MAQ®20

Industrial Data Acquisition and Control System

MA1064

MAQ20 Python API User Manual

DATAFORTH®

MAQ20 Python API User Manual
MA1064 Rev. A – April 2017

ISO9001:2008-Registered QMS

Table of Contents

About Dataforth Corporation

Our passion at Dataforth Corporation is designing, manufacturing, and marketing the best possible signal conditioning, data acquisition, and data communication products. Our mission is to set new standards of product quality, performance, and customer service. Dataforth Corporation, with more than a quarter century of experience, is the worldwide leader in Instrument Class® Industrial Electronics – rugged, high performance signal conditioning, data acquisition, and data communication products that play a vital role in maintaining the integrity of industrial automation, data acquisition, and quality assurance systems. Our products directly connect to most industrial sensors and protect valuable measurement and control signals and equipment from the dangerous and degrading effects of noise, transient power surges, internal ground loops, and other hazards present in industrial environments.

Dataforth spans the globe with more than 50 International Distributors and US Representative Companies. Our customers benefit from a team of over 130 sales people highly trained in the application of precision products for industrial markets. In addition, we have a team of application engineers in our Tucson factory ready to solve any in-depth application questions. Upon receipt of an RFQ or order, our Customer Service Department provides fast one-day delivery information turnaround. We maintain an ample inventory that allows small quantity orders to be shipped from stock.

Dataforth operates under an ISO9001:2008 quality management system.

Contacting Dataforth Corporation

Contact Method	Contact Information
E-Mail: Technical Support	techinfo@dataforth.com
Website:	www.dataforth.com
Phone:	520-741-1404 and toll free 800-444-7644
Fax:	520-741-0762
Mail:	Dataforth Corporation 3331 E. Hemisphere Loop Tucson, AZ 85706 USA

Errata Sheets

Refer to the Technical Support area of Dataforth's website (www.dataforth.com) for any errata information on this product.

1.0 System Features

The MAQ20 Data Acquisition System encompasses more than 25 years of design excellence in the process control industry. It is a family of high performance, DIN rail mounted, programmable, multi-channel, industrially rugged signal conditioning I/O and communications modules.

Instrument Class Performance

- ±0.035% Accuracy
- Industry leading ±0.3C CJC Accuracy over full operating temperature range
- Ultra low Zero and Span Tempco
- Over-range on one channel does not affect other channels
- 1500Vrms Channel-to-Bus Isolation
- 240Vrms Continuous Field I/O Protection
- ANSI/IEEE C37.90.1 Transient Protection
- Ventilated Communications and I/O Modules
- Industrial Operating Temperature of -40°C to +85°C
- Wide Range 7-34VDC Power
- CE Compliant, UL/CUL Listing and ATEX Compliance pending

Industry Leading Functionality

- The system is a Modbus Server and can be operated remotely with no local PC
- Up to 4GB of logged data can be transferred via FTP during real-time acquisition
- Up to 24 I/O modules, or 384 channels, per system, per 19” rack width
- Per-channel configurable for range, alarms, and other functions
- Backbone mounts within DIN rail and distributes power and communications
- System firmware automatically registers the installation and removal of I/O modules
- I/O modules can be mounted remotely from the Communications Module
- Equal load sharing power supply modules allow for system expansion
- Hot Swappable I/O modules with Field-side pluggable terminal blocks on most models
- Sophisticated package enables high density mounting in 3U increments
- DIN Rail can be mounted on a continuous flat panel or plate

Distributed Processing Enables Even More Functionality

- Output modules are programmable for user-defined waveforms
- Discrete I/O modules have seven high level functions:
  - Pulse Counter
  - Frequency Counter
  - Waveform Measurement
  - Time Between Events
  - Frequency Generator
  - PWM Generator
  - One-Shot Pulse Generator

Multiple Software Options

- Free Configuration Software
- Intuitive Graphical Control Software
  - ReDAQ Shape Graphical HMI Design & Runtime Solution
  - IPEmotion Muli-Vendor and Multi-Language Solution
  - Programming examples and LabVIEW VIs
  - OPC Server

2.0 System Description and Documentation

A MAQ20 Data Acquisition System must have as a minimum a Communications Module, a Backbone, and one I/O Module. Examples include:

MAQ20-COMx Communications Module with Ethernet, USB and RS-232 or RS-485 Interface

MAQ20-DIOx Discrete Input / Output Module

MAQ20-xTC Type x Thermocouple Input Module

MAQ20-mVxN, -VxN Voltage Input Module

MAQ20-IxN Process Current Input Module

MAQ20-IO, -VO Process Current Output and Process Voltage Output Module

MAQ20-BKPLx x Channel System Backbone

Refer to www.dataforth.com/maq20.aspx for a complete listing of available modules and accessories.

System power is connected to the Communications Module, which in turn powers the I/O modules. For systems with power supply requirements exceeding what the Communications Module can provide, the MAQ20-PWR3 Power Supply module is used to provide additional power. When a MAQ20 I/O module is inserted into a system, module registration occurs automatically, data acquisition starts, and data is stored locally in the module. The system is based on a Modbus compatible memory map for easy access to acquired data, configuration settings and alarm limits. Information is stored in consistent locations from module to module for ease of use and system design.

MAQ20 modules are designed for installation in Class I, Division 2 hazardous locations and have a high level of immunity to environmental noise commonly present in heavy industrial environments.

MAQ20 communications modules provide connection between a host computer and a MAQ20 Data Acquisition System over Ethernet, USB, RS-485 or RS-232. Ethernet communications use the Modbus TCP protocol, USB communications are based on the Modbus RTU protocol, and RS-485 and RS-232 communications use the Modbus RTU protocol. Serial communications over RS-485 can be either 2-wire or 4-wire. Each MAQ20-COMx module can interface to up to 24 MAQ20 I/O modules in any combination allowing high channel counts and great flexibility in system configuration. A removable microSD card can be used by the MAQ20-COMx module to log data acquired from the MAQ20 I/O modules.

For details on hardware installation, configuration, and system operation, refer to the manuals and software available for download from www.dataforth.com/maq20_download.aspx This includes, but is not limited to:

MA1036 MAQ20 Quick Start Guide
MA1040 MAQ20 Communications Module Hardware User Manual
MA1041 MAQ20 milliVolt, Volt and Current Input Module Hardware User Manual
MA1037 MAQ20 Configuration Software Tool User Manual
MA1038 MAQ20 ReDAQ Shape for MAQ20 User Manual

MAQ20-940/-941 ReDAQ Shape Software for MAQ20 – Developer Version/User Version
MAQ20-945 MAQ20 Configuration Software Tool
MAQ20-952 IPEMotion Software for MAQ20

3.0 Introduction

The MAQ20 Python API uses an object-oriented approach for communicating with MAQ20 systems, which provides an intuitive interface where the low-level Modbus commands are hidden from normal use. Users can focus on solving the measurement problems at hand, instead of re-inventing how to communicate with modules.

Before using the MAQ20 Python API, you may find it helpful to be acquainted with basic Python programming and understand the concept of objects.

What the API does for you:

- **Communication:** to MAQ20 systems from a host PC.
- **Address offsetting:** When a module is registered in a system, addresses are offset by 2000 * R, where R is the Registration Number. The API uses relative addressing for the modules, this means that when using low-level Modbus commands, 0 to 1999 are valid addresses.
- **Counts to Engineering units:** the API reads information it needs to know at initialization to do this conversion.
- **Rounding:** when dealing with small quantities such as millivolts it can get tedious to look at a bunch of trailing decimals. The API does meaningful rounding to any module up to 3 significant units of Eng. units per count.

Note: The file names of the examples in this manual are given. Run the examples using your preferred IDE or by navigating to the examples folder '*maq20/maq20/examples*' in a terminal application and type:

python 'example name'

4.0 Installation

Requirements:

- **Python 3:** This API was tested using Python 3.4 and up on Windows 7 32 bit, Windows 10 64 bit, Ubuntu 64 bit, and Raspbian 64 bit.
- **Python Modbus library:** The MAQ20 API needs *one* of the following Modbus libraries:
  - **uModbus:** install using command:
    pip install umodbus
  - **pymodbus3:** install using command:
    pip install pymodbus3

Examples only requirements:

- PyQt5
- xlsxwriter
- pyqtgraph

Two common approaches in using the API in projects are:
1. Install along the python interpreter, this makes the API available for all projects.
2. Copying into the project folder, this will only make it available for that project only.

Install with python

Windows:

Execute the file maq20-0.5.1.win-amd64.msi OR install from source.

All Platforms:

Install from source:

1. Unzip the maq20-x.x.x.zip folder.
2. Open a terminal or command prompt and navigate to the maq20-x.x.x folder.
3. Run the command: python setup.py install

To install a new version: delete the maq20 folder the python installation folder:

1. Python/Lib/site-packages/maq20

AND

2. Python/Lib/site-packages/maq20-x.x.x-py3.x.egg-info

Then run the installer again.

Install for use with single project:

Locate the folder named maq20 folder inside the maq20-x.x.x folder. Copy the maq20 folder to the location of the project.
Note: be aware that the importing modules from the API is now relative to your project location instead of the python path.

5.0 Initialization

First, the current python module needs to load the MAQ20 API, the API can be imported the following way:

```
from maq20 import MAQ20
```

Now the maq20 system needs to be initialized, the constructor has the following signature:

```
__init__(self, ip_address="192.168.128.100", port=502)
```

To call the constructor, type:

```
maq20 = MAQ20(ip_address="192.168.128.100", port=502)
```

If no exception was raised, then the MAQ20 system was initialized correctly and the maq20 variable holds a reference to it.

To see general information about the system in the command line, type:

```
print(maq20)
```

Example of a system with 4 modules (sales demo kit):

File name: *print_system_information.py*

Code:

```
1.  from maq20 import MAQ20
2.  system0 = MAQ20(ip_address="192.168.128.100", port=502)
3.  print(system0)
```

Output:

```
1.  MAQ20-COM4
2.  Registration Number: 0
3.  Serial Number -----: S0000000-01
4.  Date Code ---------: D0217
5.  Firmware Revision -: F1.33
6.  Input  Channels ---: -1
7.  Output Channels ---: -1
8.
9.  MAQ20-JTC
```

```
Registration Number: 1
Serial Number -----: S0086701-05
Date Code ---------: D1015
Firmware Revision -: F2.54
Input  Channels ---: 8
Output Channels ---: 0

MAQ20-VDN
Registration Number: 2
Serial Number -----: S0107000-03
Date Code ---------: D0116
Firmware Revision -: F2.62
Input  Channels ---: 8
Output Channels ---: 0

MAQ20-VO
Registration Number: 3
Serial Number -----: S0074061-31
Date Code ---------: D1015
Firmware Revision -: F2.65
Input  Channels ---: 0
Output Channels ---: 8

MAQ20-DIOL
Registration Number: 4
Serial Number -----: S0080710-05
Date Code ---------: D0513
Firmware Revision -: F1.12
Input  Channels ---: 5
Output Channels ---: 5
```

6.0 Getting a module reference:

Common Practice using the API is to get a reference to a module to use its functions.
The following MAQ20 functions are provided.

Return Type	Definition and description
MAQ20Module	**find(self, name_or_sn: str)** Attempts to find a module by name or serial number.
MAQ20Module	**get_module(self, registration_number: int)** Returns the MAQ20Module with the registration number requested.
COMx	**get_com(self)** Return COMx module currently registered in this system.
list(MAQ20Module)	**get_module_list(self)** The current module list that the MAQ20 object holds.

Example:

File name: *find_module_in_system.py*

Code:

```python
from maq20 import MAQ20

module_to_find = "VDN"
maq20 = MAQ20(ip_address="192.168.128.100", port=502)
```

```
module = maq20.find(module_to_find)
if module is None:  # Check if module was not found
    raise ValueError("Module not found")  # if not found, raise error.
print(module)  # print module to see information about it.
```

Output:

```
MAQ20-VDN
Registration Number: 4
Serial Number -----: S0098692-16
Date Code ---------: D0415
Firmware Revision -: F2.62
Input  Channels ---: 8
Output Channels ---: 0
```

7.0 Reading Data functions:

The following table contains the read data functions for MAQ20Modules:

Return Type	Definition and description
list(float)	**read_data(self, start_channel=0, number_of_channels=1):** Reads data from a sequential number of channels.
float	**read_channel_data(self, channel):** Reads data from a single channel.
list(float)	**read_data_minimum(self, start_channel=0, number_of_channels=1):** Reads minimum data from a sequential number of channels.
float	**read_channel_data_minimum(self, channel):** Reads minimum data from a single channel.
list(float)	**read_data_maximum(self, start_channel=0, number_of_channels=1):** Reads maximum data from a sequential number of channels.
float	**read_channel_data_maximum(self, channel):** Reads maximum data from a single channel.
list(float)	**read_data_average(self, start_channel=0, number_of_channels=1):** Reads average data from a sequential number of channels.
float	**read_channel_data_average(self, channel=0):** Reads average data from a single channel.
list(float): size 8	**read_data_history(self, channel):** Reads data history from a single channel.

Note: versions of all these functions that return raw counts are provided. Just append '_counts' to the name. Return type is *int* or *list(int).*

Example of reading data live and printing it to console:

File name: *love_reading_data.py*

Code:

```python
from maq20 import MAQ20
import time

module_to_use = "VDN"  # change this to a module in your system. ("VSN", "VDN", etc)
delay_s = 0.1  # amount in seconds the script waits before reading another sample.


maq20 = MAQ20(ip_address="192.168.128.100", port=502)

a_module = maq20.find(module_to_use)

if a_module is None:  # Check if module was found.
    raise TypeError("Module not found.")

while True:
    print(a_module.read_data(0, number_of_channels=a_module.get_number_of_channels()))

    time.sleep(delay_s)
```

Output:

```
[1.59861, 1.59861, 1.59736, 1.59736, 0.00249, -0.00249, 0.0, 0.0]
[1.59861, 1.59861, 1.59736, 1.59861, 0.00249, -0.00374, -0.00125, -0.00125]
[1.59861, 1.59736, 1.59736, 1.59861, 0.00125, -0.00249, 0.0, -0.00125]
```

8.0 Reading data Python List notation:

The MAQ20 Python API supports Python list-like behavior for reading and writing data. Indexing, slicing, iteration, and negative indexing are available.

Example of reading data with Python list notation:

File name: *module_read_list_notation.py*

Code:

```python
from maq20 import MAQ20

maq20 = MAQ20()
a_module = maq20[1]  # Module registered at slot 1, same as a_module = maq20.get_module(1)

# indexing
print('Channel 0 : a_module[0] : {}'.format(a_module[0]))

print('')
# iteration
print('for channel in a_module:')
for channel in a_module:
    print(channel)

print('')
print('for i in range(len(a_module)):')
for i in range(len(a_module)):
    print('Channel {} : {}'.format(i, a_module[i]))

print('')  # new line
# slicing
print('Slice from 1 to 4 : a_module[1:5] : {}'.format(a_module[1:5]))
```

```
print('All Channels : a_module[:] : {}'.format(a_module[:]))

# negative indexing
print('')  # new line
print('Last Channel : a_module[-1] : {}'.format(a_module[-1]))
```

Output:

```
Channel 0 : a_module[0] : -1113.26

for channel in a_module:
-1113.26
-1113.26
-1113.26
-1113.26
-1113.26
-1113.26
82.576
25.243

for i in range(len(a_module)):
Channel 0 : -1113.26
Channel 1 : -1113.26
Channel 2 : -1113.26
Channel 3 : -1113.26
Channel 4 : -1113.26
Channel 5 : -1113.26
Channel 6 : 82.576
Channel 7 : 25.243

Slice from 1 to 4 : a_module[1:5] : [-1113.26, -1113.26, -1113.26, -1113.26]
All Channels : a_module[:] : [-1113.26, -1113.26, -1113.26, -1113.26, -1113.26, -
1113.26, 82.576, 25.243]

Last Channel : a_module[-1] : 25.243
```

9.0 Write Data:

These functions, like read_data, operate with real engineering units instead of counts. Count version are available by appending '*_counts*' to the name of the function.

Return Type	Definition and description
None	**write_data(self, start_channel, data_set):** *Writes data set to the module starting at channel start_channel, data_set has to be iterable.*
Modbus response	**write_channel_data(self, channel, data):** *Writes data to channel.*

Example using voltage output module

File name: *module_write_data_functions.py*

Code:

```
from maq20 import MAQ20

```

```
maq20 = MAQ20(ip_address="192.168.128.100", port=502)  # Initialize
vo = maq20.find("VO")  # get a reference to the module by name

if vo is None:  # check if module was found
    raise TypeError("Module was not found")

initial_value = vo.read_channel_data(3)  # Read initial value
print('Initial output value: {}'.format(initial_value))

vo.write_channel_data(channel=3, data=3.3)  # write using write_data()
print('Output value after writing: {}'.format(vo.read_channel_data(3)))

vo.write_channel_data(channel=3, data=initial_value)  # write back initial value.
 (start_channel=3, number_of_channels=1))
```

Output:

```
Initial output value: 0.0
Output value after writing: 3.30176
```

10.0 Write Data List Notation:

Indexing and slicing are available for writing data to output modules.

Example of write data list notation:

File name: *output_module_list_notation.py*

Code:

```
from maq20 import MAQ20

maq20 = MAQ20()  # Initialize system with default parameters ip_address='192.168.128.10
0', port=520
output_module = maq20.find("VO")  # get a reference to the output module by finding by
name.

if output_module is None:  # check if the module was found.
    raise TypeError('Module not found')

initial_state = output_module[:]  # Read the current state of the channels.
print('Initial state = output_module[:]\n{}'.format(initial_state))  # print its curren
t state

print('\nWriting to all channels at once:')
"""
The following command construct a list by using the built-
in len() function and then using a for loop
to write a list that looks like:
[3.3, 3.3, 3.3, ..., 3.3]
"""
output_module[:] = [3.3 for _ in range(len(output_module))]
print('output_module[:] = [3.3 for _ in range(len(output_module))]\n{}'.format(output_m
odule[:]))  # print the state again.

print('\nWrite to one channel:')
output_module[0] = 1.2
```

```
print('output_module[0] = 1.2\n{}'.format(output_module[:]))  # print the state again.

print('\nWrite to a subset of channels:')
output_module[2:6] = [5, 5, 5, 5]
print('output_module[2:6] = [5, 5, 5, 5]\n{}'.format(output_module[:]))  # print the st
ate again.

output_module[:] = initial_state  # Write back initial state.
```

***Output**:*

```
Initial state = output_module[:]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Writing to all channels at once:
output_module[:] = [3.3 for _ in range(len(output_module))]
[3.30176, 3.30176, 3.30176, 3.30176, 3.30176, 3.30176, 3.30176, 3.30176]

Write to one channel:
output_module[0] = 1.2
[1.19971, 3.30176, 3.30176, 3.30176, 3.30176, 3.30176, 3.30176, 3.30176]

Write to a subset of channels:
output_module[2:6] = [5, 5, 5, 5]
[1.19971, 3.30176, 4.99878, 4.99878, 4.99878, 4.99878, 3.30176, 3.30176]
```

11.0 Modules with special functions:

Some modules provide more functionality beyond reading and writing data. To work with these modules, import them into the current module in the following way:

from maq20.modules.'module to use' import 'Module To Use'

The current list of these modules is the following, with more to come as the API is developed:

1. DIOL
2. BRDG
3. ISOx

For details, refer to the manuals located in the MAQ20 Software & User Manual Download Center

To initialize the module, its constructor needs to be called. This constructor takes a MAQ20Module returned by the system using one of the functions from section 6.0

Example using the DIOL module to output a frequency.

Code:

```
from maq20 import MAQ20
from maq20.modules.diol import DIOL

system0 = MAQ20(ip_address="192.168.128.100", port=502)
module_name = 'DIOL'
diol = system0.find(module_name)

if diol is not None:  # Check if module was found
    diol = DIOL(maq20_module=diol)
```

```
else:
    raise TypeError("Module not found.")

diol.write_special_function_5_frequency_generator(timer=0, frequency=500)  # frequency generator 500 Hz

print(diol.read_special_function_information(timer=0))  # read back the special function settings using timer 0
```

Output:

```
Function                    : 5 = Frequency Generator
Arm/Disarm                  : 1
Frequency                   : 500
```

12.0 read vs get:

At Initialization, the maq20 API interrogates all modules and saves information in the host's RAM. This information includes name, inputs, outputs, serial number, ranges information, channel's active range, and more.

Because of this, it is worth mentioning that many times you can get information without having to perform Modbus requests. These requests can be slow due to having to wait for the network and Modbus request response.

'get' functions return immediately without doing reads because data is stored in RAM already.

Example

File name: *read_vs_get_example.py*

Code:

```
from maq20 import MAQ20
import time

maq20 = MAQ20(ip_address="192.168.128.100", port=502)
module = maq20.get_module(1)  # get module with a registration number of 1. maq20[1] returns the same thing.

if module is None:  # check if module is found
    raise TypeError("Module not found")

read_start_time = time.time()  # record current time
result = module.read_data(start_channel=0, number_of_channels=module.read_input_channels())  #perform modbus request.
read_time = time.time() - read_start_time  # record time taken by subtracting current time with initial time.

# Repeat for 'get' function.
get_start_time = time.time()
result_get = module.read_data(start_channel=0, number_of_channels=module.get_number_of_channels())
```

```
get_time = time.time() - get_start_time

# print results (seconds)
print('Read time (seconds): {}'.format(read_time))
print('Get time (seconds):  {}'.format(get_time))
_data(start_channel=3, number_of_channels=1))
```

Output:

```
Read time (seconds): 0.007004737854003906
Get time (seconds):  0.004002809524536133
```

13.0 Read/Write registers directly:

While almost anything you can do with the MAQ20 system is exposed through the API functions, once in a while there may be a feature missing. This means that we need to read/write a register directly.
To do this, the functions read_register/s, write_registers are provided. Refer to the module's manual and register Address map for details on what reading or writing a register does.

Return Type	Definition and description
int	**read_register(self, address):** *Performs a modbus read register request to the MAQ20*
List(int)	**read_registers(self, address, number_of_registers):** *Performs a modbus read registers request to the MAQ20*
Modbus response	**write_register(self, address, value):** *Performs a modbus write register request to the MAQ20*
Modbus response	**write_registers(self, address, values=None):** *Performs a modbus write registers request to the MAQ20*

maq20.MAQ20 does not do address offsetting.
maq20.MAQ20Module does address offsetting making this a relative read/write register.

Example relative vs absolute register reading:

File name: *relative_read_write_registers.py*

Code:

```
from maq20 import MAQ20

maq20 = MAQ20()
module_1 = maq20[1]  # same as maq20.get_module(1)
module_2 = maq20[2]  # same as maq20.get_module(2)

print('System level:')
print('maq20.read_registers(2000, 10) : {}'.format(maq20.read_registers(2000, 10)))
print('maq20.read_registers(4000, 10) : {}'.format(maq20.read_registers(4000, 10)))

print('\nModule level:')
# Show that the result is different.
```

```
print('module_1.read_registers(0, 10) : {}'.format(module_1.read_registers(0, 10)))  # this is equivalent to address 2000 to 2010
print('module_2.read_registers(0, 10) : {}'.format(module_2.read_registers(0, 10)))  # this is equivalent to address 4000 to 4010
```

Output:

```
System level:
maq20.read_registers(2000, 10) : [77, 65, 81, 50, 48, 45, 74, 84, 67, 32]
maq20.read_registers(4000, 10) : [77, 65, 81, 50, 48, 45, 68, 73, 79, 76]

Module level:
module_1.read_registers(0, 10) : [77, 65, 81, 50, 48, 45, 74, 84, 67, 32]
module_2.read_registers(0, 10) : [77, 65, 81, 50, 48, 45, 68, 73, 79, 76]
```

Utilities module:

The MAQ20 API includes a module full of static functions that have operation commonly done before or after reading/writing to a register in the MAQ20 system. To use this module, import it as such: import maq20.utilities as <name>

To call functions inside this module: <name>.'function name'

Functions

- def **signed16_to_unsigned16** (number)
- def **unsigned16_to_signed16** (number)
- def **response_to_string** (int_array)
- def **compute_crc** (data)
- def **check_crc** (data, check)
- def **try_except** (success, failure, exceptions)
- def **int16_to_int32** (numbers, msb_first=True)
- def **int32_to_int16s** (number, msb_first=True)
- def **ints_to_float** (numbers)
- def **float_to_ints** (number)
- def **round_to_n** (x, n)
- def **counts_to_engineering_units** (counts, p_fs, n_fs, p_fs_c, n_fs_c)
- def **engineering_units_to_counts** (eng_value, p_fs, n_fs, p_fs_c, n_fs_c)
- def **engineering_units_to_counts_dict_input** (in_val, range_information)
- def **counts_to_engineering_units_dict_input** (counts, range_information)

Example:

File name: *utils_example.py*

Code:

```
from maq20 import MAQ20
import maq20.utilities as utils

maq20 = MAQ20(ip_address="192.168.128.100", port=502)

com = maq20.get_com()  # get a reference to the COM module in the MAQ20 system

```

```
8.  # The name of a module is stored in the first 15 registers
9.  name_raw = com.read_registers(0, 15)
10. name = utils.response_to_string(com.read_registers(address=0, number_of_registers=11))

11. print('Raw name: {}'.format(name_raw))
12. print('utils.response_to_string(name_raw) : {}'.format(name))
13.
14. print('')  # new line
15. # Writing a number bigger than 16 bits: 2^16-1 = 65535
16. com.write_registers(address=1368, values=utils.int32_to_int16s(987134))
17. # Reading a number bigger than 16 bits:
18. read_back = utils.int16_to_int32(com.read_registers(address=1368, number_of_registers=2
    ))
19. print('Wrote 32 bit number 987134 and we read: {}'.format(read_back))
```

Output:

```
1.  Raw name: [77, 65, 81, 50, 48, 45, 67, 79, 77, 52, -1, -1, -1, -1, -1]
2.  utils.response_to_string(name_raw) : MAQ20-COM4
3.
4.  Wrote 987134 and we read: 987134
```

14.0 Exploring the API:

Most important functions have docstrings, this means that calling the built-in help() function will display the docstrings.
The dir() function gives you a list of attributes for an object.
Most IDEs have functionality to expose these elements, so explore your IDE of choice for handy functions like that.

Example: Using help() and dir()

File name: *help_and_dir_functions.py*

Code:

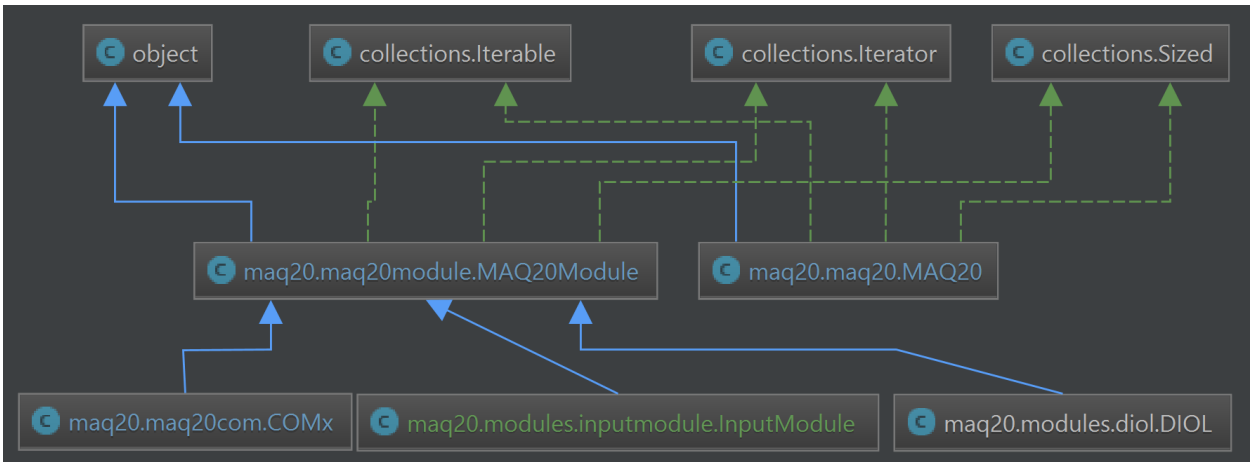
```
1.  from maq20 import MAQ20
2.
3.  maq20 = MAQ20()
4.  module_1 = maq20[1]
5.
6.  print(dir(maq20))
7.  print(dir(module_1))
8.
9.  print(help(maq20))
10. print(help(module_1))
```

Output:

Output is Omitted for this example as it is too long, run the example opening a terminal in maq20/maq20/examples/ and type:

python help_and_dir_functions.py

15.0 API Structure - Class Diagram:



The API is designed in a layered structure where the lower a module is, the more MAQ20 module-specific it is.

Layer 1 (Top):

This is all python built in classes that the API inherits from. This makes every component of the API a python object as well as a collection and iterable for Python List-like behavior.

Layer 2:

The *maq20.maq20.MAQ20* module acts as a container that holds *maq20.maq20module.MAQ20Module* modules in the system. *MAQ20Module* contains common functionality between all modules that gets passed down to bottom layers through inheritance. Standard List and Iterator magic methods are implemented.

Layer 3(Bottom):

Module configuration and COMx modules are implemented. This layer contains read/write data functions, Modbus communication functions and module-specific functions.

For a complete list of function names and descriptions available in each module, refer to *'Placeholdername'* document.

16.0 Examples:

The MAQ20 Python API is distributed with more examples that were not covered in this User Manual. The examples are located under maq20 -> examples.

Note: Some examples require additional libraries and MAQ20 hardware. Examples can be modified without breaking any API functionality. Modify examples to work with modules other than the ones used in the example.

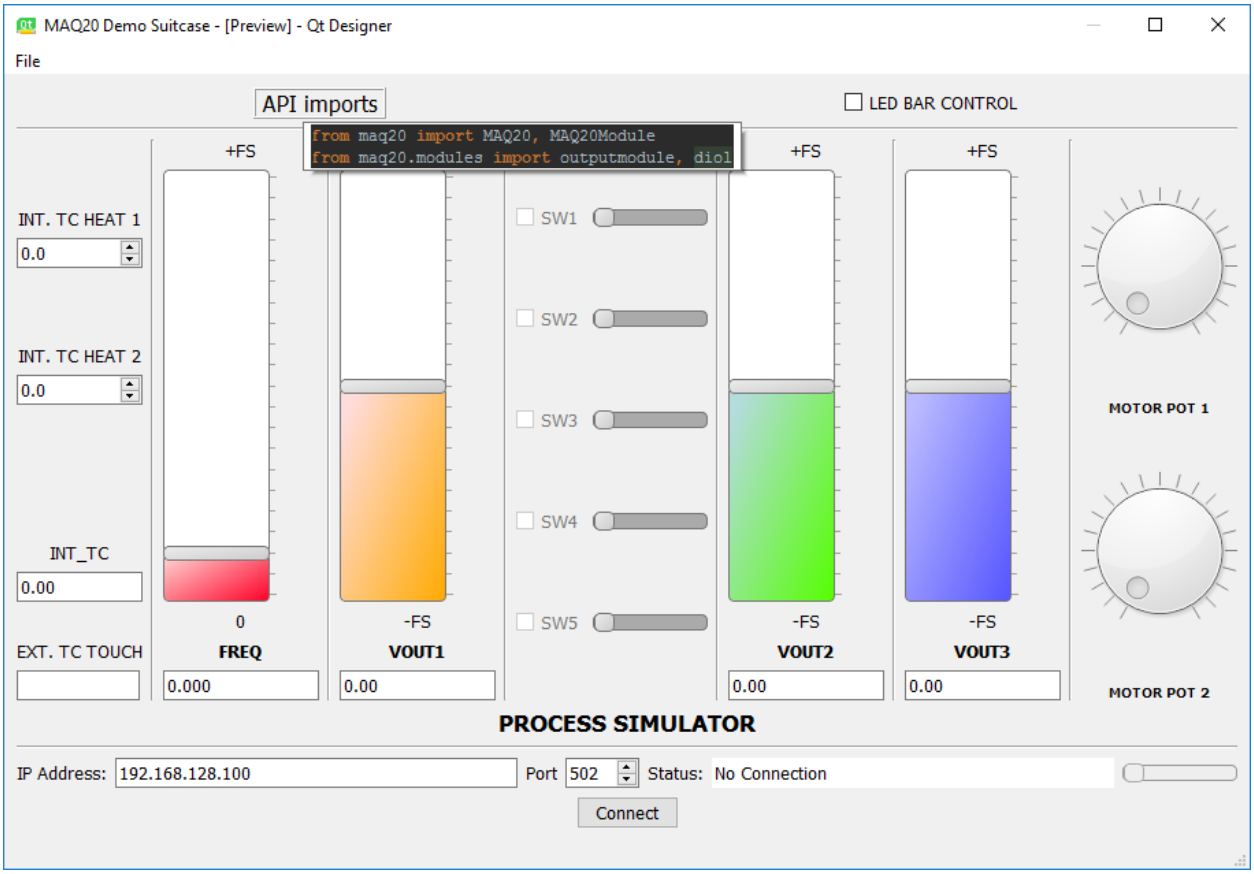
List of examples available:

1. changing_com_settings
2. demo_suitcase
3. demo_suitcase_ui
4. find_module_in_system
5. help_and_dir_functions
6. live_reading_data
7. module_data_logging_csv

8. module_data_logging_xlsx
9. module_iteration
10. module_live_plot
11. module_read_list_notation
12. module_write_data_functions
13. output_module_list_notation
14. print_system_information
15. read_vs_get_example
16. relative_read_write_registers
17. specific_module_operations
18. system_read_all_data
19. utils_example

Qt GUI Example:

The example demo_suitcase.py can be run to show a PyQT5 GUI application that is designed for learning the API. This example interfaces to the Process Simulator hardware by Dataforth, but it will still run. Tooltips show the API call behind a GUI element. And, by using Shift+F1 a “What’s This?” context menu will appear with information about hardware and channel connections to the MAQ20.



To run the example, make sure you have the PyQT5 library installed, and then open a terminal under the examples folder: maq20/maq20/examples/

Run the command: *python demo_suitcase.py*

17.0 References

Dataforth MAQ20 Software Download Center
MAQ20 Configuration Software Tool
ReDAQ Shape Software for MAQ20
MAQ20 Hardware and Software User Manuals

Python
https://www.python.org/
https://www.python.org/doc/

DATAFORTH WARRANTY
Applying to Products Sold by Dataforth Corporation

a. **General**. Dataforth Corporation ("Dataforth") warrants that its products furnished under this Agreement will, at the time of delivery, be free from defects in material and workmanship and will conform to Dataforth's applicable specifications or, if appropriate, to buyer's specifications accepted in writing by Dataforth. DATAFORTH'S OBLIGATION OR LIABILITY TO BUYER FOR PRODUCTS WHICH DO NOT CONFORM TO THE ABOVE STATED WARRANTY SHALL BE LIMITED TO DATAFORTH, AT DATAFORTH'S SOLE DISCRETION, EITHER REPAIRING, REPLACING, OR REFUNDING THE PURCHASE PRICE OF THE DEFECTIVE PRODUCT(S) PROVIDED THAT WRITTEN NOTICE OF SAID DEFECT IS RECEIVED BY DATAFORTH WITHIN THE TIME PERIODS SET FORTH BELOW:

i. for all software products including licensed programs, thirty (30) days from date of initial delivery;

ii. for all hardware products including complete systems, one (1) year from date of initial delivery;

iii. for all special products, sixty (60) days from date of initial delivery; and

further, all products warranted hereunder for which Dataforth has received timely notice of nonconformance must be returned FOB to Dataforth's plant in Tucson, Arizona USA within thirty (30) days after the expiration of the warranty periods set forth above.

The foregoing warranties shall not apply to any products which Dataforth determines have, by buyer or otherwise, been subjected to operating and/or environmental conditions in excess of the maximum value established therefore in the applicable specifications, or any products that have been the subject of mishandling, misuse, misapplication, neglect, improper testing, repair, alteration or damage. THE PROVISIONS OF THE FOREGOING WARRANTIES EXTEND TO BUYER ONLY AND NOT TO BUYER'S CUSTOMERS OR USERS OF BUYER'S PRODUCTS. THE DATAFORTH STANDARD WARRANTY IS IN LIEU OF ALL WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR USE AND ALL OTHER WARRANTIES WHETHER EXPRESS, IMPLIED OR STATUTORY, EXCEPT AS TO TITLE. THE DATAFORTH STANDARD WARRANTY MAY BE CHANGED BY DATAFORTH WITHOUT NOTICE.

b. **Buyer Indemnity**. Buyer agrees to indemnify and hold Dataforth harmless from and against any and all claims, damages and liabilities whatsoever asserted by any person, entity, industry organization, government, or governmental agency of any country resulting directly or indirectly (i) from any acts not authorized by Dataforth in writing or any statements regarding the products inconsistent with Dataforth's product documentation or standard warranty, or (ii) from any breach or threatened breach by buyer, or by any of its employees or agents, of any term, condition or provision of this Warranty or (iii) from any warranty, representation, covenant or obligation given by buyer to any third party and not expressly provided for in this Warranty or (iv) for any non-compliance (in any form) of the products with any necessary or mandatory applicable laws, regulations, procedures, government or industry policies or requirements related to the use, sale or importation of the products. Such indemnification shall include the payment of all reasonable attorneys' fees and other costs incurred by Dataforth in defending such claim.

c. **Limitation on Damages**.

(1) IN NO EVENT SHALL DATAFORTH, ITS SUPPLIERS, LICENSORS, SERVICE PROVIDERS, EMPLOYEES, AGENTS, OFFICERS, AND DIRECTORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, COVER, ECONOMIC, PUNITIVE, ACTUAL, EXEMPLARY, CONSEQUENTIAL OR OTHER DAMAGES OF ANY NATURE INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR REVENUES, COSTS OF REPLACEMENT PRODUCTS, LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE ANY DATAFORTH PRODUCT.

(2) IN NO EVENT SHALL DATAFORTH BE LIABLE FOR DIRECT, SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY NATURE RESULTING FROM BUYER'S NONCOMPLIANCE (IN ANY FORM) WITH ALL NECESSARY OR MANDATORY APPLICABLE LAWS, REGULATIONS, PROCEDURES, GOVERNMENT POLICIES OR REQUIREMENTS RELATED TO THE USE, SALE OR IMPORTATION OF PRODUCTS.

(3) IN NO EVENT WILL THE COLLECTIVE LIABILITY OF DATAFORTH AND ITS SUPPLIERS, LICENSORS, SERVICE PROVIDERS, EMPLOYEES, AGENTS, OFFICERS, AND DIRECTORS TO ANY PARTY (REGARDLESS OF THE FORM OF ACTION, WHETHER BASED UPON WARRANTY, CONTRACT, TORT, OR OTHERWISE) EXCEED THE GREATER OF EITHER US$1000.00 (ONE THOUSAND DOLLARS U.S.A. CURRENCY) OR THE AMOUNT PAID TO DATAFORTH FOR THE APPLICABLE PRODUCT OR SERVICE OUT OF WHICH LIABILITY AROSE.

(4) DATAFORTH'S LIABILITY ARISING OUT OF THE PRODUCTION, SALE OR SUPPLY OF PRODUCTS OR THEIR USE OR DISPOSITION, WHETHER BASED UPON WARRANTY, CONTRACT, TORT OR OTHERWISE, SHALL NOT EXCEED THE GREATER OF EITHER US$1000.00 (ONE THOUSAND DOLLARS U.S.A. CURRENCY) OR THE ACTUAL PURCHASE PRICE PAID BY BUYER FOR DATAFORTH'S PRODUCTS. DATAFORTH'S LIABILITY FOR ANY CLAIM OF ANY KIND SHALL IN NO CASE EXCEED THE OBLIGATION OR LIABILITY SPECIFIED IN THIS WARRANTY.

d. **Technical Assistance**. Dataforth 's Warranty as hereinabove set forth shall not be enlarged, diminished or affected by, and no obligation or liability shall arise or grow out of, Dataforth's rendering of technical advice, facilities or service in connection with buyer's order of the products furnished hereunder.

e. **Warranty Procedures**. Buyer shall notify Dataforth of any products which it believes to be defective during the applicable warranty period and which are covered by the Warranty set forth above. Buyer shall not return any products for any reason without the prior authorization of Dataforth and issuance of a Return Material Authorization ("RMA") number. After issuance of a RMA number, such products shall be promptly returned by buyer (and in no event later than thirty (30) days after the Warranty expiration date), transportation and insurance prepaid, to Dataforth's designated facility for examination and testing. Dataforth shall either repair or replace any such products found to be so defective and promptly return such products to buyer, transportation and insurance prepaid. Should Dataforth's examination and testing not disclose any defect covered by the foregoing Warranty, Dataforth shall so advise buyer and dispose of or return the products in accordance with buyer's instructions and at buyer's sole expense, and buyer shall reimburse Dataforth for testing expenses incurred at Dataforth's then current repair rates.

f. **Repair Warranty**. Dataforth warrants its repair work and/or replacement parts for a period of ninety (90) days from receipt by buyer of the repaired or replaced products or for the remainder of the warranty period for the initial delivery of such order as set forth in paragraph a above, whichever is greater.

g. **Critical Applications**. Certain applications using Dataforth's products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications"). DATAFORTH'S PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS, SAFETY EQUIPMENT, NUCLEAR FACILITY APPLICATIONS OR OTHER CRITICAL APPLICATIONS WHERE MALFUNCTION OF THE PRODUCT CAN BE EXPECTED TO RESULT IN PERSONAL INJURY, DEATH OR SEVERE PROPERTY DAMAGE. BUYER USES OR SELLS SUCH PRODUCTS FOR USE IN SUCH CRITICAL APPLICATIONS AT BUYER'S OWN RISK AND AGREES TO DEFEND, INDEMNIFY AND HOLD HARMLESS DATAFORTH FROM ANY AND ALL DAMAGES, CLAIMS, PROCEEDINGS, SUITS OR EXPENSE RESULTING FROM SUCH USE.

h. **Static Sensitive**. Dataforth ships all product in anti-static packages. Dataforth's Warranty as hereinabove set forth shall not cover warranty repair, replacement, or refund on product or devices damaged by static due to buyer's failure to properly ground.

Application Support

Dataforth provides timely, high-quality product support. Call 1**-800-444-7644 TOLL-FREE**.

Returns/Repair Policy

All warranty and repair requests should be directed to the Dataforth Customer Service Department at (520) 741-1404. If a product return is required, request a Return Material Authorization (RMA) number. You should be ready to provide the following information:

1. Complete product model number.
2. Product serial number.
3. Name, address, and telephone number of person returning product.
4. Special repair instructions.
5. Purchase order number for out-of-warranty repairs.

The product should be carefully packaged, making sure the RMA number appears on the outside of the package, and ship prepaid to:

Dataforth Corporation
3331 E. Hemisphere Loop
Tucson, AZ 85706 USA

An RMA Request Form and instructions for processing are also found at www.dataforth.com.

MAQ20 Python API User Manual
MA1064 Rev. A – April 2017

ISO9001:2008-Registered QMS