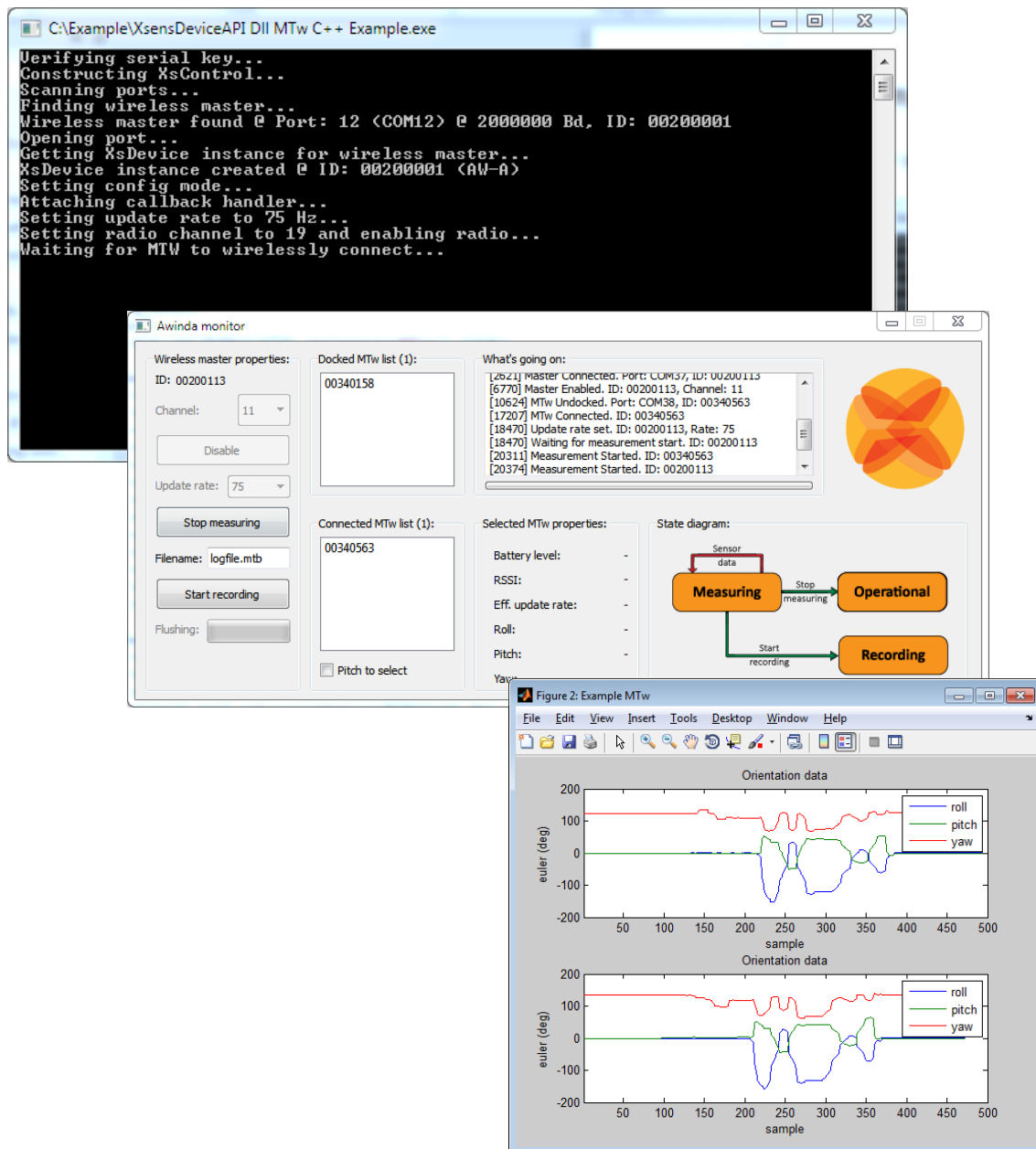




MTw SDK User Manual

Quick start guide for MTw Users

Document MW0202P, Revision D, 25 October 2013



Xsens Technologies B.V.

Pantheon 6a
P.O. Box 559
7500 AN Enschede
The Netherlands

phone +31 (0)88 973 67 00
fax +31 (0)88 973 67 01
e-mail info@xsens.com
internet www.xsens.com

Xsens North America, Inc.

10557 Jefferson Blvd,
Suite C
CA-90232 Culver City
USA

phone 310-481-1800
fax 310-416-9044
e-mail info@xsens.com
internet www.xsens.com



Revisions

Revision	Date	By	Changes
A	14 January 2011	CMO	First version.
B	December 2012	MHE	Major revision for MT SDK 4.1
C	May 2013	CMO	Revised for Release MT SDK 4.1.5
D	25 October 2013	CMO	Revised settling time details.

© 2005-2013, Xsens Technologies B.V. All rights reserved. Information in this document is subject to change without notice. Xsens, MVN, MotionGrid, MTi, MTi-G, MTx, MTw, Awinda and KiC 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	5
1.1	GETTING STARTED WITH MT MANAGER.....	6
2	SOFTWARE ARCHITECTURE.....	7
2.1	INTERFACING THROUGH THE C-INTERFACE LIBRARIES	7
2.2	INTERFACE THROUGH THE C++ WRAPPER	8
2.3	INTERFACE THROUGH THE COM-OBJECT API	8
2.4	INTERFACE THROUGH DLL API	9
3	WORKFLOW	10
3.1	STATES.....	10
4	GETTING STARTED WITH XDA	14
4.1	IMPORTANT OBJECTS WITHIN XDA	14
4.2	WORK FLOW.....	15
5	EXAMPLES	18
5.1	C++ COMMAND LINE EXAMPLE	18
5.2	C++ GUI EXAMPLE: THE AWINDA MONITOR.....	19
5.2.1	<i>Functional description</i>	19
5.2.2	<i>Qt</i>	20
5.3	MATLAB	20
5.3.1	<i>Functional Description</i>	20

Abbreviations and Terms

Term	Description
SDK	Software Development Kit
API	Application Programming Interface
MT	Motion Tracker
MTB	MT Binary Communication Protocol
MTM	MT Manager
SDI	Strap down integration
XKF-3	Xsens Kalman Filter 3 degrees of freedom
XKF-3w	Xsens Kalman Filter 3 degrees of freedom for MTw

Default folders

Description	Files	Location
MT Manager	MT Manager	C:\Program Files ... \Xsens\MT Software Suite ... \MT Manager
Software Development Kit	MTw SDK	C:\Program Files ... \Xsens\MT Software Suite ...
Documentation	MTw User Manual	C:\Program Files ... \Xsens\MT Software Suite ... \Documentation

1 Introduction

The Xsens MTw Development Kit includes a software development kit (SDK). The main objective of the SDK is to facilitate easy development of applications based on Xsens motion trackers.

The MT SDK consists of the Xsens Device API (XDA) library, its header files, and reference (API) documentation, as well as a number of examples, in C, C++, Linux and MATLAB. The XDA is an extensive library of classes and functions needed to work with Xsens motion trackers.

Source code of the MT SDK is made available in C, since this language can be handled by many other programming languages, such as C++, Java and Python. Since C++ is a more convenient language to use for many (first-time) users of the MT SDK, Xsens also supplies a C++ wrapper around the C-compiled library. Figure 1 is a schematic overview of the MT SDK, interacting with Xsens hardware or saved data. Note that conceptually XDA makes no distinction between the cases that the data source is a real-time data stream from a device or if it is a recorded file data stream.

The application developer can choose to use a COM, C or C++ interface. However, only the C interface is delivered as a compiled dynamic link library. For the C++ interface the source code of the wrapper classes are supplied as part of the SDK. The interfaces are discussed in more detail below.

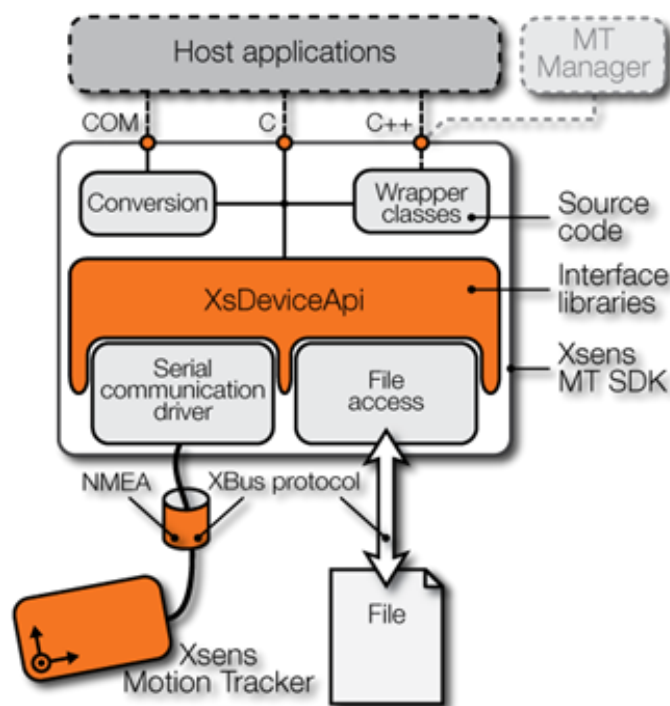


Figure 1: Overview of Xsens SDK and interaction with Xsens hardware

This document provides an overview of the MT SDK with suggested workflow, example code, including a C++ example, called the Awinda Monitor, built using the MT SDK in C++.



1.1 Getting Started with MT Manager

Xsens software, including the end-user software (eg MT Manager) is built on the Xsens Device API (XDA). The easiest way to get started with the MTw Development Kit is to use MT Manager for Windows 7. This easy to use software, with familiar Windows user interface allows users to:

- Get a feel for the proposed workflow of the MTw system;
- View 3D orientation in real-time;
- View inertial magnetic and pressure 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;

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

2 Software architecture

XDA implements a great deal of functionality for interfacing to multiple MTw devices simultaneously. It provides a wrapper for all lower level functionality and exposes the lower level functionality through its interfaces. XDA implements automatic handling of multiple connected devices and automatic data-queuing (buffering). XDA relieves a lot of the implementation details of handling MTw's and Awinda Masters from the application programmer. XDA allows automatic logging to a file, bundled data reception from multiple MTw's, automatic (software) synchronized start up, etc. The handling of incoming MTw communication protocol messages and certain device requests is performed in a separate thread and most measurement-mode messages can be sent and received without disturbing incoming data.

XDA is available to developers as a Windows dynamic link library (.DLL), as a COM object and as a Linux shared object (.SO).

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

- Strap down integration (SDI) data to calibrated data.
- SDI to orientation data.
- Conversion from one orientation mode to another (quaternion to Euler).

This is particularly useful for post-processing of logged data using different settings (e.g. XKF Scenarios).

2.1 Interfacing through the C-interface libraries

XDA is implemented in C-interface libraries that are supplied for MS Windows and Linux, consisting of two parts:

- XDA that contains the access to functionality as implemented in devices, e.g. configuring the Motion Trackers, requesting data etc (Linux only available as beta via support@xsens.com).
- XsTypes that contains generic types (vectors, matrices, quaternions, etc.) and some basic operations, e.g. converting quaternions into Euler angles.

For complete and detailed documentation please refer to the API doxygen documentation which is part of the MT SDK. Note that the MT SDK and the API documentation provide support for all Xsens motion trackers, therefore not all information is directly needed for the MTw Development Kit. The API documentation is found in C:\Program Files... \Xsens\MT Software Suite ... \Documentation\ . It is advised to refer to this documentation as soon as possible when creating an application.

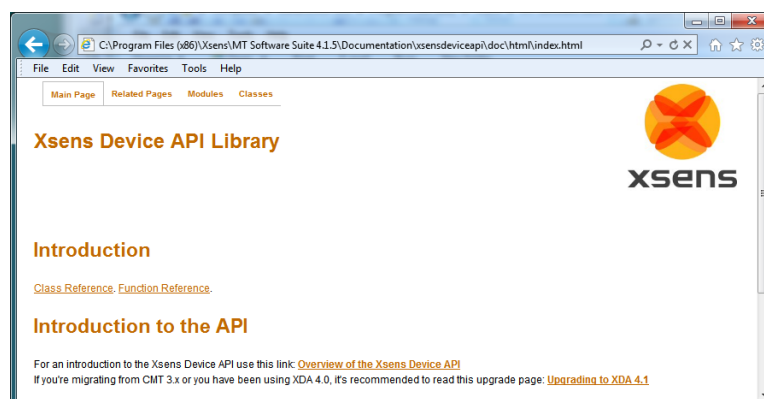


Figure 2: The API reference doxygen documentation



The C API exposes all possible functions that could be supported by an Xsens device. As such, a certain functionality implemented in devices is accessible by a function call that takes at least an XsDevice Object as a parameter. Not every Xsens device supports all functionality, e.g. an MTi or an MTw do not have position estimate whereas the MTi-G does. This means that whether the function returns a meaningful result depends on the device that is connected. Exposing all the possible functionalities has the advantage that when changing the motion tracker in the application to a device with other functionalities, the majority of the code can remain unchanged.

It is important for the user to use only functions supported by the connected device. During run time, calling an unsupported function will generate an error status in line with the normal error handling framework.

2.2 Interface through the C++ Wrapper

The XDA is also offered as a C++ interface which implements a convenience wrapper around the C API. This means that the developer does not have to deal with memory management (i.e. easy object-lifetime management) as the class implementation takes care of this. This means that for example functions named `XsDevice_<function name>` in the C interface are available in the C++ interface as the `<function name>` method of the `XsDevice` class.

2.3 Interface through the COM-object API

The easy way to develop a Windows software application, is using the COM-object API. The COM object is the preferred interface when developing an application within another application such as MATLAB, LabVIEW, Excel, etc.

The COM-object provides easy to use function calls to obtain data from the motion tracker or to change settings.

There is a 32-bit and a 64 bit version of the interface:

	DLL-name	Interface name
32-bit	<code>xsensdeviceapi_com32.dll</code>	<code>xsensdeviceapi_com32.IXsensDeviceApi</code>
64-bit	<code>xsensdeviceapi_com64.dll</code>	<code>xsensdeviceapi_com64.IXsensDeviceApi</code>

The COM-object is a DLL that is registered on the operating system (Windows). When correctly installed, it is possible to access the functions of the COM-object in all Windows applications that support COM.

It is also possible to manually register the DLL to the operating system using the following command at the command prompt:

```
regsvr32 xsensdeviceapi_com32.dll
```

The COM-object takes care of the low-level communication with hardware and it is an easy way to obtain (soft) real-time performance. Typically this is preferred when motion tracker capabilities are directly required in application software such as MATLAB, LabVIEW, Excel (Visual Basic), etc. (examples are provided).



2.4 Interface through DLL API

The preferred interface for developing standalone Windows applications for the MT is the DLL API.

The DLL interface consists of two DLLs and their accompanying link libraries; there is a 32-bit and a 64-bit version for each:

	DLL-name	Link library
32-bit	xsensdeviceapi32.dll	xsensdeviceapi32.lib
64-bit	xsensdeviceapi64.dll	xsensdeviceapi64.lib
32-bit	xstypes32.dll	xstypes32.lib
64-bit	Xstypes64.dll	Xstypes64.lib

The functions in the DLL use the standard C calling convention, which makes them easy to use in any programming language. For C++ developers, the header files provide wrapper classes around the C-interface that provide RAII functionality and other C++ language features. The `xsensdeviceapi.h` header file provides the necessary functions and classes.

3 Workflow

The following explains the recommended workflow when using the MTw Development Kit with XDA. The system, involving both the Awinda Master and the MTw, is known to be in certain states. Possible functionality is specifically related to a given state. The possible transitions from one state to another is limited.

It is advised to understand the states and transitions of both the Awinda Master and of the MTw in order to work according to the recommended workflow as much as possible.

3.1 States

Figure 3 provides an overview of the states of the Awinda Master; Table 1 describes these states and transitions with some suggestions for using XDA.

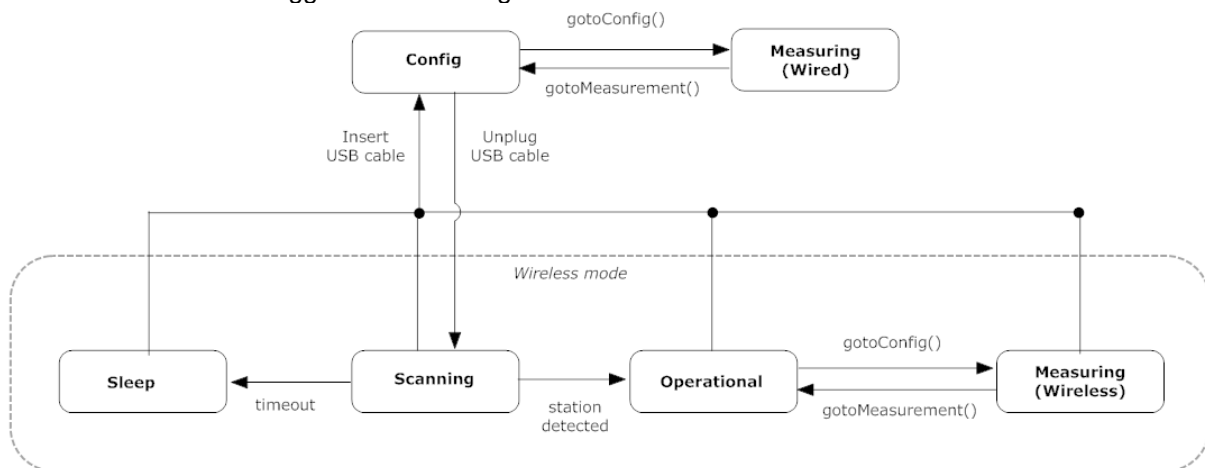


Figure 4 and Table 2 show that of the MTw.

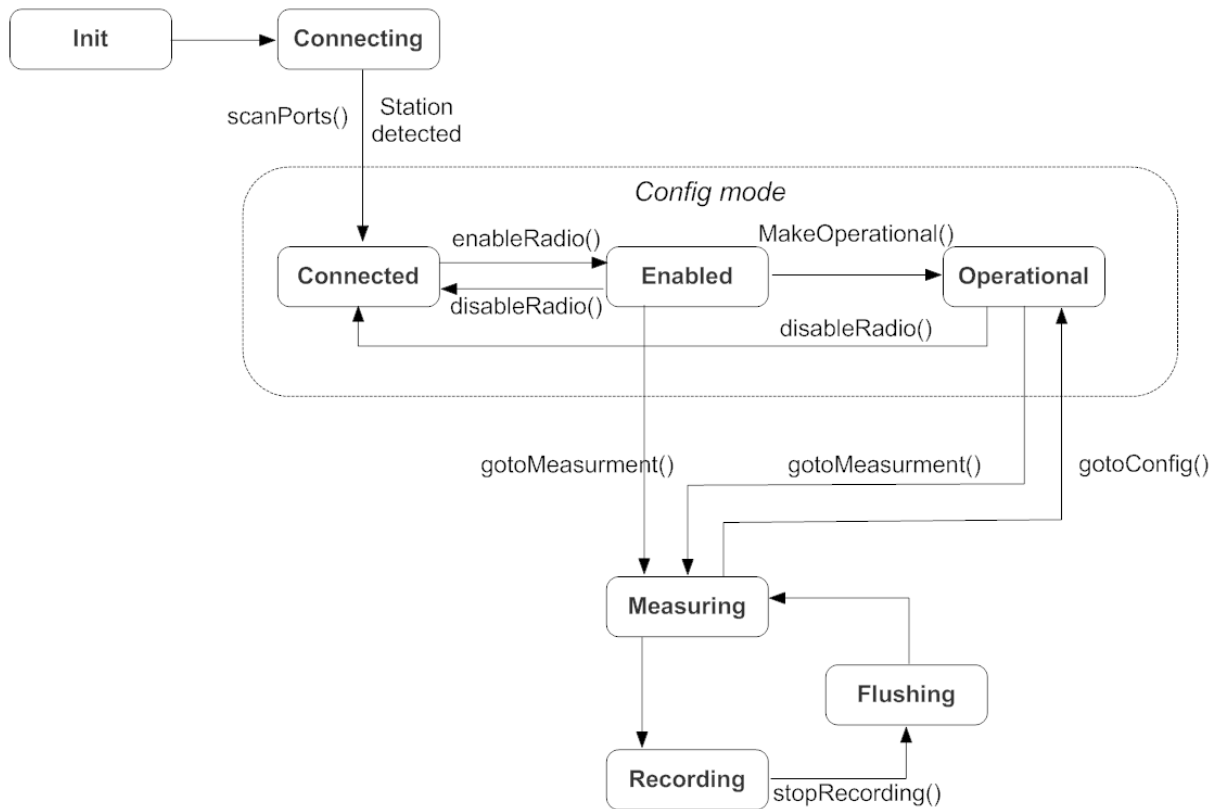


Figure 3: State diagram of the Awinda Master

Table 1: States and transitions of the Awinda Master

State	Description	Suggested Action
Connecting	Awinda Master not yet connected to XDA.	Do a port scan, open port and request device instance.
Connected	The Master is detected. The port is open and connected to XDA. The radio of the Awinda Master is not yet transmitting.	Call enableRadio() with the required radio channel.
Enabled	The radio of the Awinda Master is switched on and broadcasting on the specified channel. MTw's can detect this transmission and start to connect to the network.	Wait until all devices have joined the network. Give the gotoMeasurement() command.
Operational	The Master is operational, the wireless network is established. No new MTw's can connect to the network.	Give the gotoMeasurement() command to start measuring, or disableRadio() to switch the radio off on the Awinda Master, and return to Enabled state.
Measuring	The MTw's are measuring and transmitting data to the Master. The Master forwards each data packet received to XDA.	Continue Measuring
		Return to the Operational state to make changes to the wireless network using the gotoConfig() command.
		Enter Recording state using the startRecording() command.
Recording	Recording is in progress. A new file is created, wherein transmitted data is stored. If the maximum update rate was chosen, recorded data cannot contain retransmitted data. If a lower update rate is selected, the recording will contain retransmissions when necessary.	Continue recording.
		Return to measurement state, using the command stopRecording.
Flushing	Recording was stopped. Data still on the buffer of the MTw are retransmitted to the Awinda Master.	Wait until flushing is complete.
		Close the logfile using closeLogFile().
		Return to Enabled state using gotoConfig().

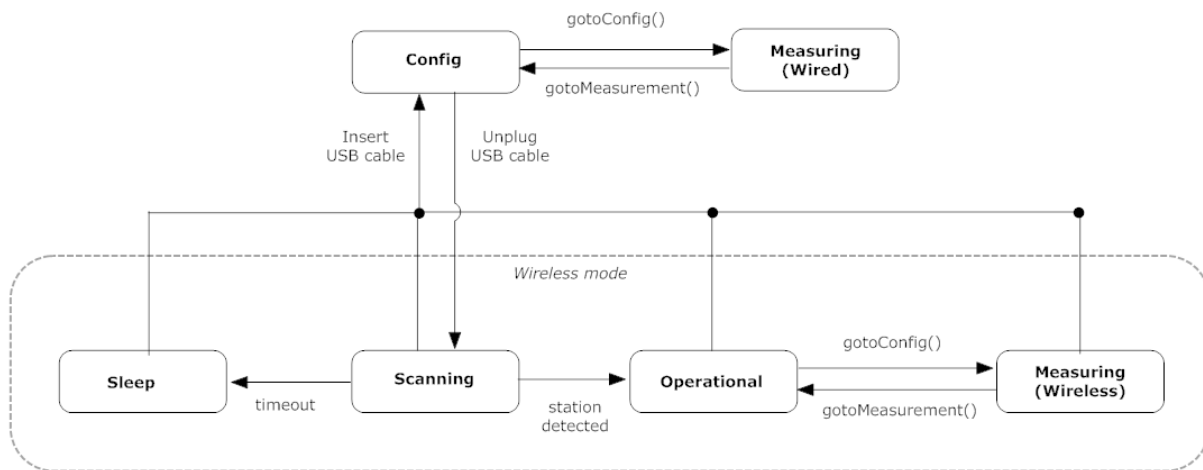


Figure 4: State Diagram of MTw

One of the important states to note in the MTw is Sleep. To wake the MTw from Sleep, it should be rotated, to change its magnetic field. Operational and wireless measuring states are controlled by the Awinda Master.

Table 2: States and transitions of the MTw

State	Description	Suggested action
Sleep	Power saving state to which the MTw enters if it is idle for some time.	Rotate the MTw to return to scanning state. Dock the MTw into the Awinda Station to enter Config state.
Scanning	MTw scans all radio channels to find a Master in the Enabled state.	Wait until connected indications are received. (see Enabled state of the Awinda Master).
Config	Device is docked. Configurations (update rate, radio channel) can be changed.	Configure the device.
		Undock device to start scanning for a Master.
		Start measurement while docked (not intended use).
Operational	MTw is wirelessly connected to a Master but not yet measuring.	None. MTw is controlled by Master.
Measuring (Wireless)	MTw is wirelessly connected to a Master and in measuring state.	None. MTw is controlled by Master.
Measuring (Wired)	MTw is wired or docked and in measuring state	Undock device to start scanning for a Master.
		Stop measurement while docked to enter config state.

4 Getting started with XDA

The system flow described above requires workflow and specific commands within Xsens XDA. The following summarises important stages of XDA and commands used during this process.

4.1 Important Objects within XDA

The following summarises important objects associated with the workflow needed for using XDA.

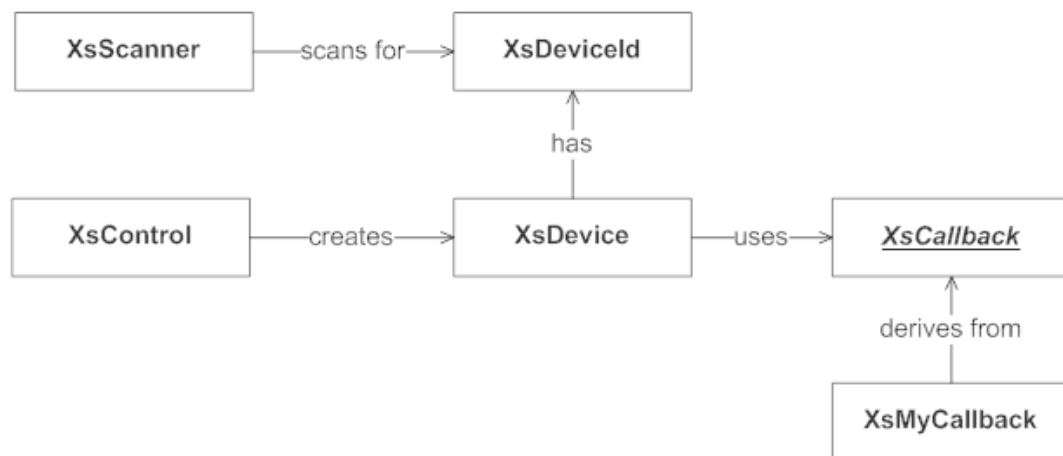


Figure 5: Overview of main XDA Objects

XDA Object	Description
XsScanner	XsScanner is used to scan ports for Xsens devices. Detected devices have an XsDeviceId, a baud rate and a port name.
XsDeviceId	XsDeviceId is a unique identifier for an Xsens device. Every Xsens product has a serial number. This is usually found on the bottom/back of the casing. The serial number is called the DeviceId in the software.
XsControl	XsControl is the central XDA access point. It can open detected ports, using the XsScanner. XsControl creates and maintains XsDevice instances for Xsens devices connected to the open ports. Using the XsDeviceId, to request an XsDevice to work with. When using the XDA, a single XsControl instance should always be created. Before the XDA can be used, set the serial key using XsControl.
XsDevice	An XsDevice object represents an Xsens device such as a Master or an MTw. Use XsDevice for all communication with the device, including configuring or starting measurements. XsDevice instances are created by XsControl.
XsCallback	XsCallback is a base class containing virtual callback methods that are called by XDA when various events such as OnDataAvailable occur. They should be overridden in derived classes to be able to handle these events. Instances of objects derived from XsCallback should be attached to XsDevice instances to be able to handle events.
XsDataPacket	An XsDataPacket is a representation of one data packet received by an XsDevice. It contains various data fields, such as a time stamp, orientation data, and inertial data, and access methods to retrieve the data.



4.2 Work Flow

This section shows the necessary steps required to start a recording on an Awinda Master with multiple wireless MTw's and process the generated data using XDA.

First, the serial key has to be set. If the serial key is not set, a `SerialKeyException` will be thrown when trying to use XDA.

- `XsControl::setSerialKey()`

Create an `XsControl` object.

- `XsControl::construct()`

Scan all ports for connected Xsens devices and locate an Awinda Master. The device type is known from the `XsDeviceId`.

- `XsScanner::scanPorts()`
- `XsDeviceId::isWirelessMaster()`

Open the port and get an `XsDevice` instance for the Master.

- `XsControl::openPort()`
- `XsControl::device()`

To handle connecting MTw's, attach a callback handler to the Master. This callback handler is an object derived from `XsCallback` and overrides the callbacks for detecting connecting MTw's (e.g. `onConnectivityChanged`).

Callback handlers are not managed by XDA; they should be deleted manually when no longer required.

- `XsDevice::addCallbackHandler()`

Configure Awinda Master.

- `XsDevice::setUpdateRate()`

Set the radio channel and enable the radio. The available channels are 11 to 26. When expecting interference with WIFI devices, try Channels 15 and 20 or 25 and 26, see MTw User Manual for more information.

- `XsDevice::enableRadio()`

When the radio is enabled, MTw's can wirelessly connect to the Master. Undock the required MTw's and wait for a connection to be established. When an MTw connects, the `onConnectivityChanged` callback is called, which provides the MTw's `XsDevice` instance and the new `XsConnectivityState`, which should be `XCS_Wireless`.

- `XsCallback::onConnectivityChanged(XsDevice*, XsConnectivityState)`



In order to obtain a good initial estimate, keep the MTw still for a short time, just before entering measurement mode.

XsDevice instances of connected MTw's are not yet fully functional at this stage. When the MTw's are in measurement mode, the full functionality will be available. When all MTw's are connected, the measurement can be started. Use the XsDevice instance of the Master to put the Master and all wirelessly connected MTw's into measurement mode.

Like all filters of its kind, the XKF-3w filter running inside the MTw needs time to settle to a stable state. This time is called settling time.

For about one minute after entering measurement mode, make calm, slow movements to warm up the filters. To check if the filters have been stabilised, ensure that the orientation of the MTw doesn't change when the MTw is stationary (in a magnetically undisturbed environment).

- `XsDevice::gotoMeasurement()`

It is **important to note** that when the Master changes its device state (particularly when starting measurement state), existing XsDevice instances may become invalidated. To get the correct XsDevice instances for all connected MTws, use the XsControl instance.

Note: the XsDeviceIds will always remain valid.

- `XsControl::deviceIds()`
- `XsDevice::isMtw()`
- `XsControl::device()`

To be able to process the data generated by the MTw's, attach a callback handler to the individual MTw XsDevice instances. This callback handler should be an object derived from XsCallback and should override the callbacks that should be handled (e.g. `onDataAvailable` for processing the generated data).

- `XsDevice::addCallbackHandler()`

When the callback handlers are attached, the callbacks will be called when data for an MTw is generated.

- `XsCallback::onDataAvailable(XsDevice*, const XsDataPacket*)`

Data generated by the MTw's is now available in an XsDataPacket and can be processed.

Callback handlers are called in the context of the XDA thread. Ensure that appropriate locking and thread synchronization are used here. Also it is not recommended to do time consuming processing in the context of the XDA thread. Take this into account when handling e.g. `onDataAvailable`. It is preferable to e.g. copy the data to a buffer and do the actual processing of the data in another thread. Orientation data, e.g. euler data, can be retrieved from the packet.

- `XsDataPacket::orientationEuler()`



When in measurement mode, recording can be started. A logfile has to be created first (using the Awinda Master XsDevice instance).

- `XsDevice::createLogFile()`
- `XsDevice::startRecording()`

Recording is started when the device is in recording state. Note that it is possible to configure the device in such a way that it will wait for an external trigger before the recording is actually started. Always wait for recording to be actually started using the `onDeviceStateChanged` callback. This callback will be called when the device is in `XDS_Recording` state.

- `XsCallback::onDeviceStateChanged(XsDevice*, XsDeviceState, XsDeviceState)`

When ready, stop recording. Flushing of data is done automatically if needed.

- `XsDevice::stopRecording`

When flushing (if any) is completed and recording has stopped, the device will be in measurement mode again. Use the `onDeviceStateChanged` callback to detect this. The device will be in `XDS_Measurement` state. After the device is back in measurement mode, the log file can be closed.

- `XsCallback::onDeviceStateChanged(XsDevice*, XsDeviceState, XsDeviceState)`
- `XsDevice::closeLogFile()`

When ready, stop measuring and return to config state (this puts the Master into the Operational state).

- `XsDevice::gotoConfig()`

Disable the radio to disconnect all wireless MTw's. Note that at this time, existing XsDevice instances for MTw's will be invalidated.

- `XsDevice::disableRadio()`

When XsControl is no longer needed, close it to free up resources.

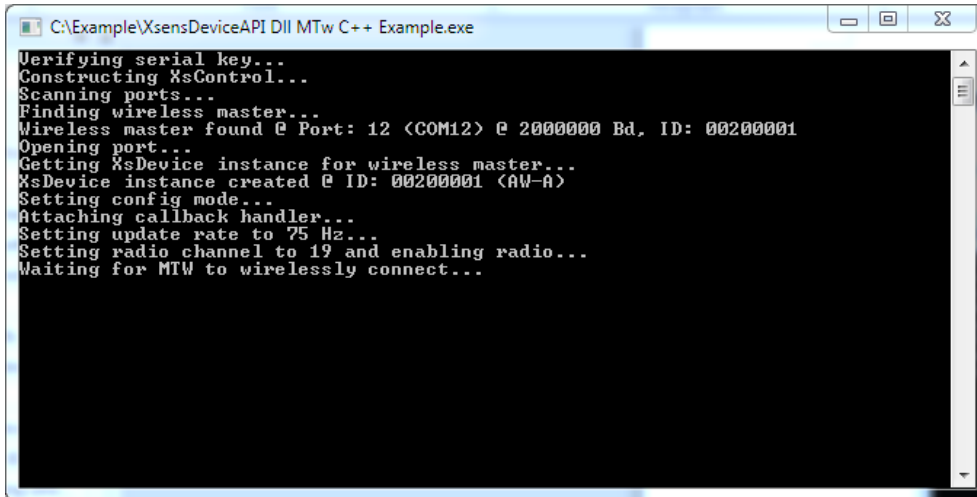
- `XsControl::close()`

Note: Remember to delete the XsCallback instances, since they are not managed by XDA.

5 Examples

5.1 C++ Command Line Example

The C++ command line example shows how an Awinda system must be initialized and how the system can be set in measurement mode.



```

C:\Example\XsensDeviceAPI Dll MTw C++ Example.exe
Verifying serial key...
Constructing XsControl...
Scanning ports...
Finding wireless master...
Wireless master found @ Port: 12 (COM12) @ 2000000 Bd, ID: 00200001
Opening port...
Getting XsDevice instance for wireless master...
XsDevice instance created @ ID: 00200001 (AW-A)
Setting config mode...
Attaching callback handler...
Setting update rate to 75 Hz...
Setting radio channel to 19 and enabling radio...
Waiting for MTW to wirelessly connect...

```

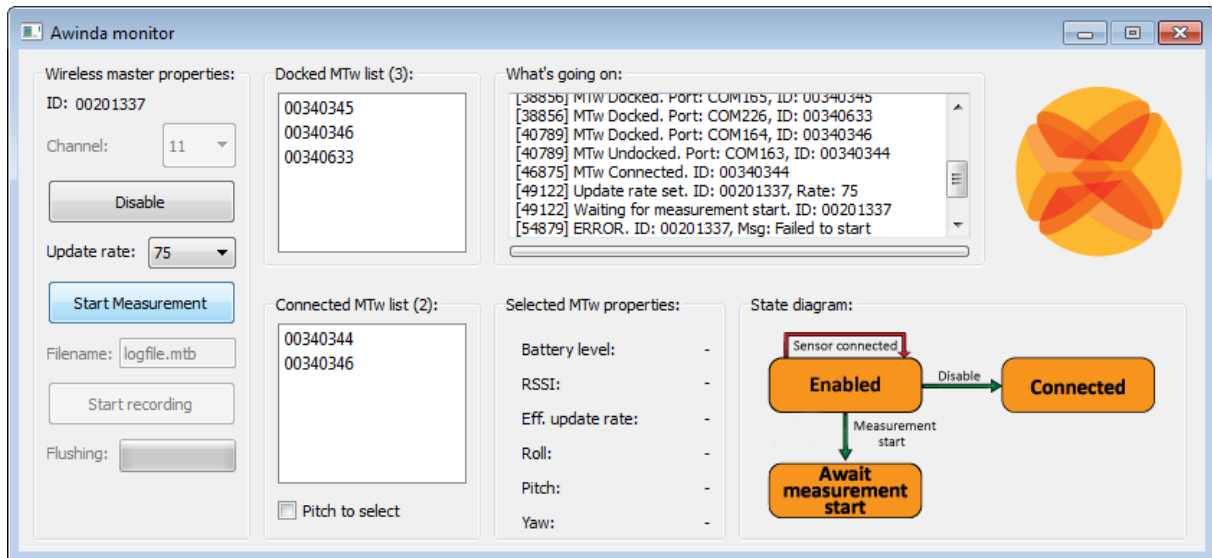
The following table summarises the file names needed for this example:

Directory	...\MT SDK 4.1.5\Examples\dll_example
Visual studio project file	dll_example_mtw_c++.vcproj
Project name	XsensDeviceAPI Dll MTw C++ Example
Source files	example_mt_w_c++.cpp serialkey.cpp

Although the command line example is quite simple it is not completely minimalistic. Aside from the basic steps required to get the Awinda system to measurement mode, it has some functionality to print the measured data on screen and for thread synchronization.

5.2 C++ GUI Example: The Awinda Monitor

A more complete C++ example is the Awinda Monitor. An example screenshot can be seen in the figure below.



The following table summarises the filenames needed to build this example

Directory	...MT SDK ...Examples\awindamonitor
Visual studio project file	awindamonitor.vcproj
Project name	awindamonitor
Source files	connectedmtwdata.cpp main.cpp mainwindow.cpp myxda.cpp serialkey.cpp
GUI form file	mainwindow.ui
Third party libraries	QtCored4.lib QtGuid4.lib

5.2.1 Functional description

The "Awinda monitor" has the following functionality:

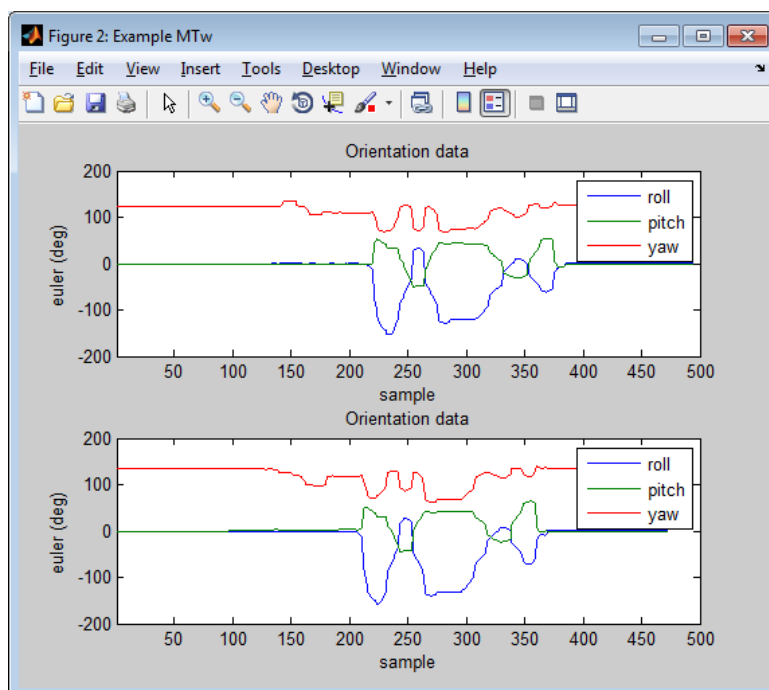
- Detect and connect an Awinda Master.
- Detect docking and undocking MTw's.
- Enable and disabling the detected Awinda Master at a specific channel.
- Detect connecting and disconnecting MTw.
- Show RSSI of connected MTw's.
- Set update rate.
- Starting and stopping the measurement.
- While measuring or recording, show battery level, RSSI, effective update rate, roll, pitch and yaw.
- Starting and stopping recording to a specified MTB-file.

Note that not all functionality offered by XDA is used in this example. The list above corresponds to the basic flow.

5.2.2 Qt

The Awinda monitor uses Qt for its graphical user interface. The Qt SDK can be downloaded for free from the Qt Project web site. For this example Qt 4.7.x or Qt 4.8.x is required. The Qt SDK contains the required Qt header files as well as Qt libraries and DLLs. It also contains the Qt Designer application that is used to edit the form file. If Visual Studio is used, the *Qt Visual Studio Add-in* is required. This add-in must be separately downloaded and installed. Among other things, this add-in takes care of Qt's Meta-Object Compiler (moc) build step. In addition to the visual studio project file, a Qt-Creator project file (awindamonitor.pro) is available for this example.

5.3 Matlab



The following table summarises the filenames needed to run this example

Directory	...\MT SDK 4.1.5\Examples\Matlab
Source file	mainMTwRTdataViewer.m

5.3.1 Functional Description

The MATLAB script provides a step wise procedure to get data from devices connected to the Awinda Master into a wireless network and collect data. The code is divided into two parts. The first part is concerned with communicating with the MTw's docked in the Awinda Station. This involves scanning and opening ports, selecting radio channel and wirelessly connecting to the MTw's that are subsequently undocked and ready for wireless transmission. The second part focuses on using the MTw in wireless mode; each step is designed to ensure that accurate recording of data and subsequent data analysis is possible.