Setup Guide - **HAL API**

HAL-API for software development with **m:explore** ultra-wideband sensors

Ilmsens GmbH
Ehrenbergstraße 11
98693 Ilmenau
Germany
Tel.: +49 3677 76130-30
Fax: +49 3677 76130-39
Email: hal-api@ilmsens.com

Table of contents

1 Setting up the HAL API for m:explore UWB sensors 3
1.1 Introduction 3
1.2 Copyright and Disclaimer 3
1.3 Platform support 4
1.4 Platform requirements 4
2 HAL API setup on Windows™ OS 5
2.1 Setup step by step 5
2.2 Example project for Visual Studio C++ 6
3 HAL API setup on Linux OS (with DEB support) 7
3.1 Setup step by step 7
3.2 Example project for gcc and make 8
4 HAL integration test 9
4.1 Program flow of integration test 9
4.2 Command line options 10
5 Further resources and revision history 11
5.1 Further resources 11
5.2 Document revision history 11

1 Setting up the HAL API for **m:explore** UWB sensors

1.1 Introduction

As a service to our customers who want to integrate Ilmsens UWB sensors into their software environment, Ilmsens offers a hardware abstraction layer (HAL) application programming interface (API) ("The software") for the **m:explore** ultra-wideband (UWB) sensors. By default, the HAL API comes in binary/complied form (i.e. a dynamic library with corresponding C header files) working on top of the device drivers. This guide explains setup of the HAL to start application development. Further information can be found in a separate programming guide and a function reference manual from Ilmsens (compare section 5.1).

The HAL API is available for different popular operating systems and allows device management, sensor configuration, acquisition configuration, and performing measurements. It abstracts from device- or digital interface-specific details as much as possible to enable portability of the application software. Future generations of the HAL API and Ilmsens UWB sensors will be developed with backward compatibility in mind. Product-specific extensions will extend the API rather than changing existing functions.

1.2 Copyright and Disclaimer

1.3 Platform support

The HAL API is currently available for the following operating system platforms:

- Windows™ Vista™ / Windows™ 7 / Windows™ 8.x / Windows™ 10
  - x86 (32 Bit) and x64 (64 Bit) versions
- Standard Linux distributions (currently those based on Debian package management)
  - i386 (32 Bit), amd64 (64 Bit), armel (ARM ports) versions
  - Ubuntu LTS 14.04 "Trusty Tahr" and up
  - Ubuntu LTS 16.04 "Xenial Xerus" and up
  - Ubuntu 16.10 "Yakkety Yak" and up
  - Debian LTS 7 "Wheezy" and up
  - Debian 8 "Jessie"

If you want to use the HAL API on other Linux distributions, please contact Ilmsens (see section 5.1). Support may be possible, if a sufficient development C/C++ tool chain is available for your distribution and Ilmsens has access to a reference installation.

Support for other platforms, such as MacOS™, Solaris™, FreeBSD, etc., may be added in the future.

1.4 Platform requirements

The HAL API works on top of the device drivers provided with your **m:explore** sensor and requires the sensor(s) to be connected to the computer and powered on for useful operation. Therefore, these platform requirements apply:

- Any of the supported operating systems listed under section 1.3
- USB2.0 host port
- Windows OS™:
  - USB device drivers provided with the Ilmsens **m:explore** evaluation kit
  - Microsoft Visual C++™ redistributable runtime in a matching version (included in the HAL package, usually already installed on Windows™ PCs)
- Linux OS:
  - LibUSB-1.0-0: V1.0.11 (or later) package installed
  - LibPocoFoundation9: V1.3.6 (or later) package installed
  - Device driver integrated into HAL API deb-package
- C/C++ development environment with dynamic library import
  - Free Microsoft™ Visual C++ Express 2012 and up supported
  - Linux: gcc 4.7.2 or later (5.x.x may work but was not tested)

2 HAL API setup on Windows™ OS

The HAL library requires the device drivers to be installed for your **m:explore** devices. It interfaces with the driver to communicate with sensors attached to your computer. It is important to note, that all library components and your development environment must have matching platforms, i.e. you can either integrate the x32 HAL with a native 32 bit application or the x64 HAL with a native 64 bit application. It is recommended to match the development platform with your operating system platform, too.

2.1 Setup step by step

The following steps must be taken to setup the HAL library for application development. Some steps may not be necessary for your computer and can be skipped (e.g. if dependencies are already installed). Other steps must be done, but the actual procedure depends on your development environment (e.g. importing functions from a DLL):

1. Install the device driver (if not already done)
    a. Please consult your **m:explore** hardware manual for installation details.
    b. You must have elevated administrator privileges for driver installation.
    c. **m:explore** sensors use drivers based on WinUSB, which are built into modern Windows operating systems.
    d. The driver installer is included in both the software package and the HAL library package. It does not matter which one you use.
    e. On systems using Windows 8 or newer, registration of the signing certificate for the driver installer is required (otherwise the OS will refuse the install)
2. Extract the contents of the HAL library archive
    a. If you received the HAL API in an archive (e.g. ZIP), extract the archive contents to an user-accessible folder.
    b. There are separate archives for different machine architectures (e.g. Win32 and Win64).
    c. That folder will be the HAL library root folder `<HAL root>`
3. Install the VC runtime redistributable for your platform
    a. Maybe skipped, if the correct redist. package is already installed.
    b. Installation requires elevated administrator privileges.
    c. For 32 bit development, use:
        `<HAL root>\redistributables\vc_redist.x86.exe`
    d. For 64 bit development, use:
        `<HAL root>\redistributables \vc_redist.x64.exe`
4. Check contents of ‘bin’ sub-folder in the platform directory `<HAL root>\bin`
    a. Contains all required dependencies and the actual HAL DLL `ilmsens_hal.dll`

5. Test HAL library installation using the integration test application
    a. This step makes sure, all dependencies are fulfilled
    b. Connect your **m:explore** device to your computer
    c. Check Windows device manager to make sure, drivers are installed correctly
    d. Start a command console (CMD.EXE) and go to respective 'bin' folder
    e. Start the HAL integration test with trace mode logging, e.g.:
    ```
    hal_itest.exe --timeoutMillis 100 --responseCount 10 --logLevel
    trace 2> hal_itest.log
    ```
    f. Check output log file for errors
    g. For details about the integration test, see chapter 4
6. Import the HAL library into your development environment
    a. Copy all DLLs from the HAL 'bin' folder to your application development project's binary output folder (where your EXE-file will be created).
    b. Add the 'include' folder to your project's include paths: `<HAL root>\include`
    c. The C header files contain, #defines, HAL types, and the HAL API function declaration.
    d. The HAL API header files and DLL can be used to generate an import-library for linking the DLL's functions with your application. Consult your development environment's documentation if that step is required and how to do so.
    e. Include the optionally generated libraries as input to your linker tool.
7. Your C/C++ applications can now use the HAL functions
    a. Make sure, that your executable can find the DLLs of the 'bin' folder at runtime (see step 7.a.).
    b. Include the main header file into your C/C++ code:
    ```
    #include "ilmsens/hal/ilmsens_hal.h"
    ```
    c. Make sure, that the release package of your application includes all DLLs from the 'bin' folder of the HAL.
    d. Please include the corresponding VC runtime redist. executable provided by the HAL package in your application release.

2.2 Example project for Visual Studio C++

An example project with a simple test application will be added in a future HAL release. Contact Ilmsens (see section 5.1) for a schedule and further information.

3 HAL API setup on Linux OS (with DEB support)

The HAL library comes in form of a Debian DEB package and requires the device drivers to be registered for your **m:explore** devices. Since the communication utilises the standard library "libusb" and "udev" device management, no proprietary kernel module will be installed. Registration of the devices' hardware IDs for udev will be part of the DEB package installation. It is important to note, that your Linux OS, all library components, and your development environment must have matching platforms, e.g. x86, x64, or armel.

3.1 Setup step by step

The following steps must be taken to setup the HAL library for application development and deployment. Some steps may not be necessary for your computer and can be skipped (e.g. if dependencies are already installed). Other steps must be done, but the actual procedure depends on your development environment (e.g. importing functions from a dynamic library). It is assumed, that your Linux OS uses DEB package management:

1. Download the HAL DEB-package to your computer.
    - Make sure to get the package matching your distribution and architecture
2. Root privileges are required for some of the installation steps
    - They can also be done via ‘sudo’.
3. Do an update of package index (optional)
    - Start a console
    - `$ sudo apt-get update`
4. Try to install the HAL DEB-package
    - `$ sudo dpkg -i ilmsens-hal-X.Y.Z-<dist_name><dist_ver>_<arch>.deb`
5. Install unsatisfied dependencies
    - e.g. libpocofoundation9, libusb-1.0-0, etc.
    - `$ sudo apt-get install –f`
6. Finish installation of the HAL DEB-package
    - `$ sudo dpkg -i ilmsens-hal-X.Y.Z-<dist_name><dist_ver>_<arch>.deb`
    - This will put the HAL library into the `/usr/lib` tree and test binaries into `/usr/bin`
7. Test HAL library installation using the integration test application
    - This step makes sure, all dependencies are fulfilled
    - Connect your **m:explore** device to your computer
    - Start a console
    - Check to make sure, the device is seen by the kernel: `$ lsusb`
    - Go to binary's folder: `$ cd /usr/bin`
    - Start the HAL integration test with trace mode logging
    - `$ hal_itest --timeoutMillis 100 --responseCount 10 --logLevel trace 2> hal_itest.log`

- Check output log file for errors
- For details about the integration test, see chapter 4

8. Import the HAL library into your development environment
    - Add the user include folder to your project's include paths: `/usr/include`
    - The C header files contain, #defines, HAL types, and the HAL API function declaration.
    - Include the HAL library as input to your linker tool, e.g. with the GNU linker: `ld … -lilmsens_hal …`
9. Your C/C++ applications can now use the HAL functions
    - Make sure, that your executable can find the dynamic HAL libraries.
    - Include the main header file into your C/C++ code:
      ```
      #include "ilmsens/hal/ilmsens_hal.h"
      ```
    - Please include the Ilmsens HAL library package as a dependency of your application release package.

3.2 Example project for gcc and make

An example project with a simple test application using gcc and make will be added in a future HAL release. Contact Ilmsens (see section 5.1) for a schedule and further information.

4 HAL integration test

The HAL integration test binary 'hal_itest[.exe]' application is included in the API package to let the user test correct installation of all dependencies and dynamic libraries. It is a console application that puts logging messages to `std::cerr` which can be redirected to a log file if needed. The test does not perform any kind of data processing but merely checks correct communication with the device in a short measurement. Furthermore, a successful run of the test does not guarantee that the HAL is correctly integrated into your application development environment. However, if the test fails, overall setup of the HAL library is not correct and should be fixed before working on your own application.

4.1 Program flow of integration test

The HAL integration test performs the following operations:

- Parse command line options (see section 4.2)
- Initialise the HAL library
- Connect to the first **m:explore** sensor detected (order number 1)
    - An error is output, if no sensor could be detected. Please check sensor connection to the computer and device driver installation.
- Setup the sensor with default basic parameters
- Set the sensor to be a master device and perform digital synchronisation
- Measurement test run (options see section 4.2)
    - Start a measurement
    - While waiting for measured data, check for a configurable timeout
    - Collect a configurable number of impulse responses
    - Stop measurement run
- Repeat the measurement test a configurable number of times
- Disconnect from the sensor
- De-initialise the HAL library
- Report return code to user

The return value of the test application is either 0 on success or 1 on failure (if any error occurred). If you have any problems or questions about using the HAL integration test, please contact us (see section 5.1).

4.2 Command line options

Most options require a value argument to be specified. If the command line contains usage errors, the application will also fail and report an error. The following options can be set from the command line:

Option	Valid range	Default value	Description
`--mlbsOrder`	9, 12, or 15	9	M-sequence order of the sensor
`--rfClock`	0.1 .. 18.0	13.312	RF clock rate of the sensor [GHz]
`--softwareAvg`	1 .. 4096	32	Software averages done for each impulse response acquired
`--repeatCount`	1 .. MAX_UINT32	1	Number of test measurement runs
`--responseCount`	1 .. MAX_UINT32	10	Number of impulse responses acquired per test measurement run
`--timeoutMillis`	0 .. MAX_UINT32	500	Timeout [ms] the application waits between impulse responses
`--logLevel`	• none (no logging) • fatal • critical • error • warning • notice • information • debug • trace	warning	Set log verbosity of HAL integration test application. Debug and trace level output a lot of information and `std:cerr` should be redirected into a log file, e.g.: `... 2> trace.log`
`--bufferedMode`	<none>	<not specified>	If this command line flag is specified, the measurement uses buffered (threaded) mode. Otherwise raw mode is used.

The following is an example for running the integration test:

```
$ hal_itest --timeoutMillis 100 --responseCount 10 --logLevel trace 2> hal_itest.log
```

This test sets the impulse response timeout to 100 ms, measures 10 responses per test run and sets logging verbosity to `"trace"` level (maximum logging output). The logging messages are redirected into the log file `"hal_itest.log"`.

5 Further resources and revision history

5.1 Further resources

For further information please visit our website at www.ilmsens.com.

The following documents provide further information for development:

- Measurement hardware and device drivers: "Ilmsens Hardware Manual m:explore" (provided with your **m:explore** Evaluation Kit)
- Drivers and Ilmsens application software: "Ilmsens Software Manual m:explore"
- HAL API design & background info: "Ilmsens HAL API Programming Guide"
- HAL API function reference: "Ilmsens HAL API Function Reference"

If you need assistance with the HAL API feel free to contact us at:

Ilmsens GmbH
Ehrenbergstraße 11
98693 Ilmenau
Germany

Tel.: +49 3677 76130-30
Fax: +49 3677 76130-39
Email: hal-api@ilmsens.com

5.2 Document revision history

Rev.	Date	Author	Description
1.2	01/2017	Her	Initial revision for the **m:explore** HAL API setup guide based on split of "Ilmsens HAL API Manual" V1.2