ardas Documentation

Release 1.1.2

Olivier Kaufmann, Christophe Bastin

Jan 22, 2019

Contents:

CHAPTER 1

Getting started

Get the code:

```
git clone https://github.com/UMONS-GFA/ardas.git
```

Edit the **settings_example.py** file according to your configuration and rename it settings.py.

See the settings list parameters for more information.

Create your sensors

See the sensors for more information.

Don't forget: for using ardas tty, the user must be in **dialout** group !

CHAPTER 2

Describe your sensors

Sensors are described in special files named *sensorXXXX.ssr* where XXXX stands for the sensor that contain *raspardas.py*.

2.1 CALIBRATE THE SENSORS

Uncalibrated FM sensors produce readings that are a frequency. You may want to calibrate your FM sensors to produce meaningful readings in units such as °C or µm and not just Hz.

2.2 CREATE THE .SSR FILES

In the example below, four sensors are created and saved to *.ssr* files.

```
from ardas import sensor_tools as st
# Saves a set of sensors including its calibration in a binary '.ssr' file
sensors = (st.FMSensor(sensor_id='0001', processing_parameters=(-16.9224032438, 0.
↪0041525221, -1.31475837290789E-07,
                                                                2.39122208189129E-012, -1.
↪72530800355418E-017),
                    quantity='temp.', units='°C', output_format='%6.3f', log_
↪output=True),
       st.UncalibratedFMSensor(sensor_id='0002', log_output=False),
       st.UncalibratedFMSensor(sensor_id='0003', log_output=True),
       st.UncalibratedFMSensor(sensor_id='0004', log_output=True),
       )

if __name__ == '__main__':
    for s in sensors:
        print(s.sensor_id + ' - ' + s.quantity + ' : ' + s.output_repr(10000))
        s.save()
    print('reload sensor...')
```

(continues on next page)

(continued from previous page)

```
    s = st.load_sensor('1401')
    print(s.sensor_id + ' - ' + s.quantity + ' : ' + s.output_repr(10000) + ' Log:' +␣
↪str(s.log))
```

CHAPTER 3

Settings

The acquisition settings are given in a the python file *settings.py* that must be present in the same folder as *raspardas.py*.

3.1 IMPORTS

A typical *settings.py* will start with an import section.

```
from ardas import sensor_tools as st
```

3.2 SENSORS

The next section is a tuple of sensors objects named SENSORS. The recommended way is to save the sensor object as a sensor####.ssr file where #### stands for the sensor id (usually a four digit number). Then the *load_sensor* method is used to create the sensor objects.

```
# ArDAS sensors objects connected to the ArDAS channels
SENSORS = (st.load_sensor('0001'), st.load_sensor('0002'), st.load_sensor('0003'), st.
↪load_sensor('0004'))
```

Then several dictionaries define the settings :

3.3 DATABASE

This section gives the settings of the InfluxDB where the output of the sensors will be stored.

```
# connexion to InfluxDB
DATABASE = {
    'host': '*hostname.domain*',
```

(continues on next page)

(continued from previous page)

```
    'port': 8086,
    'user': '*user_name*',
    'password': '*password*',
    'dbname': '*database_name*',
    'series': '*series_name*'
}
```

3.4 ARDAS

In this section the settings of the ArDAS are given.

```
# ArDAS configuration
ARDAS_CONFIG = {
    'station': '0001',
    'net_id': '001',
    'shield_id': '001',
    'integration_period': '0001',
    'tty': '/dev/ttyACM0',
    'raw_data_on_disk': False
}
```

3.5 MASTER

This section lists the settings used to setup a remote connection (i.e. over a telnet connection)

```
# connection to master
MASTER_CONFIG = {
    'local_host': '0.0.0.0',
    'local_port': *port_number*
```

3.6 MESSAGES LOGGING

These settings control how the messages generated by raspardas.py are logged.

```
# messages logging configuration
LOGGING_CONFIG ={
    'debug_mode': False,
    'logging_to_console': False,
    'file_name': 'raspardas_log',
    'max_bytes': 262144,
    'backup_count': 30,
    'when': 'd',
    'interval': 1
}
```

3.7 DATA LOGGING

These settings control how the data are logged locally.

```
# data logging configuration
DATA_LOGGING_CONFIG ={
    'file_name': 'data_log',
    'max_bytes': 1048576,
    'backup_count': 1024,
    'when': 'd',
    'interval': 1,
    'influxdb_logging': True
}
```

3.8 SFTP

These settings are used to define how to transfer updates of raspardas.py to the raspberry_pi (remote).

```
# parameters to push code to raspberry via sftp_transfer.py
SFTP = {
    'host': '*name_of_the_raspberry_pi',
    'username': '*username_for_raspberry_pi_login*',
    'password': '*password_for_raspberry_pi_login*',
    'local_path': '*path_to_local_file*',
    'remote_path': '*path_to_remote_file*'
}
```

CHAPTER 4

Automatic start on boot

Add a cron job

In the example below we assume that the user name is *pi*

```
PYTHONPATH=/home/pi/ardas

# m h  dom mon dow   command
@reboot /usr/bin/python3 /home/pi/ardas/ardas/raspardas.py > /home/pi/ardas/cronlog.
↪log 2>&1
```

CHAPTER 5

Build Arduino sketch

Required libraries

- Adafruit RTClib >= 1.0.0

5.1 Requirements

Install required packages

```
sudo apt-get install arduino cmake gcc-avr binutils-avr avr-libc avrdude
```

Fixing missing openjdk library in Debian Stretch:

```
sudo mkdir /usr/lib/jvm/java-1.8.0-openjdk-armhf/jre/lib/arm/server
sudo ln -s /usr/lib/jvm/java-8-openjdk-armhf/jre/lib/arm/client/libjvm.so /usr/lib/
↪jvm/java-8-openjdk-armhf/jre/lib/arm/server/libjvm.so
```

Create a directory for your project

Put the arduino cmake directory inside this directory

Create a CMakeLists.txt and modify it to your needs

```
cmake_minimum_required(VERSION 2.8)
set(CMAKE_TOOLCHAIN_FILE ${CMAKE_SOURCE_DIR}/cmake/ArduinoToolchain.cmake)
set(PROJECT_NAME ardas)
project(${PROJECT_NAME})
link_directories(${CMAKE_CURRENT_SOURCE_DIR}/libraries)
set(${CMAKE_PROJECT_NAME}_SKETCH ardas.ino)
generate_arduino_firmware(${CMAKE_PROJECT_NAME}
    SERIAL cutecom @SERIAL_PORT@ -b 9600 -l
    PORT  /dev/ttyACM0
    BOARD uno
)
```

5.2 Build and upload to arduino board

Prepare the build

```
mkdir build
cd build
cmake ..
```

Compile

```
make
```

Upload

```
make upload
```

5.3 Work in Clion

Copy the .ino file inside this directory and open it with Clion

Edit the configuration

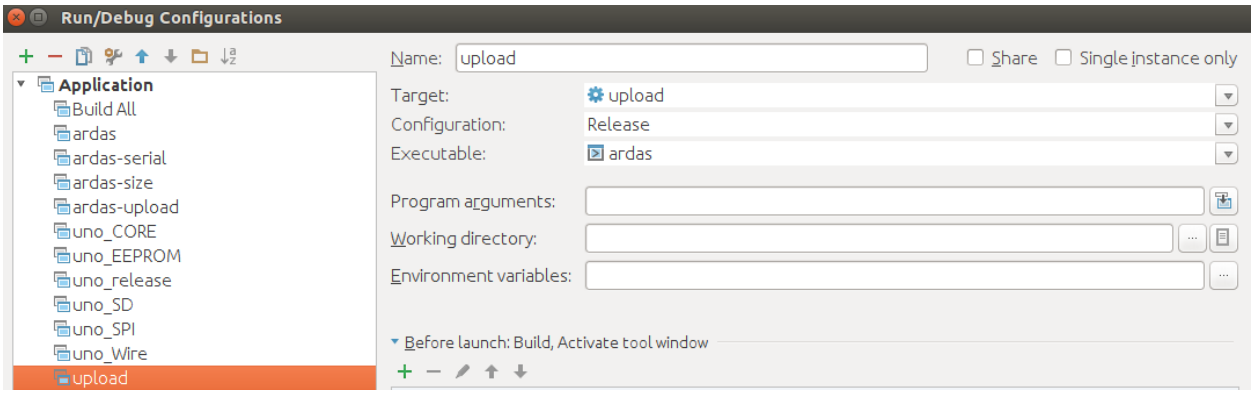

Fig. 1: Arduino Clion config

CHAPTER 6

Low level commands

Raspardas uses a set of commands to communicate with the ardas. It is possible to forward such commands through raspardas using a telnet connection using port 10001 or a direct serial connection to the ardas is also possible but not recommended. Indeed, some of those commands are intercepted by raspardas and processed before being forwarded to the ardas while other are only known to raspardas and have no effect when sent directly to an ardas.

Note : Every data sequence send by an ardas is followed by nr (LFCR) However each command end line sent to ardas must be r (CR)

- -nnn : Call Das with netID nnn
- -999 : Call all Das
- #HE : Help
- #CF :
- #DG : Toggle ardas debug mode
- #E0 : No Echo
- #E1 : Only Data
- #E2 : Data + Time
- #ND : Set naïve data mode
- #RD : Set raspardas data mode
- #RC : Toggle between raw data and calibrated data
- #SD yyyy mm dd hh nn ss : Set Date
- #SR iiii : Set Integration Period
- #SI nnn : Set Das netID
- #SS ssss : Set Station Number
- #RI : Read Info
- #RV : Read version

- #ZR station netId integrationPeriod nbInst sensor1 ... code (Ex: #ZR 1111 222 3333 4 0001 0002 0003 0004 31): Reconfig
- #KL : Stop (Kill) raspardas and closes files

Some commands are no longer supported and will be removed in a future version:

- #ZF : erase memory and set default configuration
- #XB : Full Download
- #XP : yyyy mm dd hh ss yyyy mm dd hh nn ss : Partial Download
- #XS : Stop download

CHAPTER 7

Indices and tables

- genindex
- modindex
- search