MAQ20 Python API Reference

Dataforth Corporation
Version 0.9.0
Thu Apr 13 2017

Module Documentation

maq20 Namespace Reference

Namespaces

- **maq20**
- **maq20com**
- **maq20module**
- **utilities**
- **virtualcom**

maq20.utilities Namespace Reference

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)

Variables

- list **nums** = [0x0bcd, 0xe120]
- list **floats** = []
- list **results** = []

Detailed Description

```
This module provides a set of static functions that are meant to be used for common tasks.

Import this module by typing: from maq20.utilities import *
```

Function Documentation

def maq20.utilities.check_crc (*data*, *check*)

```
Checks if the data matches the passed in CRC

:param data: The data to create a crc16 of
:param check: The CRC to validate
:returns: True if matched, False otherwise
```

def maq20.utilities.compute_crc (*data*)

```
Computes a crc16 on the passed in string. For modbus,
this is only used on the binary serial protocols (in this
case RTU).

The difference between modbus's crc16 and a normal crc16
is that modbus starts the crc value out at 0xffff.

:param data: The data to create a crc16 of
:returns: The calculated CRC
```

def maq20.utilities.counts_to_engineering_units (*counts*, *p_fs*, *n_fs*, *p_fs_c*, *n_fs_c*)

```
Converts a counts representation of a measurement into an engineering unit
representation.
:param counts: counts to be converted
:param p fs: positive full scale
:param n fs: negative full scale
:param p_fs_c: positive full scale in counts
:param n_fs_c: negative full scale in counts
:return: float
```

def maq20.utilities.counts_to_engineering_units_dict_input (*counts*, *range_information*)

```
Wrapper of counts to engineering units() that takes a dictionary as an input.
:param counts: counts to be converted
:param range information: a dict() that contains range information, returned by
MAQ20Object.get_ranges_information
:return: number
```

def maq20.utilities.engineering_units_to_counts (*eng_value*, *p_fs*, *n_fs*, *p_fs_c*, *n_fs_c*)

```
Converts an Eng Value to the respective Count representation based on range information.
:param eng_value: number
:param p fs: positive full scale
:param n fs: negative full scale
:param p fs c: positive full scale in counts
:param n_fs_c: negative full scale in counts
:return: integer
```

def maq20.utilities.engineering_units_to_counts_dict_input (*in_val*, *range_information*)

```
Wrapper of engineering_units_to_counts() that takes a dictionary as an input.
:param in_val: eng_value to be converted
:param range information: a dict() that contains range information, returned by
MAQ20Object.get ranges information
:return: integer
```

def maq20.utilities.float_to_ints (*number*)

```
Converts a float to a list of two integers.
Used for COM module PID loop controls.
:param number: float number
:return: list of integers of size 2
```

def maq20.utilities.int16_to_int32 (*numbers*, *msb_first* = `True`)

```
This function is meant to be used when a number from the register map spans two registers.
This means that address map x is MSB, and x+1 is LSB.
```

```
:param numbers: list of the numbers. This can be the response directly from read
registers.
:param msb first: choose whether msb or lsb is first, True or False.
:return: 32 bit interpretation of input.
```

def maq20.utilities.int32_to_int16s (*number*, *msb_first* = `True`)

```
Convert a number into two small enough numbers to be 16 bit.
:param msb first:
:param number: a number to convert
:return: list of integers of length 2
```

def maq20.utilities.ints_to_float (*numbers*)

```
numbers[0]: integer part
numbers[1]: decimal part
Used for COM module PID loop controls.
:param numbers: two integers that represent a floating point number
:return: float type number
```

def maq20.utilities.response_to_string (*int_array*, *str*)

```
Utility function used to convert a low level register access to ASCII characters.
:param int array: input should be an array of integers returned by the low level register
access functions.
:return: a str composed of ASCII characters.
```

def maq20.utilities.round_to_n (*x*, *n*)

```
Round a number to 3 significant digits
:param x:
:param n:
:return:
```

def maq20.utilities.signed16_to_unsigned16 (*number*)

```
Converts negative numbers into positive numbers.
:param number: Should be a negative number. If positive the function returns the same
number
:return: Returns the unsigned 16 bit representation of a negative number.
```

def maq20.utilities.try_except (*success*, *failure*, *exceptions*)

def maq20.utilities.unsigned16_to_signed16 (*number*)

```
Convert unsigned 16 bit numbers to signed 16 bit numbers.
:param number: input number.
:return: signed number.
```

Variable Documentation

list maq20.utilities.floats = []

list maq20.utilities.nums = [0x0bcd, 0xe120]

list maq20.utilities.results = []

maq20.virtualcom Namespace Reference

Classes

- class **VirtualCOM**

Functions

- def **format_response** (response)
- def **assemble_command** (function_code, bytes_left, rest_of_message)

Function Documentation

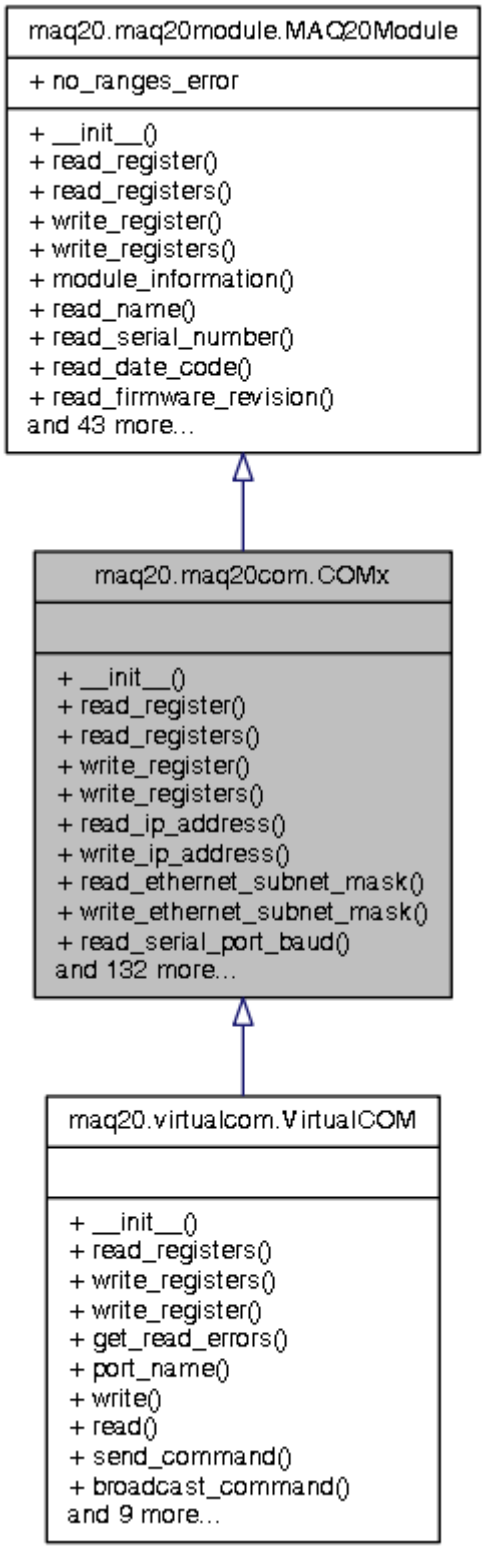
def maq20.virtualcom.assemble_command (*function_code*, *bytes_left*, *rest_of_message*)

def maq20.virtualcom.format_response (*response*)

Data Type Documentation

maq20.maq20com.COMx Class Reference

Inheritance diagram for maq20.maq20com.COMx:



Public Member Functions

- def **__init__** (self, ip_address, port)
- def **read_register** (self, address)
- def **read_registers** (self, address, number_of_registers)
- def **write_register** (self, address, value)
- def **write_registers** (self, address, values=None)
- def **read_ip_address** (self)
- def **write_ip_address** (self, ip_address)
- def **read_ethernet_subnet_mask** (self)
- def **write_ethernet_subnet_mask** (self, mask)
- def **read_serial_port_baud** (self)
- def **write_serial_port_baud** (self, baud)
- def **read_serial_port_parity** (self)
- def **write_serial_port_parity** (self, parity)
- def **read_rs485_type** (self)
- def **write_rs485_type** (self, rs485_type)
- def **read_termination** (self)
- def **write_termination** (self, termination)
- def **read_slave_id** (self)
- def **write_slave_id** (self, slave_id)
- def **write_save_port_and_server_settings** (self)
- def **read_file_server_username** (self)
- def **write_file_server_username** (self, username)
- def **read_file_server_password** (self)
- def **write_file_server_password** (self, password)
- def **read_file_server_anonymous_login** (self)
- def **write_file_server_anonymous_login** (self, input_value)
- def **read_module_status** (self)
  Module Configuration.
- def **write_module_status** (self, input_value)
- def **read_ethernet_port_present** (self)
- def **read_usb_port_present** (self)
- def **read_rs485_port_present** (self)
- def **read_rs232_port_present** (self)
- def **read_can_port_present** (self)
- def **auto_registration** (self, enable)
  Registration and Data Logger.
- def **delete_registration_numbers** (self, numbers)
- def **register_module** (self, serial_number, registration_number)
- def **read_log_file_name** (self)
- def **write_log_file_name**
- def **read_log_start_address_1** (self)
- def **write_log_start_address_1** (self, value)
- def **read_number_of_registers_to_log_1** (self)
- def **write_number_of_registers_to_log_1** (self, value)
- def **read_log_start_address_2** (self)
- def **write_log_start_address_2** (self, value)
- def **read_number_of_registers_to_log_2** (self)
- def **write_number_of_registers_to_log_2** (self, value)
- def **read_log_start_address_3** (self)
- def **write_log_start_address_3** (self, value)
- def **read_number_of_registers_to_log_3** (self)
- def **write_number_of_registers_to_log_3** (self, value)
- def **read_log_start_address_4** (self)
- def **write_log_start_address_4** (self, value)
- def **read_number_of_registers_to_log_4** (self)

- def **write_number_of_registers_to_log_4** (self, value)
- def **read_log_interval** (self)
- def **write_log_interval** (self, value)
- def **read_log_number_of_samples** (self)
- def **write_log_number_of_samples** (self, value)
- def **read_log_enable** (self)
- def **write_log_enable** (self, enable)
- def **read_card_available** (self)
- def **read_total_space** (self)
- def **read_free_space** (self)
- def **read_second** (self)
  Module RTC and Temperature.
- def **write_second** (self, second)
- def **read_minute** (self)
- def **write_minute** (self, minute)
- def **read_hour** (self)
- def **write_hour** (self, hour)
- def **read_day** (self)
- def **write_day** (self, day)
- def **read_date** (self)
- def **write_date** (self, date)
- def **read_month** (self)
- def **write_month** (self, month)
- def **read_year** (self)
- def **write_year** (self, year)
- def **read_internal_temperature_sensor** (self)
- def **read_pid_id** (self)
  PID Loop Controllers.
- def **write_pid_id** (self, input_id)
- def **read_pid_enable** (self)
- def **write_pid_enable** (self, enable)
- def **read_pid_name** (self)
- def **write_pid_name** (self, name)
- def **read_pid_description** (self)
- def **write_pid_description** (self, description)
- def **read_pid_engineering_units** (self)
- def **write_pid_engineering_units** (self, engineering_units)
- def **read_pid_pv_range_unit** (self)
- def **write_pid_pv_range_unit** (self, pv_range_unit)
- def **read_pid_co_range_unit** (self)
- def **write_pid_co_range_unit** (self, co_range_unit)
- def **read_pid_pv_modbus_address** (self)
- def **read_pid_co_modbus_address** (self)
- def **read_pid_pv_count_maximum** (self)
- def **write_pid_pv_count_maximum** (self, pv_count_maximum)
- def **read_pid_pv_count_minimum** (self)
- def **write_pid_pv_count_minimum** (self, pv_count_minimum)
- def **read_pid_co_count_maximum** (self)
- def **write_pid_co_count_maximum** (self, co_count_maximum)
- def **read_pid_co_count_minimum** (self)
- def **write_pid_co_count_minimum** (self, co_count_minimum)
- def **read_pid_pv_range_maximum** (self)
- def **write_pid_pv_range_maximum** (self, pv_range_maximum)
- def **read_pid_pv_range_minimum** (self)
- def **write_pid_pv_range_minimum** (self, pv_range_minimum)
- def **read_pid_co_range_maximum** (self)

- def **write_pid_co_range_maximum** (self, co_range_maximum)
- def **read_pid_co_range_minimum** (self)
- def **write_pid_co_range_minimum** (self, co_range_minimum)
- def **read_pid_algorithm** (self)
- def **write_pid_algorithm** (self, algorithm)
- def **read_pid_control_direction** (self)
- def **read_pid_setpoint_action** (self)
- def **write_pid_setpoint_action** (self, setpoint_action)
- def **read_pid_mode** (self)
- def **read_pid_output_type** (self)
- def **read_pid_setpoint** (self)
- def **read_pid_process_variable** (self)
- def **read_pid_control_output** (self)
- def **read_pid_pv_maximum** (self)
- def **read_pid_pv_minimum** (self)
- def **read_pid_co_maximum** (self)
- def **read_pid_co_minimum** (self)
- def **read_pid_kc** (self)
- def **read_pid_ti** (self)
- def **read_pid_td** (self)
- def **read_pid_scan_time** (self)
- def **read_pid_co_high_clamp** (self)
- def **read_pid_co_low_clamp** (self)
- def **read_pid_pv_tracking** (self)
- def **read_pid_gap_width** (self)
- def **read_pid_gap_multiplier** (self)
- def **read_pid_filter_time_constant** (self)
- def **read_pid_active_alarm** (self)
- def **read_pid_alarm_deadband** (self)
- def **read_pid_high_high_alarm_limit** (self)
- def **read_pid_high_alarm_limit** (self)
- def **read_pid_low_alarm_limit** (self)
- def **read_pid_low_low_alarm_limit** (self)
- def **ftp_settings** (self)
  Helper Functions.
- def **__del__** (self)

Additional Inherited Members

Detailed Description

```
COM Module:
Takes care of the communication backend and provides functions that read the COMx register
map.
```

Constructor & Destructor Documentation

def maq20.maq20com.COMx.__init__ (***self*****,** ***ip_address*****,** ***port*****)**

def maq20.maq20com.COMx.__del__ (***self*****)**

Member Function Documentation

def maq20.maq20com.COMx.auto_registration (*self*, *enable*)

Registration and Data Logger.

```
Enables or disables auto registration.
:param enable: Boolean
:return: response from modbus backend.
```

def maq20.maq20com.COMx.delete_registration_numbers (*self*, *numbers*)

def maq20.maq20com.COMx.ftp_settings (*self*, *str*)

Helper Functions.

def maq20.maq20com.COMx.read_can_port_present (*self*)

def maq20.maq20com.COMx.read_card_available (*self*)

def maq20.maq20com.COMx.read_date (*self*)

1-31

def maq20.maq20com.COMx.read_day (*self*)

1-7, 1 = Sunday

def maq20.maq20com.COMx.read_ethernet_port_present (*self*)

def maq20.maq20com.COMx.read_ethernet_subnet_mask (*self*)

def maq20.maq20com.COMx.read_file_server_anonymous_login (*self*)

def maq20.maq20com.COMx.read_file_server_password (*self*)

def maq20.maq20com.COMx.read_file_server_username (*self*, *str*)

def maq20.maq20com.COMx.read_free_space (*self*)

def maq20.maq20com.COMx.read_hour (*self*)

0-23

def maq20.maq20com.COMx.read_internal_temperature_sensor (*self*)

0-59, Degree C

def maq20.maq20com.COMx.read_ip_address (*self*)

def maq20.maq20com.COMx.read_log_enable (*self*)

def maq20.maq20com.COMx.read_log_file_name (*self*)

def maq20.maq20com.COMx.read_log_interval (*self*)

def maq20.maq20com.COMx.read_log_number_of_samples (*self*)

def maq20.maq20com.COMx.read_log_start_address_1 (*self*)

```
Default = 2000 (Start Address of I/O Module in Slot 1.  Data for this module is at Start
Address 3000)
:return: int
```

def maq20.maq20com.COMx.read_log_start_address_2 (*self*)

```
Default = 4000 (Start Address of I/O Module in Slot 2.  Data for this module is at Start
Address 5000)
:return: int
```

def maq20.maq20com.COMx.read_log_start_address_3 (*self*)

```
Default = 6000 (Start Address of I/O Module in Slot 3.  Data for this module is at Start
Address 7000)
:return: int
```

def maq20.maq20com.COMx.read_log_start_address_4 (*self*)

```
Default = 8000 (Start Address of I/O Module in Slot 4.  Data for this module is at Start
Address 9000)
:return: int
```

def maq20.maq20com.COMx.read_minute (*self*)

0-59

def maq20.maq20com.COMx.read_module_status (*self*)

Module Configuration.

def maq20.maq20com.COMx.read_month (*self*)

1-12

def maq20.maq20com.COMx.read_number_of_registers_to_log_1 (*self*)

```
Number of Registers to Log starting at Log Start Address 1.  Maximum = 100, Default
= 8
:return: int
```

def maq20.maq20com.COMx.read_number_of_registers_to_log_2 (*self*)

```
Number of Registers to Log starting at Log Start Address 2.  Maximum = 100, Default
= 8
:return: int
```

def maq20.maq20com.COMx.read_number_of_registers_to_log_3 (*self*)

```
Number of Registers to Log starting at Log Start Address 3.  Maximum = 100, Default
= 8
:return: int
```

def maq20.maq20com.COMx.read_number_of_registers_to_log_4 (*self*)

```
Number of Registers to Log starting at Log Start Address 4.  Maximum = 100, Default
= 8
:return: int
```

def maq20.maq20com.COMx.read_pid_active_alarm (*self*)

```
Indicates which alarm condition is active. 0 = Low-Low, 1 = Low, 2 = None, 3 = High-High,
4 = High.
:return: string
```

def maq20.maq20com.COMx.read_pid_alarm_deadband (*self*)

```
Deadband or Hysteresis.  Adds to low limits, subtracts from high limits.
Integer part at Address 1442, fractional part at Address 1443.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_algorithm (*self*)

```
PID Control Algorithm.  0 = Noninteractive, 1 = Parallel
:return: string
```

def maq20.maq20com.COMx.read_pid_co_count_maximum (*self*)

```
Control Output maximum count value.  MSB at Address 1372, LSB at Address 1373.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.read_pid_co_count_minimum (*self*)

```
Control Output minimum count value.  MSB at Address 1374, LSB at Address 1375.
```

```
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.read_pid_co_high_clamp (*self*)

```
Controller Output upper limit (%).
Integer part at Address 1418, fractional part at Address 1419.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_co_low_clamp (*self*)

```
Controller Output lower limit (%).
Integer part at Address 1420, fractional part at Address 1421.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_co_maximum (*self*)

```
Control Output maximum value.
Integer part at Address 1406, fractional part at Address 1407.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_co_minimum (*self*)

```
Control Output minimum value.
Integer part at Address 1408, fractional part at Address 1409.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_co_modbus_address (*self*)

```
System Address where Control Output is sent to.
:return: 0 to 65,535
```

def maq20.maq20com.COMx.read_pid_co_range_maximum (*self*)

```
Control Output Range maximum value.  Integer part at Address 1380, fractional part at
Address 1381.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_co_range_minimum (*self*)

```
Control Output Range minimum value.  Integer part at Address 1382, fractional part at
Address 1383.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_co_range_unit (*self*)

```
Units chosen for Control Output.  5 characters max.  Standard unit = "%".
:return: string of length 5
```

def maq20.maq20com.COMx.read_pid_control_direction (*self*)

```
Control Direction.  0 = Reverse Acting, 1 = Direct Acting
:return: string
```

def maq20.maq20com.COMx.read_pid_control_output (*self*)

```
Control Output.
Integer part at Address 1400, fractional part at Address 1401.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_description (*self*)

```
PID Controller Description, 10 characters max.
:return: string of length 10
```

def maq20.maq20com.COMx.read_pid_enable (*self*)

```
Enable/Disable Controller
:return: 0 or 1
```

def maq20.maq20com.COMx.read_pid_engineering_units (*self*)

```
Engineering Units (EU) chosen for the Controller, 5 characters max.
:return: string of length 5
```

def maq20.maq20com.COMx.read_pid_filter_time_constant (*self*)

```
PV Filter Time Constant (minutes).
Integer part at Address 1427, fractional part at Address 1428. Fractional part is in
10,000ths of a second.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_gap_multiplier (*self*)

```
Gain multiplier inside Gap.
Integer part at Address 1425, fractional part at Address 1426.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_gap_width (*self*)

```
Gap around setpoint (Engineering Units).
Integer part at Address 1423, fractional part at Address 1424.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_high_alarm_limit (*self*)

```
High Alarm Limit (Engineering Units). Integer part at Address 1446, fractional part
at Address 1447.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_high_high_alarm_limit (*self*)

```
High-High Alarm Limit (Engineering Units). Integer part at Address 1444, fractional
part at Address 1445.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_id (*self*)

PID Loop Controllers.

TODO: Implement the write functions for PID loop controllers. TODO: Document write functions.

```
Unique instance ID of Controller
:return: 0 to 31
```

def maq20.maq20com.COMx.read_pid_kc (*self*)

```
Controller Gain (%/%).
Integer part at Address 1410, fractional part at Address 1411.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_low_alarm_limit (*self*)

```
Low Alarm Limit (Engineering Units). Integer part at Address 1448, fractional part at
Address 1449.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_low_low_alarm_limit (*self*)

```
Low-Low Alarm Limit (Engineering Units). Integer part at Address 1450, fractional part
at Address 1451.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_mode (*self*)

```
Operational Mode.  0 = Manual, 1 = Automatic
:return: string
```

def maq20.maq20com.COMx.read_pid_name (*self*)

```
PID Controller name, 10 characters max.
:return: string of length 10
```

def maq20.maq20com.COMx.read_pid_output_type (*self*)

```
Control Output Signal Type.  0 = Voltage, 1 = Current, 2 = Discrete Output (PWM)
:return: string
```

def maq20.maq20com.COMx.read_pid_process_variable (*self*)

```
Process Variable.
Integer part at Address 1398, fractional part at Address 1399.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_pv_count_maximum (*self*)

```
Process Variable maximum count value.  MSB at Address 1368, LSB at Address 1369.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.read_pid_pv_count_minimum (*self*)

```
Process Variable minimum count value.  MSB at Address 1370, LSB at Address 1371.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.read_pid_pv_maximum (*self*)

```
Process Variable maximum value.
Integer part at Address 1402, fractional part at Address 1403.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_pv_minimum (*self*)

```
Process Variable minimum value.
Integer part at Address 1404, fractional part at Address 1405.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_pv_modbus_address (*self*)

```
System Address where Process Variable is obtained from.
:return: 0 to 65,535
```

def maq20.maq20com.COMx.read_pid_pv_range_maximum (*self*)

```
Process Variable Range maximum value.  Integer part at Address 1376, fractional part
at Address 1377.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_pv_range_minimum (*self*)

```
Process Variable Range minimum value.  Integer part at Address 1378, fractional part
at Address 1379.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_pv_range_unit (*self*)

```
Units chosen for Process Variable.  5 characters max.  Standard unit = "%".
:return: string of length 5
```

def maq20.maq20com.COMx.read_pid_pv_tracking (*self*)

```
Track Process Variable in Manual Mode. 0 = Do Not Track PV, 1 = Track PV
:return: string
```

def maq20.maq20com.COMx.read_pid_scan_time (*self*)

```
PID Controller Update Rate (seconds).
Integer part at Address 1416, fractional part at Address 1417.  This value is fixed
at 1s.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_setpoint (*self*)

```
Setpoint.
Integer part at Address 1396, fractional part at Address 1397.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_setpoint_action (*self*)

```
Setpoint Action.  0 = Proportional & Derivative on Error, 1 = Proportional on Error
/ Derivative on PV,
2 = Proportional & Derivative on PV.
:return: string
```

def maq20.maq20com.COMx.read_pid_td (*self*)

```
Derivative Time (minutes).
Integer part at Address 1414, fractional part at Address 1415.  Fractional part is in
10,000ths of a second.
:return: float type number
```

def maq20.maq20com.COMx.read_pid_ti (*self*)

```
Integral Time (minutes).
Integer part at Address 1412, fractional part at Address 1413.  Fractional part is in
10,000ths of a second.
:return: float type number
```

def maq20.maq20com.COMx.read_register (*self*, *address*)

```
Low level register access.
Performs a modbus read register request to the MAQ20
:param address: requested address
:return: int - [-32767, 32767]
```

def maq20.maq20com.COMx.read_registers (*self*, *address*, *number_of_registers*)

```
Low level register access.
Performs a modbus read registers request to the MAQ20
:param address: starting address.
:param number of registers: number of registers to be read in sequence.
:return: list(int) [-32767, 32767]
```

def maq20.maq20com.COMx.read_rs232_port_present (*self*)

def maq20.maq20com.COMx.read_rs485_port_present (*self*)

def maq20.maq20com.COMx.read_rs485_type (*self*)

def maq20.maq20com.COMx.read_second (*self*)

Module RTC and Temperature.

0-59

def maq20.maq20com.COMx.read_serial_port_baud (*self*, *int*)

def maq20.maq20com.COMx.read_serial_port_parity (*self*, *str*)

def maq20.maq20com.COMx.read_slave_id (*self*)

def maq20.maq20com.COMx.read_termination (*self*)

def maq20.maq20com.COMx.read_total_space (*self*)

def maq20.maq20com.COMx.read_usb_port_present (*self*)

def maq20.maq20com.COMx.read_year (*self*)

0-99

def maq20.maq20com.COMx.register_module (*self*, *serial_number*, *registration_number*)

def maq20.maq20com.COMx.write_date (*self*, *date*)

1-31

def maq20.maq20com.COMx.write_day (*self*, *day*)

1-7, 1 = Sunday

def maq20.maq20com.COMx.write_ethernet_subnet_mask (*self*, *mask*)

def maq20.maq20com.COMx.write_file_server_anonymous_login (*self*, *input_value*)

def maq20.maq20com.COMx.write_file_server_password (*self*, *password*)

def maq20.maq20com.COMx.write_file_server_username (*self*, *username*)

def maq20.maq20com.COMx.write_hour (*self*, *hour*)

0-23

def maq20.maq20com.COMx.write_ip_address (*self*, *ip_address*)

def maq20.maq20com.COMx.write_log_enable (*self*, *enable*)

def maq20.maq20com.COMx.write_log_file_name (*self*, *name*)

def maq20.maq20com.COMx.write_log_interval (*self*, *value*)

def maq20.maq20com.COMx.write_log_number_of_samples (*self*, *value*)

def maq20.maq20com.COMx.write_log_start_address_1 (*self*, *value*)

```
Default = 2000 (Start Address of I/O Module in Slot 1.  Data for this module is at Start
Address 3000)
:return: modbus response.
```

def maq20.maq20com.COMx.write_log_start_address_2 (*self*, *value*)

```
Default = 4000 (Start Address of I/O Module in Slot 2.  Data for this module is at Start
Address 5000)
:return: modbus response.
```

def maq20.maq20com.COMx.write_log_start_address_3 (*self*, *value*)

```
Default = 6000 (Start Address of I/O Module in Slot 3.  Data for this module is at Start
Address 7000)
```

```
:return: modbus response.
```

def maq20.maq20com.COMx.write_log_start_address_4 (*self*, *value*)

```
Default = 8000 (Start Address of I/O Module in Slot 4.  Data for this module is at Start
Address 9000)
:return: modbus response.
```

def maq20.maq20com.COMx.write_minute (*self*, *minute*)

0-59

def maq20.maq20com.COMx.write_module_status (*self*, *input_value*)

def maq20.maq20com.COMx.write_month (*self*, *month*)

1-12

def maq20.maq20com.COMx.write_number_of_registers_to_log_1 (*self*, *value*)

```
Number of Registers to Log starting at Log Start Address 1.  Maximum = 100, Default
= 8
:param value: number to write
:return: modbus response
```

def maq20.maq20com.COMx.write_number_of_registers_to_log_2 (*self*, *value*)

```
Number of Registers to Log starting at Log Start Address 2.  Maximum = 100, Default
= 8
:param value: number to write
:return: modbus response
```

def maq20.maq20com.COMx.write_number_of_registers_to_log_3 (*self*, *value*)

```
Number of Registers to Log starting at Log Start Address 3.  Maximum = 100, Default
= 8
:param value: number to write
:return: modbus response
```

def maq20.maq20com.COMx.write_number_of_registers_to_log_4 (*self*, *value*)

```
Number of Registers to Log starting at Log Start Address 4.  Maximum = 100, Default
= 8
:param value: number to write
:return: modbus response
```

def maq20.maq20com.COMx.write_pid_algorithm (*self*, *algorithm*)

```
PID Control Algorithm.  0 = Noninteractive, 1 = Parallel
:return: string
```

def maq20.maq20com.COMx.write_pid_co_count_maximum (***self*****,** ***co_count_maximum*****)**

```
Control Output maximum count value.  MSB at Address 1372, LSB at Address 1373.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.write_pid_co_count_minimum (***self*****,** ***co_count_minimum*****)**

```
Control Output minimum count value.  MSB at Address 1374, LSB at Address 1375.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.write_pid_co_range_maximum (***self*****,** ***co_range_maximum*****)**

```
Control Output Range maximum value.  Integer part at Address 1380, fractional part at
Address 1381.
:return: float type number
```

def maq20.maq20com.COMx.write_pid_co_range_minimum (***self*****,** ***co_range_minimum*****)**

```
Control Output Range minimum value.  Integer part at Address 1382, fractional part at
Address 1383.
:return: float type number
```

def maq20.maq20com.COMx.write_pid_co_range_unit (***self*****,** ***co_range_unit*****)**

```
Units chosen for Control Output.  5 characters max.  Standard unit = "%".
:return: string of length 5
```

def maq20.maq20com.COMx.write_pid_description (***self*****,** ***description*****)**

```
PID Controller Description, 10 characters max.
:return: string of length 10
```

def maq20.maq20com.COMx.write_pid_enable (***self*****,** ***enable*****)**

```
Enable/Disable Controller
:return: modbus response
```

def maq20.maq20com.COMx.write_pid_engineering_units (***self*****,** ***engineering_units*****)**

```
Engineering Units (EU) chosen for the Controller, 5 characters max.
:return: string of length 5
```

def maq20.maq20com.COMx.write_pid_id (*self*, *input_id*)

```
Unique instance ID of Controller
:return: modbus response
```

def maq20.maq20com.COMx.write_pid_name (*self*, *name*)

```
PID Controller name, 10 characters max.
:return: modbus response
```

def maq20.maq20com.COMx.write_pid_pv_count_maximum (*self*, *pv_count_maximum*)

```
Process Variable maximum count value.  MSB at Address 1368, LSB at Address 1369.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.write_pid_pv_count_minimum (*self*, *pv_count_minimum*)

```
Process Variable minimum count value.  MSB at Address 1370, LSB at Address 1371.
:return: 0 to 2^32-1
```

def maq20.maq20com.COMx.write_pid_pv_range_maximum (*self*, *pv_range_maximum*)

```
Process Variable Range maximum value.  Integer part at Address 1376, fractional part
at Address 1377.
:return: float type number
```

def maq20.maq20com.COMx.write_pid_pv_range_minimum (*self*, *pv_range_minimum*)

```
Process Variable Range minimum value.  Integer part at Address 1378, fractional part
at Address 1379.
:return: float type number
```

def maq20.maq20com.COMx.write_pid_pv_range_unit (*self*, *pv_range_unit*)

```
Units chosen for Process Variable.  5 characters max.  Standard unit = "%".
:return: string of length 5
```

def maq20.maq20com.COMx.write_pid_setpoint_action (*self*, *setpoint_action*)

```
Setpoint Action.  0 = Proportional & Derivative on Error, 1 = Proportional on Error
/ Derivative on PV,
2 = Proportional & Derivative on PV.
:return: string
```

def maq20.maq20com.COMx.write_register (*self*, *address*, *value*)

```
Low level register access.
Performs a modbus write register request to the MAQ20
:param address: starting address.
:param value: int [-32767, 32767] or a str of size 1
:return: modbus response.
```

def maq20.maq20com.COMx.write_registers (*self*, *address*, *values* = `None`)

```
Low level register access.
Performs a modbus write registers request to the MAQ20
:param address: starting address.
:param values: list(int) [-32767, 32767] or a str
:return: modbus response.
```

def maq20.maq20com.COMx.write_rs485_type (*self*, *rs485_type*)

def maq20.maq20com.COMx.write_save_port_and_server_settings (*self*)

```
Saves the com port and file server information into the COM's EEPROM memory.
```

Note: Changes apply after power cycle.

def maq20.maq20com.COMx.write_second (*self*, *second*)

0-59

def maq20.maq20com.COMx.write_serial_port_baud (*self*, *baud*)

def maq20.maq20com.COMx.write_serial_port_parity (*self*, *parity*)

def maq20.maq20com.COMx.write_slave_id (*self*, *slave_id*)

def maq20.maq20com.COMx.write_termination (*self*, *termination*)

def maq20.maq20com.COMx.write_year (*self*, *year*)

0-99

The documentation for this class was generated from the following file:

- maq20/**maq20com.py**

maq20.maq20.MAQ20 Class Reference

Classes

- class **StdoutCatcher**

Public Member Functions

- def **__init__** (self, ip_address="192.168.128.100", port=502, virtual_com=False)
- def **scan_module_list** (self)
- def **get_system_information** (self)
- def **get_module** (self, registration_number)
- def **get_module_list** (self)
- def **read_system_data** (self)
- def **print_system_data** (self)
- def **get_com** (self)
- def **read_data** (self, a_module=1, channel=0)
- def **find**
- def **time** (self)
- def **ftp_login** (self, username='maq20', password='1234')
- def **ftp_dir** (self)
- def **ftp_get**
- def **ftp_del**
- def **ftp_filenames** (self)
- def **setup_sd_card_logging** (self, filename, start_address1=3000, num_of_registers1=8, start_address2=5000, num_of_registers2=0, start_address3=7000, num_of_registers3=0, start_address4=9000, num_of_registers4=0, interval_ms=100, number_of_samples=100)
- def **__str__** (self)
- def **__repr__** (self)
- def **__getitem__** (self, item)
- def **__iter__** (self)
- def **__next__** (self)
- def **__len__** (self)

Detailed Description

```
"
MAQ20 System
A class that provides easy to use high level functions for MAQ20 Modules.
```

Constructor & Destructor Documentation

def maq20.maq20.MAQ20.__init__ (*self*, *ip_address* = `"192.168.128.100"`, *port* = `502`, *virtual_com* = `False`)

```
Initializes the MAQ20 with the given input parameters.
:param ip_address: a string containing the ip address of the MAQ20, default is
192.168.128.100
:param port: default is 502
```

Member Function Documentation

def maq20.maq20.MAQ20.__getitem__ (*self*, *item*)

def maq20.maq20.MAQ20.__iter__ (*self*)

def maq20.maq20.MAQ20.__len__ (*self*)

def maq20.maq20.MAQ20.__next__ (*self*)

def maq20.maq20.MAQ20.__repr__ (*self*)

def maq20.maq20.MAQ20.__str__ (*self*)

def maq20.maq20.MAQ20.find (*self*, *name_or_sn*)

def maq20.maq20.MAQ20.ftp_del (*self*, *filename*)

def maq20.maq20.MAQ20.ftp_dir (*self*)

def maq20.maq20.MAQ20.ftp_filenames (*self*)

def maq20.maq20.MAQ20.ftp_get (*self*, *filename*)

def maq20.maq20.MAQ20.ftp_login (*self*, *username* = `'maq20'`, *password* = `'1234'`)

def maq20.maq20.MAQ20.get_com (*self*)

```
:return: MAQ20COM currently registered in this system.
```

def maq20.maq20.MAQ20.get_module (*self*, *registration_number*)

```
Returns the MAQ20Module with the registration number requested.
:param registration number: int, 0 to 23
:return: MAQ20Module
```

def maq20.maq20.MAQ20.get_module_list (*self*)

```
:return: the current module list that the MAQ20 object holds
note: This does not refresh the list of registered modules.
```

def maq20.maq20.MAQ20.get_system_information (*self*)

```
:return: a string containing information about every registered module in the system.
```

def maq20.maq20.MAQ20.print_system_data (*self*)

```
Prints system data to the console
:return: None
```

def maq20.maq20.MAQ20.read_data (*self*, *a_module* = 1, *channel* = 0)

```
Calls the read_data() function registered at a_module index.
:param a module: int, 0 to 23
:param channel: int, 0 and greater.
:return: list
```

def maq20.maq20.MAQ20.read_system_data (*self*)

```
Read every channel in every module
:return: a list of lists containing the values read.
```

def maq20.maq20.MAQ20.scan_module_list (*self*)

```
Refreshes the internal list of registered modules in a system.
To get the list call function: get_module_list()
```

def maq20.maq20.MAQ20.setup_sd_card_logging (*self*, *filename*, *start_address1* = 3000, *num_of_registers1* = 8, *start_address2* = 5000, *num_of_registers2* = 0, *start_address3* = 7000, *num_of_registers3* = 0, *start_address4* = 9000, *num_of_registers4* = 0, *interval_ms* = 100, *number_of_samples* = 100)

def maq20.maq20.MAQ20.time (*self*, *str*)

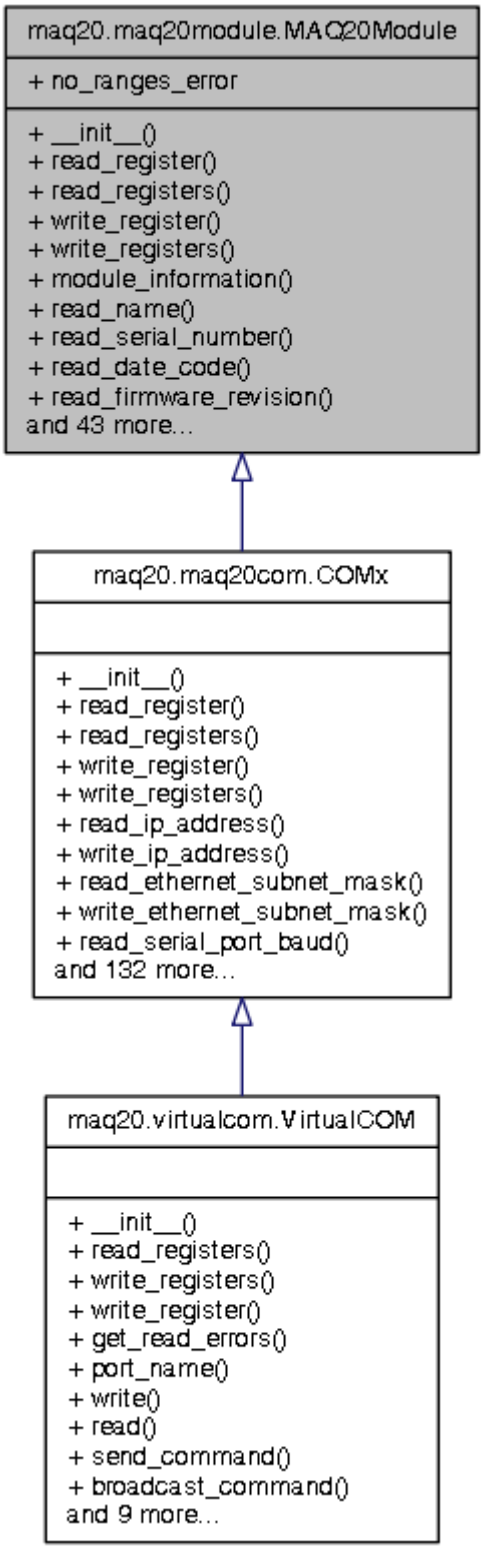
```
returns the time read from the module in a human readable form.
:return: str
```

The documentation for this class was generated from the following file:

- maq20/**maq20.py**

maq20.maq20module.MAQ20Module Class Reference

Inheritance diagram for maq20.maq20module.MAQ20Module:



Public Member Functions

- def **__init__** (self, com=None, registration_number=0)
- def **read_register** (self, address)

Modbus Communication.

- def **read_registers** (self, address, number_of_registers)
- def **write_register** (self, address, value)
- def **write_registers** (self, address, values=None)
- def **module_information** (self)
- def **read_name** (self)
- def **read_serial_number** (self)
- def **read_date_code** (self)
- def **read_firmware_revision** (self)
- def **read_input_channels** (self)
- def **read_output_channels** (self)
- def **write_module_detect** (self)
- def **write_reset_register** (self, input_value)
- def **has_range_information** (self)

  Ranges Information [1700,1899].
- def **read_range_count** (self)
- def **read_range** (self, a_range=0)
- def **load_ranges_information** (self)
- def **get_engineering_full_scale_positive** (self, channel)
- def **get_engineering_full_scale_negative** (self, channel)
- def **get_counts_full_scale_positive** (self, channel)
- def **get_counts_full_scale_negative** (self, channel)
- def **get_engineering_units** (self, channel)
- def **load_channel_active_ranges** (self)
- def **get_channel_active_range** (self, channel=0)
- def **get_ranges_information** (self)
- def **get_channel_ranges_information** (self, channel)
- def **display_ranges_information** (self)
- def **counts_to_eng_units_list** (self, data_list, channel)
- def **get_name** (self)
- def **get_registration_number** (self)
- def **get_inputs** (self)
- def **get_outputs** (self)
- def **get_number_of_channels** (self)
- def **get_serial_number** (self)
- def **get_date_code** (self)
- def **get_firmware_version** (self)
- def **get_com_module** (self)
- def **read_channel_data_counts** (self, channel)
- def **write_channel_data_counts** (self, channel, data)
- def **read_data_counts** (self, start_channel=0, number_of_channels=1)
- def **write_data_counts** (self, start_channel, data_set)
- def **read_channel_data** (self, channel)
- def **write_channel_data** (self, channel, data)
- def **read_data** (self, start_channel=0, number_of_channels=1)
- def **write_data** (self, start_channel, data_set)
- def **__str__** (self)

  Magic Methods Override.
- def **__repr__** (self)
- def **__iter__** (self)
- def **__next__** (self)
- def **__len__** (self)
- def **__getitem__** (self, item)
- def **__setitem__** (self, key, value)

Static Public Attributes

- **no_ranges_error** = AttributeError('Converting to Engineering units not available for this module')

Detailed Description

```
Every Module, including the COM inherits from MAQ20Object.
This contains every functionality that is shared between all modules.
```

Constructor & Destructor Documentation

def maq20.maq20module.MAQ20Module.__init__ (*self*, *com* = `None`, *registration_number* = `0`)

```
Reads general information about the MAQ20 module to registers.
Registers: [0, 100]
:param com: a reference to the COM object in the system.
:param registration_number: this is used to calculate address map automatically.
```

Member Function Documentation

def maq20.maq20module.MAQ20Module.__getitem__ (*self*, *item*)

def maq20.maq20module.MAQ20Module.__iter__ (*self*)

def maq20.maq20module.MAQ20Module.__len__ (*self*)

def maq20.maq20module.MAQ20Module.__next__ (*self*)

def maq20.maq20module.MAQ20Module.__repr__ (*self*)

def maq20.maq20module.MAQ20Module.__setitem__ (*self*, *key*, *value*)

def maq20.maq20module.MAQ20Module.__str__ (*self*)

Magic Methods Override.

def maq20.maq20module.MAQ20Module.counts_to_eng_units_list (*self*, *data_list*, *channel*)

def maq20.maq20module.MAQ20Module.display_ranges_information (*self*)

```
Construct a string in human readable form of range information for all ranges in the
module.
:return: str
```

def maq20.maq20module.MAQ20Module.get_channel_active_range (*self*, *channel* = 0)

```
:param channel: channel requested.
:return: integer of the active channel range.
```

def maq20.maq20module.MAQ20Module.get_channel_ranges_information (*self*, *channel*)

def maq20.maq20module.MAQ20Module.get_com_module (*self*)

def maq20.maq20module.MAQ20Module.get_counts_full_scale_negative (*self*, *channel*)

def maq20.maq20module.MAQ20Module.get_counts_full_scale_positive (*self*, *channel*)

def maq20.maq20module.MAQ20Module.get_date_code (*self*, *str*)

def maq20.maq20module.MAQ20Module.get_engineering_full_scale_negative (*self*, *channel*)

def maq20.maq20module.MAQ20Module.get_engineering_full_scale_positive (*self*, *channel*)

def maq20.maq20module.MAQ20Module.get_engineering_units (*self*, *channel*)

def maq20.maq20module.MAQ20Module.get_firmware_version (*self*, *str*)

def maq20.maq20module.MAQ20Module.get_inputs (*self*, *int*)

def maq20.maq20module.MAQ20Module.get_name (*self*, *str*)

def maq20.maq20module.MAQ20Module.get_number_of_channels (*self*, *int*)

def maq20.maq20module.MAQ20Module.get_outputs (*self*, *int*)

def maq20.maq20module.MAQ20Module.get_ranges_information (*self*)

```
returns the current ranges information.
:return: list(dict())
```

def maq20.maq20module.MAQ20Module.get_registration_number (*self*)

def maq20.maq20module.MAQ20Module.get_serial_number (*self*, *str*)

def maq20.maq20module.MAQ20Module.has_range_information (*self*)

Ranges Information [1700,1899].

```
Checks if this module has range information stored.
```

```
Range information is used to convert from counts to engineering units.
:return: Boolean
```

def maq20.maq20module.MAQ20Module.load_channel_active_ranges (*self*)

```
All channel's active range is stored in RAM.
:return: the list saved.
```

def maq20.maq20module.MAQ20Module.load_ranges_information (*self*)

```
Loads a module's ranges information into RAM.
Ranges are stored in a list of dictionaries that has the keys:
"Engineering-FS"
"Engineering+FS"
"EngineeringUnits"
"CountValue-FS"
"CountValue+FS"
```

def maq20.maq20module.MAQ20Module.module_information (*self*, *str*)

```
:return: a str containing information about this module. (Registers [0, 100] and
registration number).
```

def maq20.maq20module.MAQ20Module.read_channel_data (*self*, *channel*)

```
Reads channel data from the module.
:param channel: channel to read.
:return: float
```

def maq20.maq20module.MAQ20Module.read_channel_data_counts (*self*, *channel*)

```
Reads data from a channel of this module in raw counts.
:param channel: channel number to read.
:return: int
```

def maq20.maq20module.MAQ20Module.read_data (*self*, *start_channel* = 0, *number_of_channels* = 1)

```
Reads channel data from the module.
:param start_channel: channel to start reading from.
:param number_of_channels: how many channels you want to read,
self.get number of channels() can be used.
:return: list(float)
```

def maq20.maq20module.MAQ20Module.read_data_counts (*self*, *start_channel* = 0, *number_of_channels* = 1)

```
Reads data for the requested channel in COUNTS.
:param start_channel: channel to start reading from.
:param number_of_channels: number of channels to be read.
```

```
:return: list(int)
```

def maq20.maq20module.MAQ20Module.read_date_code (*self*, *str*)

def maq20.maq20module.MAQ20Module.read_firmware_revision (*self*, *str*)

def maq20.maq20module.MAQ20Module.read_input_channels (*self*, *int*)

def maq20.maq20module.MAQ20Module.read_name (*self*, *str*)

def maq20.maq20module.MAQ20Module.read_output_channels (*self*, *int*)

def maq20.maq20module.MAQ20Module.read_range (*self*, *a_range* = 0)

def maq20.maq20module.MAQ20Module.read_range_count (*self*)

def maq20.maq20module.MAQ20Module.read_register (*self*, *address*)

Modbus Communication.

```
Calls the MAQ20-COMx module's read_register().
Performs a modbus read register request to the MAQ20
:param address: requested address
:return: int - [-32767, 32767]
```

def maq20.maq20module.MAQ20Module.read_registers (*self*, *address*, *number_of_registers*)

```
Calls the MAQ20-COMx module's read registers().
Performs a modbus read registers request to the MAQ20
:param address: starting address.
:param number_of_registers: number of registers to be read in sequence.
:return: list(int) [-32767, 32767]
```

def maq20.maq20module.MAQ20Module.read_serial_number (*self*, *str*)

def maq20.maq20module.MAQ20Module.write_channel_data (*self*, *channel*, *data*)

```
Writes data to channel.
:param channel: int
:param data: float
:return: modbus_response
```

def maq20.maq20module.MAQ20Module.write_channel_data_counts (*self*, *channel*, *data*)

def maq20.maq20module.MAQ20Module.write_data (*self*, *start_channel*, *data_set*)

```
Writes data_set to the module starting at channel start_channel, data_set has to be
iterable.
:param start channel:
:param data set:
:return:
```

def maq20.maq20module.MAQ20Module.write_data_counts (*self*, *start_channel*, *data_set*)

def maq20.maq20module.MAQ20Module.write_module_detect (*self*)

def maq20.maq20module.MAQ20Module.write_register (*self*, *address*, *value*)

```
Calls the MAQ20-COMx module's write register().
Performs a modbus write register request to the MAQ20
:param address: starting address.
:param value: int [-32767, 32767] or a str of size 1
:return: modbus response.
```

def maq20.maq20module.MAQ20Module.write_registers (*self*, *address*, *values* = `None`)

```
Calls the MAQ20-COMx module's write_registers().
Performs a modbus write registers request to the MAQ20
:param address: starting address.
:param values: list(int) [-32767, 32767] or a str
:return: modbus response.
```

def maq20.maq20module.MAQ20Module.write_reset_register (*self*, *input_value*)

Member Data Documentation

maq20.maq20module.MAQ20Module.no_ranges_error = AttributeError('Converting to Engineering units not available for this module') `[static]`

The documentation for this class was generated from the following file:

- maq20/**maq20module.py**

maq20.maq20.MAQ20.StdoutCatcher Class Reference

Public Member Functions

- def **__init__** (self)
- def **write** (self, txt)
- def **get_text** (self)
- def **__del__** (self)

Detailed Description

```
This inner class is used to catch stdout output because the standard FTP libraries write
to stdout.
```

Constructor & Destructor Documentation

def maq20.maq20.MAQ20.StdoutCatcher.__init__ (***self*** **)**

def maq20.maq20.MAQ20.StdoutCatcher.__del__ (***self*** **)**

Member Function Documentation

def maq20.maq20.MAQ20.StdoutCatcher.get_text (***self*** **)**

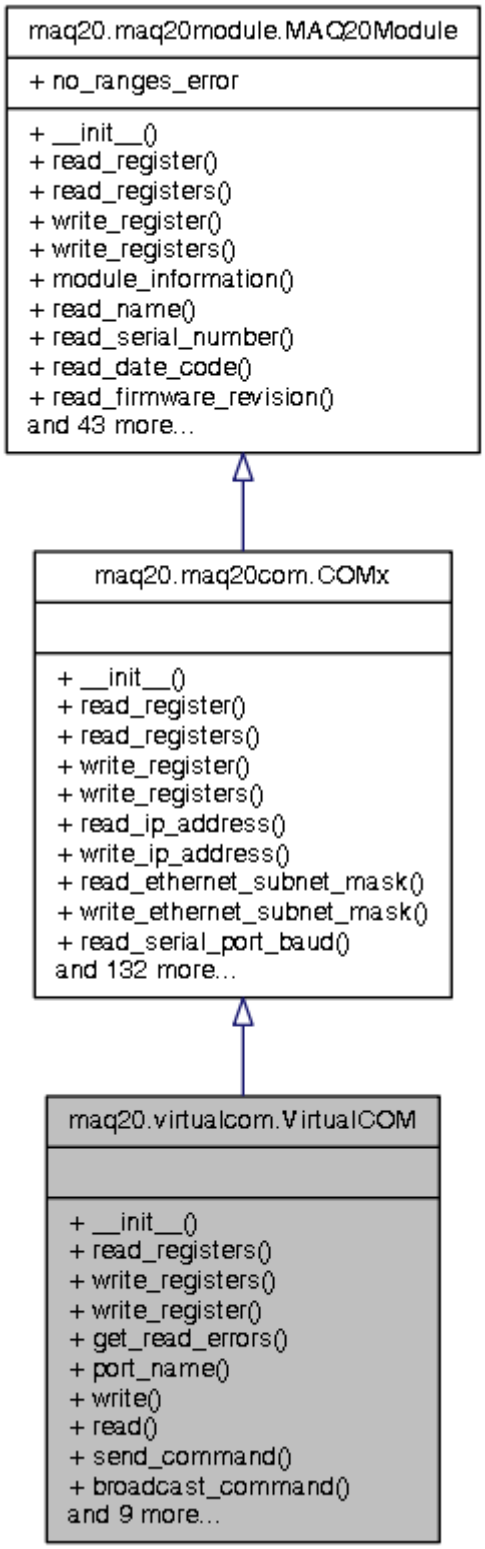
def maq20.maq20.MAQ20.StdoutCatcher.write (***self*** **,** ***txt*** **)**

The documentation for this class was generated from the following file:

- maq20/**maq20.py**

maq20.virtualcom.VirtualCOM Class Reference

Inheritance diagram for maq20.virtualcom.VirtualCOM:



Public Member Functions

- def **__init__** (self, port=None)
- def **read_registers** (self, address, number_of_registers)

- def **write_registers** (self, address, values=None)
- def **write_register** (self, address, value)
- def **get_read_errors** (self)
- def **port_name** (self)
- def **write** (self, data)
- def **read** (self, size)
- def **send_command** (self, message)
- def **broadcast_command** (self)
- def **flush_serial_buffer** (self)
- def **read_name** (self)
  START OVERRIDING.
- def **__str__** (self)
- def **module_information** (self)
- def **read_serial_number** (self)
- def **read_date_code** (self)
- def **read_module_status** (self)
- def **read_firmware_revision** (self)
- def **__del__** (self)

Additional Inherited Members

Detailed Description

This is a virtual MAQ20 COMx module that talks directly to the modules in the MAQ20 System.

Constructor & Destructor Documentation

def maq20.virtualcom.VirtualCOM.__init__ (***self*****,** ***port*** **= `None`)**

def maq20.virtualcom.VirtualCOM.__del__ (***self*****)**

Member Function Documentation

def maq20.virtualcom.VirtualCOM.__str__ (*self*, *str*)

def maq20.virtualcom.VirtualCOM.broadcast_command (*self*)

def maq20.virtualcom.VirtualCOM.flush_serial_buffer (*self*)

def maq20.virtualcom.VirtualCOM.get_read_errors (*self*)

def maq20.virtualcom.VirtualCOM.module_information (*self*, *str*)

def maq20.virtualcom.VirtualCOM.port_name (*self*)

def maq20.virtualcom.VirtualCOM.read (*self*, *size*)

def maq20.virtualcom.VirtualCOM.read_date_code (*self*, *str*)

def maq20.virtualcom.VirtualCOM.read_firmware_revision (*self*, *str*)

def maq20.virtualcom.VirtualCOM.read_module_status (*self*)

def maq20.virtualcom.VirtualCOM.read_name (*self*, *str*)

START OVERRIDING.

def maq20.virtualcom.VirtualCOM.read_registers (*self*, *address*, *number_of_registers*)

def maq20.virtualcom.VirtualCOM.read_serial_number (*self*, *str*)

def maq20.virtualcom.VirtualCOM.send_command (*self*, *message*)

def maq20.virtualcom.VirtualCOM.write (*self*, *data*)

def maq20.virtualcom.VirtualCOM.write_register (*self*, *address*, *value*)

def maq20.virtualcom.VirtualCOM.write_registers (*self*, *address*, *values* = `None`)

The documentation for this class was generated from the following file:

- maq20/virtualcom.py