QontrolDoc Documentation

Release 0

Qontrol

May 23, 2020

CONTENTS:

Welcome to Qontrol's documentation!

This documentation describes teh functionality of the main commands for the python **Qontrol** code. See 'https://qontrol.co.uk/'

---

qontrol.py is a library for configuring and controlling through python the qontrol products.

This guide will guide you through the process of setting up your Qontrol Q8 device and give you a description of the basic commands.

CHAPTER

ONE

GETTING STARTED GUIDE

This Getting started document will guide you through the first configuration and test of the q8 unit attached to a Qontrol motherboard.

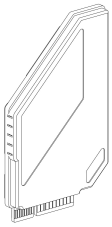
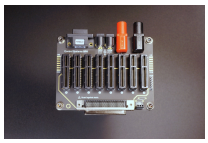
This guide requires :

- a computer connected to the internet to let the system download the serial port USB drivers
- (optionally) a program for serial port communication (e.g. Teraterm for windows, CoolTerm for Mac Osx, Linux)

1.1 Connection of the unit

This procedure is valid for both the Q8 and the Q8iv products.

- Insert the Q8iv (or any other compatible control unit) in one of the backplanes/motherboards (e.g. BP8):

- Connect the backplane (motherboard) unit to the computer using a cable/adapter with a USB mini b female plug at one of the two ends, like the one shown below:

- Power the unit, using a compatible power supply (e.g. PS15KIT):

- Al the side LEDs in the units should progressively turn on and off again leaving only the bottom green LEDs on, while the units are in idle.

1.2 Configuration of the serial communication

1.2.1 Controlling the unit using a serial communication software

Serial communication software in any operating system (OS) can be used to control the units, some examples:

- Teraterm (Windows)
- CoolTerm (Mac)
- Terminal/Command line (Linux)

General Configuration settings.

Serial parameters:

- 8 bits for Data
- 1 bit for stop
- no parity check
- no flow control
- Baud Rate 115200

Teraterm Configuration

Mac CoolTerm Configuration To check the name of the device

ls /dev/tty.usb*

Linux Command Line

http://my.fit.edu/~msilaghi/ROB/iCreate/serial.pdf

To check the name of the device

ls /dev/tty.usb*

1.3 First operations and tests

1.4 First Troubleshouting

1.5 Notes and disclaimer

If you find an error in this document, or have suggestions for how we could make it better, please do get in touch with us at support@qontrol.co.uk with your comments.

The information provided in this document is believed to be accurate at the time of publication. It is provided for information only, 'as is', and without guarantee of any kind.

Qontrol Systems LLP, its subsidiaries and associates accept no liability for damage to equipment, hardware, or the customer application, or for labour costs incurred due to the information contained in this document.

CHAPTER

TWO

QONTROL_API

2.1 qontrol module

class qontrol.**ChannelVector**(*base_list*)
Bases: object

Custom list class which has a fixed length but mutable (typed) elements, and which phones home when its elements are read or modified.

get_handle = None

set_handle = None

valid_types = (<class 'int'>, <class 'float'>)

class qontrol.**QXOutput**(**args*, ***kwargs*)
Bases: *qontrol.Qontroller*

get_all_values(*para='V'*)

get_value(*ch*, *para='V'*)

set_all_values(*para='V'*, *values=0*)
Convenience function for slicing up set commands into vectors for each module and transmitting.

para: Parameter to set {'V' or 'I'} values: Either float/int or list of float/int of length n_chs

set_value(*ch*, *para='V'*, *new=0*)

class qontrol.**Qontroller**(**args*, ***kwargs*)
Bases: object

Super class which handles serial communication, device identification, and logging.

device_id = None Device ID serial_port = None Serial port object serial_port_name = None Name of serial port, eg 'COM1' or '/dev/tty1' error_desc_dict = Q8x_ERRORS Error code descriptions log = fifo(maxlen = 256) Log FIFO of sent commands and received errors log_handler = None Function which catches log dictionaries log_to_stdout = True Copy new log entries to stdout response_timeout = 0.050 Timeout for response or error to commands inter_response_timeout = 0.020 Timeout for response or error to get commands

Log handler: The log handler may be used to catch and dynamically handle certain errors, as they arise. In the following example, it is set up to raise a RuntimeError upon reception of errors E01, E02, and E03:

q = Qontroller()

fatal_errors = [1, 2, 3]

def my_log_handler(err_dict):

if err_dict['type'] is 'err' and err_dict['id'] in fatal_errors: raise RuntimeError('Caught Qontrol error "{1}" at {0} ms'.format(1000*err_dict['proctime'], err_dict['desc']))

q.log_handler = my_log_handler

close()
Release resources

issue_binary_command(*command_id*, *ch=None*, *BCAST=0*, *ALLCH=0*, *ADDM=0*, *RW=0*, *ACT=0*, *DEXT=0*, *value_int=0*, *addr_id_num=0*, *n_lines_requested=2147483648*, *target_errors=None*, *output_regex='(.*)'*, *special_timeout=None*)
Transmit command ([command_id][ch][operator][value]) to device, collect response.

command_id: Command descriptor, either int (command index) or str (command name). ch: Channel address (0x0000..0xFFFF for ADDM=0, 0x00..0xFF for ADDM=1). BCAST,

ALLCH, ADDM, RW, ACT, DEXT: Header byte bits. See Programming Manual for full description.

value_int: Data, either int (DEXT=0) or list of int (DEXT=1). addr_id_num: For device-wise addressing mode (ADDM=1) only, hex device ID code.

All other arguments same as those for issue_command()

issue_command(*command_id*, *ch=None*, *operator=''*, *value=None*, *n_lines_requested=2147483648*, *target_errors=None*, *output_regex='(.*)'*, *special_timeout=None*)
Transmit command ([command_id][ch][operator][value]) to device, collect response.

command_id Command header (e.g. 'v' in 'v7=1.0') ch Channel index to apply command to (e.g. '7' in 'v7=1.0') operator Type of command in {?, =} (e.g. '=' in 'v7=1.0') value Value of set command (e.g. '1.0' in 'v7=1.0') n_lines_requested Lines of data (not error) to stop after receiving, or timeout target_errors Error numbers which will be raised as RuntimeError special_timeout Timeout to use for this command only (!= self.response_timeout)

log_append(*type='err'*, *id=''*, *ch=0*, *value=0*, *desc=''*, *raw=''*)
Append an event to the log, adding both a calendar- and a process-timestamp."

parse_error(*error_str*)
Parse an encoded error (e.g. E02:07) into its code, channel, and human-readable description.

print_log(*n=None*)
Print the n last log entries. If n == None, print all log entries.

receive()
Low-level receive data method which also checks for errors.

transmit(*command_string*, *binary_mode=False*)
Low-level transmit data method. command_string can be of type str or bytearray

wait(*seconds=0.0*)
Do nothing while watching for errors on the serial bus.

qontrol.**run_interactive_shell**()
Interactive shell for interacting directly with Qontrol hardware.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

q

INDEX