE INTERFACE.....	4
2.3	CMT LEVEL 3 – DEVICE INTERFACE	4
2.4	CMT LEVEL 4 – APPLICATION INTERFACE	4
2.5	CMT LEVEL 4 API CONCEPTS.....	5
2.5.1	<i>Searching for connected MotionTrackers.....</i>	5
2.5.2	<i>Using the MotionTrackers.....</i>	5
2.5.3	<i>Addressing multiple MTs / XMs.....</i>	5
2.5.4	<i>File processing.....</i>	5
2.6	CMT FUNCTION OVERVIEW	6
3	CMT EXAMPLES	10
4	CMT COM OBJECT INTRODUCTION	11
4.1	OVERVIEW	11
4.2	IDISPATCH INTERFACE TO APPLICATIONS ON WINDOWS.....	12
5	XSENS RESULT VALUES	13
5.1	GENERIC	13
5.2	HARDWARE GENERATED ERRORS.....	13
5.3	CMT GENERATED ERRORS	13

1 Introduction

This document treats four different ways of interfacing with your MTi-G, MTi or MTx (MT for short). Direct low-level interfacing on the serial COM port (RS-232/422/485) can be accomplished by implementing the MT binary communication protocol, referring to the **MT Low-level communication protocol documentation**. This is not discussed in further detail in this document. Xsens' own reference implementation of this communication protocol (CMT) is discussed in this document.

NOTE: To understand the functionality of the functions described in this documentation, it is needed to have knowledge on the low level functions. Please refer to the MT Low-level communication protocol documentation and the doxygen documentation (C:\Program Files\Xsens\Documentation; doc_src, doc_dll, doc_com and doc_static).

In this document we will discuss the Xsens Communication MT (CMT) software and its architecture. This software should be able to save a large amount of time in interfacing in a reliable way with the MT. 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.

The architecture of CMT is discussed in section 2 (Software architecture). An example based on CMT is discussed briefly in section 3.

The reader is encouraged to refer to the HTML documentation for reference and to the source code itself for details. Also, example source code for C++ and MATLAB is provided to get started in an easy way.

1.1.1 Getting Started with the MT Manager

The easiest way to get started with your MT is to use the **MT Manager** software for Windows XP/Vista/7. This easy to use software with familiar Windows user interface allows you to:

- view 3D orientation in real-time
- view latitude, longitude, altitude plots in real time
- view inertial and magnetic sensor data in real time
- export log files to ASCII
- post-process raw data messages with different settings of the sensor fusion algorithm
- change and view various device settings and properties
- interactively "chat" with the MT through a terminal emulator.

The MT Manager is therefore an easy way to get to know and to demonstrate the capabilities of the MT.

Applies to: Windows PC platform
--

1.1.2 Interface through COM-object API

If you want to develop a Windows software application that uses the MT, you can consider using the COM-object API (XsensCMT.DLL). 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 XsensCMT.DLL COM-object provides easy to use function calls to obtain data from the sensor or to change settings.

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. The name of the function interface (IDispatch) is "MotionTracker.CMT".

The COM-object takes care of the hardware communication interfacing and it is an easy way to get (soft) real-time performance. Typically this is preferred when you want to access the MT's capabilities directly in application software such as MATLAB, LabVIEW, Excel (Visual Basic), etc. (examples included in MT SDK). Both polling and events based methods are supported.

1.1.3 Interface through DLL API

If you want to develop a Windows software application using a programming language (C, C++, etc.) that uses the MT you can consider using the DLL API. This method of interfacing (the function calls) is similar to the COM object, but is based on a standard C dynamic linked library interface method. So, 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 DLL to be used is the XsensCMT.DLL, so it is the same binary as the COM-object, but a different interface. If you program in C, C++ or other programming languages you will find that the DLL interface provides easier support for structured data, and this is therefore the recommended method.

Applies to: Windows PC platform

1.1.4 Direct low-level communication with MT

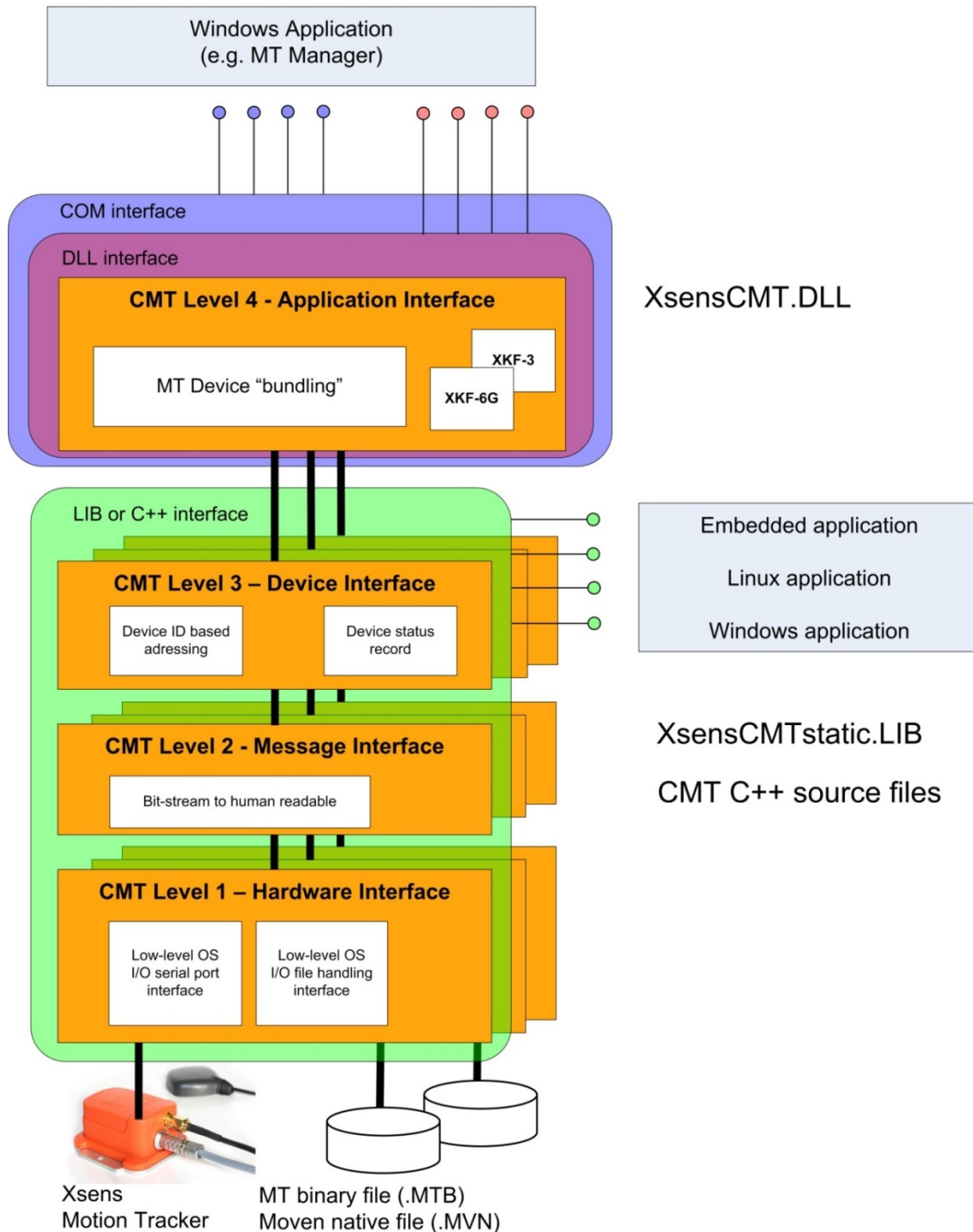
Direct interfacing with the MT (RS-232) is the natural choice if you are looking for full-control, maximum flexibility and/or have hard real-time performance requirements. The MT's low power embedded DSP performs all the calculations/calibration, you just retrieve the data from the serial port using the MT binary communication protocol using streaming (free-running) mode or polling (request) mode. Even this part is made easy for you by the inclusion of the source code (C++) of the Communication MT C++ classes (the CMT source code) in the MT SDK. Example C++ application code should get you quickly started on your development platform of choice. Example code that has been functionally checked and compiled on both Windows and Linux is included.

Applies to: Any (RT)OS or processor platform (C++)

→ Please refer to the **MT Low-level communication protocol documentation** and the doxygen documentation for more information on this topic.

2 Software architecture

This section describes the concept of the CMT interfaces at each level.



2.1 CMT Level 1 – Hardware interface

The functions at this level give access to the low-level IO communications. These functions are platform (operating system and processor) dependent so they may have to be modified for embedded applications. The purpose of separating this level from the other CMT levels is for embedded device programmers to be able to supply their own modified CMT Level 1 code and then recompile the library up to CMT Level 3 so they can use the high level functionality immediately.

CMT Level 1 is available to developers as part of the Windows static library and as source code.

2.2 CMT Level 2 – Message interface

The CMT Level 2 Message interface assembles the MT communication protocol messages using the L1 functions that return raw message data. If the checksum is correct the complete message is transmitted to the next CMT Level. CMT Level 2 has its own timeout(s) that are independent of Level 1, except that provisions are made that they will always be longer than the Level 1 timeouts.

The CMT Level 2 interface does not use any specific addressing method other than that it introduces and uses the MT Communication protocol Message structure.

Normally, there is no need to directly use/interface to or to modify CMT Level 2.

CMT Level 2 is available to developers as part of the Windows static library and as source code.

2.3 CMT Level 3 – Device interface

The CMT Level 3 implements following capabilities:

- Keep track of the MT device configurations for a single Master MT Device (communication port or file)
- This level implements all MT and XM messages. For example gotoConfig, setDeviceMode, getProductCode, etc.
- MT Data is retrieved in a special Packet structure that interprets and translates the contents of the data for easy data access.
- This level also has logging functionality, which allows you to log all received messages automatically and later read the file.
- Devices are addressed by DeviceID and not by serial port number.

CMT Level 3 is available to developers as the highest level component of the static library and as source code.

2.4 CMT Level 4 – Application interface

The CMT Level 4 implements a great deal of additional functionality for optimal ease of use and interfacing to multiple MT devices simultaneously. It provides a wrapper for all lower levels and exports the CMT Level 3 functionality as well. CMT Level 4 implements automatic handling of multiple connected devices and automatic data-queuing (buffering). Basically, CMT Level 4 relieves a lot of the implementation details of handling with MTs and XbusMasters from the application programmer. Level 4 allows automatic logging to a file, bundled data reception from multiple MT's, automatic (software) synchronized startup, etc. The handling of incoming MT communication protocol messages and certain device requests is performed in a separate thread and most measurement-mode messages can be sent and received without worrying about disturbing the incoming data.

CMT Level 4 is available to developers as a Windows dynamic link library (DLL) and as a COM object and requires a multi-threaded environment to function. The binary is the same for both interfaces, only different interfaces are used.

CMT Level 4 also supports software-based conversion of data by implementing the Xsens Kalman Filters (XKF) to emulate direct device output:

- raw data to calibrated data
- calibrated data to orientation data (only when originally converted from raw data)
- calibrated data and GPS PVT¹ data to orientation and position data (only when originally converted from raw data)
- conversion from one orientation mode to another (quaternion to euler)

In particular this can be useful for post-processing of logged binary data using different settings (e.g. XKF Scenarios) or non-casual processing.

2.5 CMT Level 4 API concepts

2.5.1 Searching for connected MotionTrackers

At CMT Level 4, instead of using the COM port number, the DeviceID of the sensors can be used. This leads to true transparency of the XbusMaster. It is possible to scan for available Xsens MT devices with a single function call. This will also detect the baud rate at which the devices are connected.

NOTE: Scanning all com ports can be a long process because of installed Bluetooth (and other) ports, which can take a long time to open, whether they are enabled or disabled.

2.5.2 Using the MotionTrackers

When connected MT's have been found and their ports opened with the `cmtOpenPort` function, the DeviceIDs returned by `cmtGetDeviceId()` can be used to individually address each Motion Tracker. It can be seen as a handle, similar to a file or window handle.

With this unique handle the settings of the MT can be modified or data can be requested. The functions translate the handle to the appropriate COM port number and BusID in the case of an Xbus Master.

2.5.3 Addressing multiple MTs / XMs

A user would frequently use a function that will be used for multiple MTs / XMs. For example a `cmtSetDeviceMode` will often be sent to all connected units.

To use multiple ports in the same way, the CMT Level 4 interface is required. This level allows a user to connect to multiple ports and use the special DeviceID **CMT_DID_BROADCAST** to send configuration messages to all connected devices. Some messages can also be broadcast during measurement mode. See the Low Level documentation for more information.

2.5.4 File processing

It is possible to log all messages coming from the connected MT's as well as opening one of these log files and reading the messages from it as if an MT were connected. A file containing MT binary communication protocol

¹ GPS PVT data: Position, Velocity Time data directly from the GPS board, including barometric pressure. GPS PVT data has not been processed in XKF

messages normally has extension .MTB. A single file will contain all data of a single port. The log file will first write some sensor meta-data that CMT can use when reading the file and then it will simply log all binary data as it comes in from the port. All data in the file is addressable by DeviceID as if it came from a live connected MT. Of course, settings can no more be modified when operating on a file.

NOTE: Each CMT instance can handle reading *either* from a port *or* reading from a file.

2.6 CMT Function overview

CMT contains groups of functions which can be performed under Config mode (*cmtGotoConfig*), Measurement mode (*cmtGotoMeasurement*) or any mode. Look through the accompanying HTML documentation for clarification on this. Many of these functions take the instance number and DeviceID as common parameters. This informs CMT of which CMT object to use and of course the deviceid is bound to a specific Xsens device. The list given here is *not* complete. See the accompanying HTML documentation for a complete list of available functions.

- **Initialise & deinitialise** – use these functions to initialise CMT for Device or File handling.
 - **cmtCreateInstance** – Create an instance of the CMT object. Each instance can handle one or more files otherwise one or more COM ports, but not both. The `serialNumber` parameter should contain a valid serial number as supplied upon purchase of the SDK. For testing “demo1” can be used² as a serial number.
 - **cmtDestroyInstance** - Destroys the given CMT object, freeing up its instance number for use by another process. It will close all files and ports associated with it. This function takes in the instance number of the CMT object in question and returns an Xsens result value.
 - **cmtClose** – Close all devices connected to the given instance.
 - **cmtOpenPort** – Opens the specified COM port.
 - **cmtOpenLogFile** – Opens a log file.
 - **cmtScanPorts** – Scan for connectable ports for Xsens devices.
- **Configuration Functions Get & Set** – retrieve and set configuration settings of an **Xsens device**. These functions specifically address an Xsens device by deviceid.

Get ()

- **cmtGetBatteryLevel** – Get the Battery life of the Xbus master.
- **cmtGetBaudrate** – Get the baud rate currently in use.
- **cmtGetBluetoothState** – Check if Bluetooth is in use or not.
- **cmtGetConfiguration** – Get the complete device configuration.
- **cmtGetDeviceId** – Get the deviceid.
- **cmtGetDroppedPacketCount** – Get the number of packets dropped during measurement.
- **cmtGetMainDeviceCount**– Get the number of connected master devices or 0 if not connected.
- **cmtGetMtCount** – Get the number of MTs currently interfacing with a CMT object.
- **cmtGetMainDeviceId** – Get the deviceid from the main device associated with the CMT object.
- **cmtGetMtDeviceId** – Get the deviceid from the device at the supplied index, whilst skipping XM’s.
- **cmtGetDeviceMode** – Get the complete device output mode of a device.
- **cmtGetErrorMode** – Get the error mode in use.
- **cmtGetHeading** – Get the heading offset of a device. The range is -pi to +pi.

² Note that the demo user has a limited number of function calls that can be made. It is intended only for testing.

- **cmtGetLocationId** – Get the location ID of a device.
- **cmtGetMagneticDeclination** – Get the stored magnetic declination. The range is $-\pi$ to $+\pi$.
- **cmtGetPortNr** – Get the port that a specific device is connected to.
- **cmtGetSampleFrequency** – Get the sample frequency of a device.
- **cmtGetSerialBaudrate** – Get the baud rate that is reported for the serial communication of a port.
- **cmtGetSyncInSettings** – Get the current SyncIn settings of the device, i.e. mode, skip factor or offset.
- **cmtGetSyncOutSettings** – Get the current SyncOut settings of the device, i.e. mode, skip factor or offset.
- **cmtGetSyncMode** – Get the sync mode of the Xbus Master.
- **cmtGetXmDeviceId** – Get the Id of the XM at the given index in the IdList (skipping non-XM IDs) or 0 if not connected.
- **cmtGetXmOutputMode** – Get the dual-mode output settings of the XM.
- **cmtGetAvailableScenarios** – Get a list of the available scenarios. This function may return more scenarios than are present in the specified sensor, since the CMT software may define additional scenarios.
- **cmtGetScenario** – Get the scenario currently in use by the specified device.
- **cmtGetGravityMagnitude** – Get the currently used magnitude of the gravity vector.
- **cmtGetGravityMagnitude** – Get the currently used magnitude of the gravity vector.
- **cmtGetLatLonAlt** – Get the currently used latitude/longitude/altitude.
- **cmtGetObjectAlignmentMatrix** – Get the currently used object alignment matrix.
- **cmtGetProcessingFlags** – Get the currently used processing flags.
- **cmtGetTransmissionDelay** – Get the currently used transmission delay.

Set ()

- **cmtSetBaudrate** – Set the baud rate and reconnect at the new baud rate.
- **cmtSetBluetoothState** – Set the state of the bluetooth communication to on or off.
- **cmtSetBusPowerState** – Switch the XM bus power on or off.
- **cmtSetDeviceMode** – Set the complete device output mode of a device. It only updates if the settings are different to the current device mode settings.
- **cmtForceSetDeviceMode** – Set the complete device output mode of a device. It always writes the mode, settings and the sample frequency.
- **cmtSetErrorMode** – Set the error mode. The error mode defines how the device should deal with non-message related errors. By default, if a sample is missed, the sample counter is increased and no further action is required (ERRORMODE = 1).
- **cmtSetQueueMode** – Sets the queue mode (FIFO or LAST).
- **cmtSetHeading** – Set the heading of the given device. The valid range is $-\pi$ to $+\pi$.
- **cmtSetMagneticDeclination** – Set the stored magnetic declination (radians between $-\pi$ and π).
- **cmtSetSyncInSettings** – Set the inbound synchronization settings of a device.
- **cmtSetSyncOutSettings** – Set the outbound synchronization settings of a device.
- **cmtSetSyncMode** – Set the sync mode of a device.
- **cmtSetXmOutputMode** – Set the dual-mode output settings of the XM.
- **cmtSetXmPowerOff** – Switch off one or all connected XMs.
- **cmtSetScenario** – Specifies the scenario to use for processing data.
- **cmtSetGravityMagnitude** – Sets the magnitude of the gravity vector. By default it is 9.81, but it can be altered for special purposes such as space applications or near-free-fall experiments.
- **cmtSetScenario** – Specifies the scenario to use for processing data.
- **cmtSetLatLonAlt** – Specifies the currently used latitude/longitude/altitude.
- **cmtSetObjectAlignmentMatrix** – Specifies the currently used object alignment matrix.
- **cmtSetTransmissionDelay** – Sets the transmission delay.

- **cmtSetProcessingFlags** – Specifies the currently used processing flags.
- **cmtSetNoRotation** – Specifies the ‘no rotation’ duration.
- **Data Handling Functions** – These functions are valid under measurement mode.
 - **cmtGetNextDataBundle** – Retrieve a data message. Specify whether to get data in a FIFO or latest data manner.
 - **cmtDataGetRawData** – Get the Raw Data component of a data item.
 - **cmtDataContainsRawData** – Check if data item contains Raw Data.
 - **cmtDataGetRawGpsData** – Get the GPS PVT component of a data item.
 - **cmtDataContainsRawGpsData** – Check if data item contains GPS PVT Data.
 - **cmtDataGetRawPressureData** – Get the Raw Pressure Data.
 - **cmtDataContainsRawPressureData** – Check if data item contains Raw Pressure Data.
 - **cmtDataGetCalData** – Get the Calibrated Data component of a data item.
 - **cmtDataContainsCalData** – Check if data item contains Calibrated Data.
 - **cmtDataGetOriEuler** – Get the Orientation component of a data item as Euler angles.
 - **cmtDataContainsOriEuler** – Check if data item contains Euler Orientation data.
 - **cmtDataGetOriMatrix** – Get the Orientation component of a data item as an Orientation Matrix.
 - **cmtDataContainsOriMatrix** – Check if data item contains Matrix Orientation data.
 - **cmtDataGetPositionLLA** – Get the Position LLA component of a data item.
 - **cmtDataContainsPositionLLA** – Check if data item contains Position LLA data.
 - **cmtDataGetVelocity** – Get the Velocity NED/NWU component of a data item.
 - **cmtDataContainsVelocity** – Check if data item contains Velocity data.
 - **cmtDataGetSampleCounter** – Get the Sample Counter component of the packet.
 - **cmtDataContainsSampleCounter** – Check if the data bundle contains sample counter
 - **cmtDataGetRtc** – Return the RTC component of the packet.
 - **cmtDataGetAccG** – Get the XKF-3 Acc-G component of the packet.
 - **cmtGetGpsLeverArm** – Gets the currently used GPS lever arm.
 - **cmtGetGpsStatus** – Request the status of the GPS satellites in orbit.
- **File Handling Functions** – These functions are valid under measurement mode.
 - **cmtOpenLogFile** – Open a log file for input.
 - **cmtCloseLogFile** – Close an open log file.
 - **cmtCreateLogFile** – Create a log file for the given port.
 - **cmtCloseLogFileForPort** – Close an open log file that is associated with the given port.
 - **cmtGetLogFileReadPos** – Retrieve the current log file read position of the log file.
 - **cmtGetLogFileSize** – Retrieve the size of the log file.
 - **cmtResetLogFileReadPos** – Reset Log file read position.
 - **cmtSetLogMode** – Enable or disable logging data to a file.
- **Generic Functions**
 - **cmtGetDllVersion** – Return version of the DLL.
 - **cmtGetFirmwareRevision** – Retrieve the firmware revision of a device.
 - **cmtGetLastHwError** – Return the error code of the last hardware problem.
 - **cmtGetProductCode** – Retrieve the product code of a device.
 - **cmtGotoConfig** – Switch to Config mode.
 - **cmtGotoMeasurement** – Switch to Measurement mode.
 - **cmtReset** – Reset a device.
 - **cmtResetOrientation** – Perform an orientation reset on a device. The method to reset and the deviceId must be specified.
 - **cmtRestoreFactoryDefaults** – Completely restore the device to default factory settings.

- **cmtSendCustomMessage** – Send a custom message to the selected device.
- **cmtSetCallbackFunction** – Set one of the callback functions for an instance.
- **cmtSetSoftwareCalibration** – Switch software-based calibration on or off.
- **cmtSetSoftwareXkf3Filtering** – Switch software-based XKF-3 filtering on or off.
- **cmtSetSoftwareXkf6Filtering** – Switch software-based XKF-6 filtering on or off.
- **cmtIdsMtig / cmtIdsMtix / cmtIdsMtxXbus / cmtIdsXm** – Determine the MT type.

3 CMT Examples

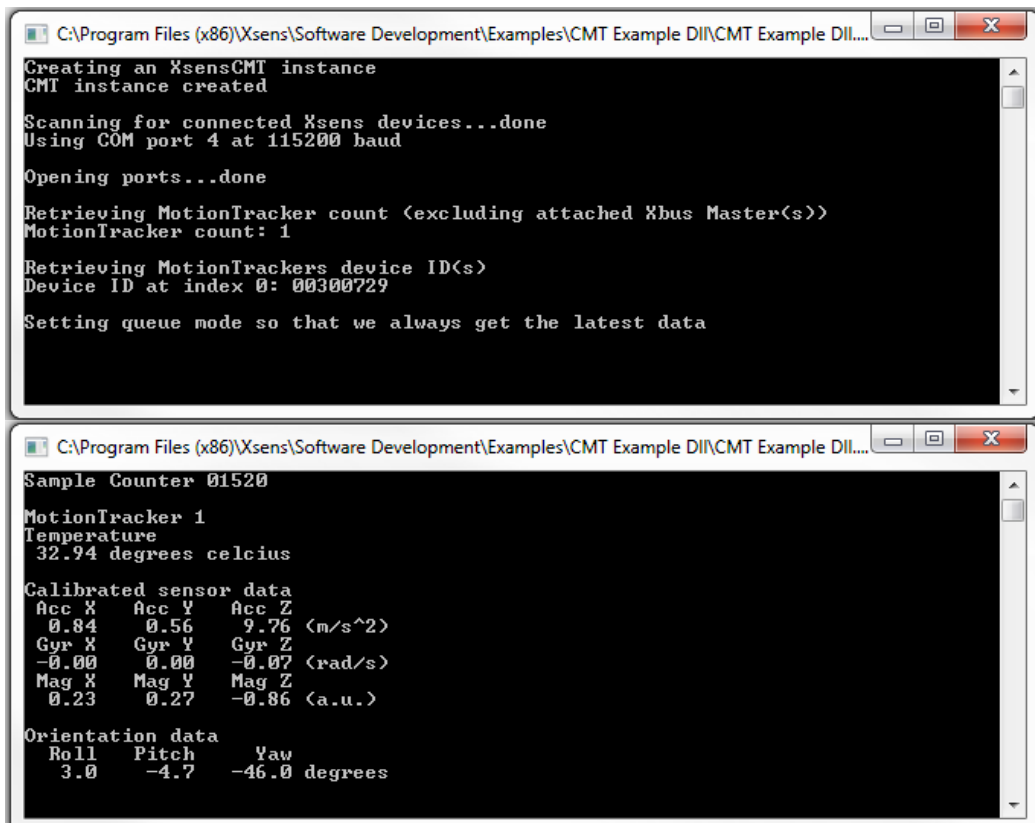
Examples of how to use CMT can be found in the examples directory, which is included in the SDK. These simple programs show how easy it is to setup and retrieve data from your Xsens device.

NOTE: When using a demo version of the CMT dll or COM object the output is displayed for only a few moments because the number of function calls is limited for the “demo1”user. You can recompile the example code using your own serial number. The Static library version has less functionality, but its use is not restricted in any way, so it will keep running.

The code structure of all examples is as follows:

- (C++ only) Includes, global variables, including the “XsensCMT” header file and a macro which test for errors and exits accordingly.
- (C++ only) An exit function which facilitates program termination and destruction of the CMT object.
- Main function:
 - Creates the CMT instance or CMT level 3 object.
 - Scans connectable ports for Xsens devices.
 - Opens the port.
 - Switches to config mode to perform common Get and Set functions.
 - User input to choose output mode and output settings.
 - Switches to measurement mode when ready for measurements.

Refer to the source code for details.



```

C:\Program Files (x86)\Xsens\Software Development\Examples\CMT Example DLL\CMT Example DLL...
Creating an XsensCMT instance
CMT instance created

Scanning for connected Xsens devices...done
Using COM port 4 at 115200 baud

Opening ports...done

Retrieving MotionTracker count (excluding attached Xbus Master(s))
MotionTracker count: 1

Retrieving MotionTrackers device ID(s)
Device ID at index 0: 00300729

Setting queue mode so that we always get the latest data

Sample Counter 01520
MotionTracker 1
Temperature
32.94 degrees celcius

Calibrated sensor data
Acc X    Acc Y    Acc Z    <m/s^2>
0.84     0.56     9.76
Gyr X    Gyr Y    Gyr Z    <rad/s>
-0.00    0.00     -0.07
Mag X    Mag Y    Mag Z    <a.u.>
0.23     0.27     -0.86

Orientation data
Roll    Pitch    Yaw
3.0     -4.7     -46.0 degrees
  
```

A CMT example program

Example code is available for C++ (DLL / Static lib), Excel (VBA / COM) and Matlab (COM).

4 CMT COM Object Introduction

While for C++ applications, the Static Library and Dll interfaces are preferable, CMT also has a COM interface. The basic functionality of the MTi-G, MTi and MTx, but also the MT Xbus (all supported device together called “MT”, for MotionTracker) are made available to software developers through this COM-object for the Windows platform. After installing the software component on your operating system you can access the MT device functionality with simple function-calls in your own (application) code.

The list of available COM function calls is available in the HTML documentation. The XsensCMT.dll contains both the DLL interface as the COM interface.

NOTE: Linux users are encouraged to use the CMT Level 1, 2 and 3 C++ classes described in the previous chapter. This provides a more powerful and flexible interface to your Motion Tracker hardware and is more applicable to application development on the Linux platform.

The integration with MATLAB, LabVIEW, and other application programs, is made possible through the COM-interfaces of those applications and is therefore only supported on the Windows platform. Of course you can also develop in C/C++ or other programming languages using the CMT, however, C/C++ users are encouraged to use the CMT C++ class or LIB and example code, as this provides a more powerful and flexible interface to your Motion Tracker hardware.

MS Windows 2000/XP are officially supported by the MT COM object API. Please visit <http://www.xsens.com/support> for other platforms or custom developments (e.g. PocketPC's).

4.1 Overview

The CMT software component is implemented in a COM object. When using the CMT Object, the user does not have to deal with subjects like data IO (serial communication). This functionality is all implemented in the COM object. All the user needs are the pointers to the functions of the CMT object (see Figure 1). The component itself is registered in the operating system and its interfaces can be accessed by instantiating an object “MotionTracker.CMT” on 32 bit systems and MotionTracker.CMT64 on 64 bit systems.

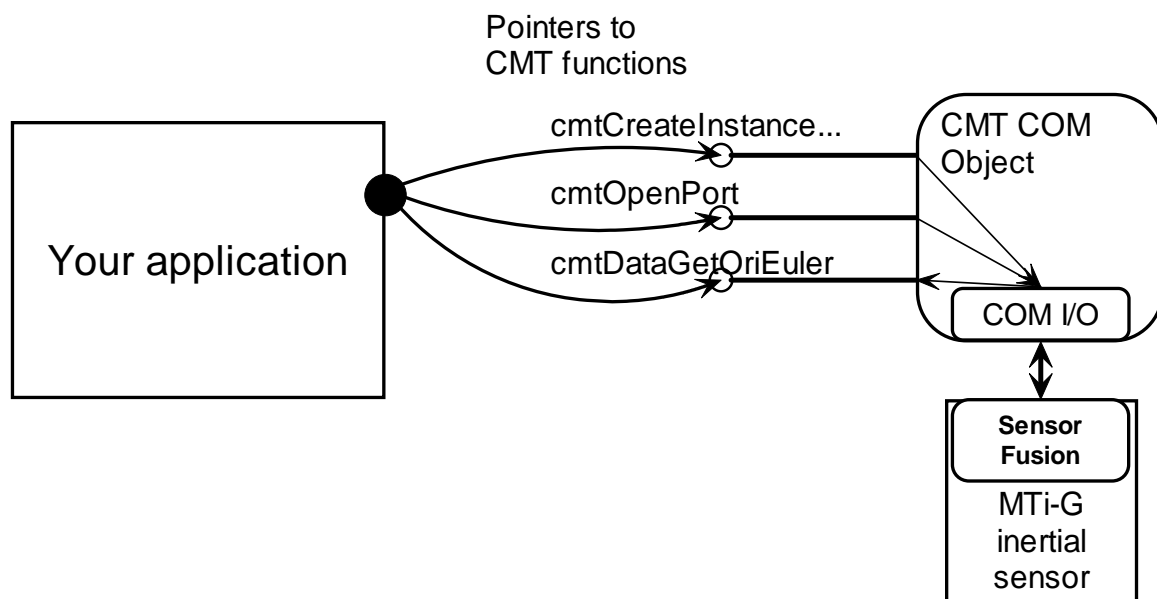


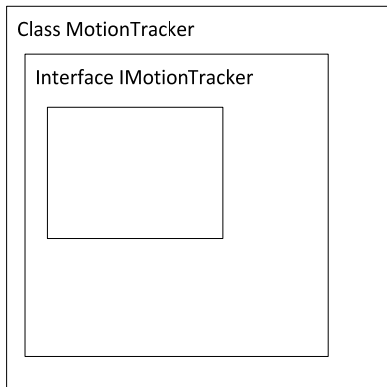
Figure 1 - Representation of data flow when using CMT COM Object with an MT

4.2 *IDispatch interface to applications on Windows*

The CMT COM object supports the IDispatch interface. This enables it to be used in application programs like Visual Basic, Matlab 7 or Labview 8.0 and higher that supports this interface. The interface IProvideClassInfo is also available for use in conjunction with IDispatch. With the use of IDispatch, some naming schemes can be confusing. Officially, the following are the correct names for all parts of the object.

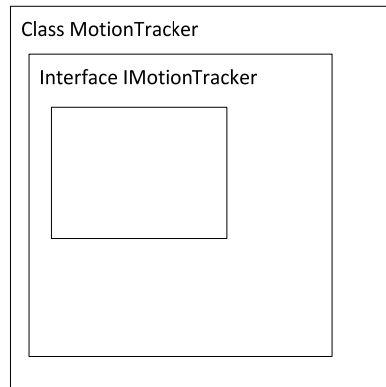
32 bit:

Library: IMotionTrackerCMT
 ProgID: MotionTracker.CMT(.1)
 TypeLib: MotionTracker 1.0 Type Library
 Version: 5.0



64 bit:

Library: IMotionTrackerCMT
 ProgID: MotionTracker.CMT64(.1)
 TypeLib: MotionTracker x64 Type Library
 Version: 5.0



Various programs may use different parameters for identification. For example MATLAB and VBA use the ProgID for identification, Labview uses the TypeLib and then creates a name based on the library name and an interface name.

5 Xsens Result Values

Most functions in the CMT return an XsensResultValue. This section explains the meaning of these values. The COM IDispatch interface does not return these values immediately. If COM reports an error, use the `cmtGetLastResult` function to retrieve the actual error that occurred.

5.1 Generic

Name	Value	Description
XRV_OK	0	Operation was performed successfully

5.2 Hardware generated errors

Name	Value	Description
XRV_NOBUS	1	No bus communication possible
XRV_BUSNOTREADY	2	InitBus and/or SetBID are not issued
XRV_INVALIDPERIOD	3	Period sent is invalid
XRV_INVALIDMSG	4	The message is invalid or not implemented
XRV_INITBUSFAIL1	16	A slave did not respond to WaitForSetBID
XRV_INITBUSFAIL2	17	An incorrect answer received after WaitForSetBID
XRV_INITBUSFAIL3	18	After four bus-scans still undetected Motion Trackers
XRV_SETBIDFAIL1	20	No reply to SetBID message during SetBID procedure
XRV_SETBIDFAIL2	21	Other than SetBIDAck received
XRV_MEASUREMENTFAIL1	24	Timer overflow - period too short to collect all data from Motion Trackers
XRV_MEASUREMENTFAIL2	25	Motion Tracker responds with other than SlaveData message
XRV_MEASUREMENTFAIL3	26	Total bytes of data of Motion Trackers including sample counter exceeds 255 bytes
XRV_MEASUREMENTFAIL4	27	Timer overflows during measurement
XRV_MEASUREMENTFAIL5	28	Timer overflows during measurement
XRV_MEASUREMENTFAIL6	29	No correct response from Motion Tracker during measurement
XRV_TIMEROVERFLOW	30	Timer overflows during measurement
XRV_BAUDRATEINVALID	32	Baud rate does not comply with valid range
XRV_PARAMINVALID	33	An invalid parameter is supplied
XRV_INVALIDPARAM	33	An invalid parameter is supplied
XRV_MEASUREMENTFAIL7	35	TX PC Buffer is full
XRV_MEASUREMENTFAIL8	36	TX PC Buffer overflow, cannot fit full message
XRV_DEVICEERROR	40	The device generated an error, try updating the firmware

5.3 CMT generated errors

Name	Value	Description
XRV_ERROR	256	A generic error occurred
XRV_NOTIMPLEMENTED	257	Operation not implemented in this version (yet)
XRV_TIMEOUT	258	A timeout occurred
XRV_TIMEOUTNODATA	259	Operation aborted because of no data read
XRV_CHECKSUMFAULT	260	Checksum fault occurred
XRV_OUTOFMEMORY	261	No internal memory available

XRV_NOTFOUND	262	The requested item was not found
XRV_UNEXPECTEDMSG	263	Unexpected message received (e.g. no acknowledge message received)
XRV_INVALIDID	264	Invalid id supplied
XRV_INVALIDOPERATION	265	Operation is invalid at this point
XRV_INSUFFICIENTSPACE	266	Insufficient buffer space available
XRV_INPUTCANNOTBEOPENED	267	The specified i/o device can not be opened
XRV_OUTPUTCANNOTBEOPENED	268	The specified i/o device can not be opened
XRV_ALREADYOPEN	269	An I/O device is already opened with this object
XRV_ENDOFFILE	270	End of file is reached
XRV_COULDNOTREADSETTINGS	271	A required settings file could not be opened or is missing some data
XRV_NODATA	272	No data is available
XRV_READONLY	273	Tried to change a read-only value
XRV_NULLPTR	274	Tried to supply a NULL value where it is not allowed
XRV_INSUFFICIENTDATA	275	Insufficient data was supplied to a function
XRV_BUSY	276	Busy processing, try again later
XRV_INVALIDINSTANCE	277	Invalid instance called
XRV_DATACORRUPT	278	A trusted data stream proves to contain corrupted data
XRV_READINITFAILED	279	Failure during read of settings
XRV_NOXMFOUND	280	Could not find any MVN-compatible hardware
XRV_ONLYONEXMFOUND	281	Found only one responding Xbus Master
XRV_MTCOUNTZERO	282	No sensors found
XRV_MTLOCATIONINVALID	283	One or more sensors are not where they were expected
XRV_INSUFFICIENTMTS	284	Not enough sensors were found
XRV_INITFUSIONFAILED	285	Failure during initialization of Fusion Engine
XRV_OTHER	286	Something else was received than was requested
XRV_NOFILEOPEN	287	No file opened for reading/writing
XRV_NOPORTOPEN	288	No serial port opened for reading/writing
XRV_NOFILEORPORTOPEN	289	No file or serial port opened for reading/writing
XRV_PORTNOTFOUND	290	A required port could not be found
XRV_INITPORTFAILED	291	The low-level port handler failed to initialize
XRV_CALIBRATIONFAILED	292	A calibration routine failed
XRV_CONFIGCHECKFAIL	293	The in-config check of the device failed
XRV_ALREADYDONE	294	The operation is once only and has already been performed
XRV_SYNC_SINGLE_SLAVE	295	The single connected device is configured as a slave
XRV_SYNC_SECOND_MASTER	296	More than one master was detected
XRV_SYNC_NO_SYNC	297	A device was detected that was neither master nor slave
XRV_SYNC_NO_MASTER	298	No master detected
XRV_SYNC_DATA_MISSING	299	A device is not sending enough data
XRV_VERSION_TOO_LOW	300	The version of the object is too low for the requested operation
XRV_VERSION_PROBLEM	301	The object has an unrecognized version, so it's not safe to perform the operation