# pyaardvark Documentation

***Release 0.1***

**Kontron Europe GmbH**

**Oct 15, 2018**

# Contents

Contents:

CHAPTER 1

# Introduction

The `pyaardvark` module tries to provide a very simple API to use the Total Phase Aardvark I²C/SPI Host adatper within your python program.

## 1.1 Simple Example

In this example we access an I²C-EEPROM on address *0x50* and read the first five bytes of its content:

```
import pyaardvark

a = pyaardvark.open()
data = a.i2c_master_write_read(0x50, '\x00', 5)
# data = b'\x00\x01\x02\x03\x04'
a.close()
```

Easy, huh?

For those, who are not familiar with I²C-EEPROM accesses: You first write the offset to read from to the device (*0x00* in the example above) and then you read the desired amount of bytes from the device. The offset counter will automatically be incremened. Therefore, in the example above you read the bytes at the offsets 0, 1, 2, 3 and 4. Please note, that there are byte- and word-addressable EEPROMs. In this example we assumed a byte-addressable one, because our offset is only one byte.

## 1.2 Tutorial

### 1.2.1 Opening an Aardvark device

You have three choices to open your Aardvark device. The first is the one you saw in the simple example above:

```
a = pyaardvark.open()
```

If you have only one device connected to your machine, this is all you have to do. `pyaardvark.open()` automatically uses the first device it finds.

If you have multiple devices connected, you can either use the port parameter:

```
a = pyaardvark.open(1)
```

or the serial number, which you can find on the device itself or in your USB properties of your machine:

```
a = pyaardvark.open(serial_number='1111-222222')
```

In all cases `pyaardvark.open()` returns an `pyaardvark.Aardvark` object, which then can be used to access the host adapter.

### 1.2.2 Using the context manager protocol to open an Aardvark device

All methods of the `pyaardvark.Aardvark` object can raise an `IOError`. Instead of using *try .. except .. finally .. you can use the with* statement to open the device. Closing the device will then happen automatically after the block:

```
with pyaardvark.open() as a:
    print a.api_version
# no need for a.close() here
```

### 1.2.3 Accessing your I²C and SPI devices

To issue I²C or SPI transactions you have to first configure the adapter in the corresponding output mode. Each interface, I²C or SPI, can either be GPIOs or the actual interface. So if, for example you want to use both I²C and SPI at the same time and none of them as GPIOs:

```
a.enable_i2c = True
a.enable_spi = True
```

After you enabled the I²C interface you can issue transactions on the bus:

```
a.i2c_master_write(0x50, b'\x00\x02\0x00\x00')
```

This will write adress device *0x50* and sends the byte sequence *0x00*, *0x02*, *0x00*, *0x00* to it. To read from a device use `pyaardvark.Aardvark.i2c_master_read()`. Eventually, both can be combined and issued in one transaction: `pyaardvark.Aardvark.i2c_master_write_read()`.

### 1.2.4 Closing the device

Releasing the device can be done with `pyaardvark.Aardvark.close()`:

```
a.close()
```

## 1.3 FAQ

### 1.3.1 On pyaardvark datatypes

Most parameters of the API take bytes (eg. `pyaardvark.Aardvark.i2c_master_write_read()`). Former versions of `pyaardvark` used strings, which where handled differently in Python 2 and Python 3. For this reason, `pyaardvark` now uses the bytes object to encapsulate data. For Python 2 compatibility, the bytes backport is used (`newbytes`). This simplifies the data handling because you don't have to explicitly convert the individual characters of the string to integers (using `ord()`) anymore.

> **Warning:** Therefore the following is only valid for older `pyaardvark` versions (=< 0.5).

Iterables to strings using the built-in chr function:

```
data = (0x01, 0xaf, 0xff)
data = ''.join(chr(c) for c in data) # data is '\x01\xaf\xff'
a.i2c_master_write(0x50, data)       # writes 1h, AFh, FFh to address 50h
```

To convert a character/string to a number you can use the build-in ord function:

```
data_str = a.i2c_master_read(0x50, 3) # data_str is '\xc0\x01\xff'
data = [ord(b) for b in data_str]     # data is [192, 1, 255]
```

CHAPTER 2

# API

## 2.1 Module Interface

## 2.2 Constants

## 2.3 Aardvark Object

CHAPTER 3

# Indices and tables

- genindex
- modindex
- search