Moler Documentation

Release stable

Nokia

Sep 15, 2023

Contents:

Moler key features

- Event observers & callbacks (alarms are events example)
  - to allow for online reaction (not offline postprocessing)
- Commands as self-reliant object
  - to allow for command triggering and parsing encapsulated in single object (lower maintenance cost)
- Run observers/commands in the background
  - to allow for test logic decomposition into multiple commands running in parallel
  - to allow for handling unexpected system behavior (reboots, alarms)
- State machines -> automatic auto-connecting after dropped connection
  - to increase framework auto-recovery and help in troubleshooting “what went wrong”
- Automatic logging of all connections towards devices used by tests
  - to decrease investigation time by having logs focused on different parts of system under test

CHAPTER 1

Installation

Moler is a pure-python project. But since there is nor official release yet, you can just clone it:

```
$ git clone https://github.com/nokia/moler.git
```

Later it will added to PyPI and be possible to use pip as all other Python packages. This means installing will be extremely simple:

```
$ pip install moler
```

Always you are welcome to our GitHub page!

CHAPTER 2

moler

2.1 moler package

2.1.1 Subpackages

moler.cmd package

Subpackages

moler.cmd.adb package

Submodules

moler.cmd.adb.adb_shell module

adb shell command module.

class moler.cmd.adb.adb_shell.**AdbShell**(*connection*, *serial_number=None*, *prompt=None*, *expected_prompt=None*, *newline_chars=None*, *target_newline='n'*, *runner=None*, *set_timeout=None*, *allowed_newline_after_prompt=False*, *set_prompt=None*, *prompt_after_login=None*, *prompt_from_serial_number=False*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

re_generated_prompt = '^adb_shell@{} \\$'

Module contents

Package for implementing ADB commands.

moler.cmd.at package

Submodules

moler.cmd.at.at module

AT .

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.at.**At**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to check if AT console is operable. Example output:

AT OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

moler.cmd.at.attach module

AT+CGATT=1 . Attach

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.attach.**Attach**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to trigger attach. Example output:

AT+CGATT=1 OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

moler.cmd.at.cu module

Run cu -l /dev/ttyS{} -s 19200 command.

class moler.cmd.at.cu.**Cu**(*connection*, *serial_devname*, *prompt=None*, *newline_chars=None*, *target_newline='n'*, *runner=None*, *options=None*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

Command to connect COM port using cu. Example output:

$ cu -l /dev/ttyS21 -s 19200 -E - Connected.

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.at.detach module

AT+CGATT=0 . Detach

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.detach.**Detach**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to trigger detach. Example output:

AT+CGATT=0 OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

moler.cmd.at.enable_echo module

ATE1

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.enable_echo.**EnableEcho**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to enable echo. Example output:

ATE1 OK

build_command_string()
Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

moler.cmd.at.exit_serial_proxy module

Exit AtConsole<->stdio proxy

class moler.cmd.at.exit_serial_proxy.**ExitSerialProxy**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

build_command_string()
command string to exit from moler_serial_proxy

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.at.genericat module

Common part for all commands.

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.genericat.**GenericAtCommand**(*connection*, *operation='execute'*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_apns module

AT+CGDCONT

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_apns.**GetApns**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get APNs. Example output:

+CGDCONT: 1,”IPV4V6”,”apnscp1”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0 +CGDCONT: 2,”IPV4V6”,””,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0 +CGDCONT: 3,”IPV4V6”,”ims”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0 +CGDCONT: 4,”IPV4V6”,”sos”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,1 +CGDCONT: 5,”IPV4V6”,”xcap”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0

OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo.

+CGDCONT: 1,”IPV4V6”,”apnscp1”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0 +CGDCONT: 2,”IPV4V6”,””,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0 +CGDCONT: 3,”IPV4V6”,”ims”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0 +CGDCONT: 4,”IPV4V6”,”sos”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,1 +CGDCONT: 5,”IPV4V6”,”xcap”,”0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0”,0,0,0,0

OK

Write your own implementation but don’t forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_attach_state module

AT+CGATT? . Check attach state: attached/detached

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_attach_state.**GetAttachState**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to check attach state. Example output:

AT+CGATT? +CGATT: 1 OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo.

+CGATT: 1 OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_cell_id module

AT+CREG?

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_cell_id.**GetCellId**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Example output:

+CREG: <n>,<stat>[,[<lac>],[<ci>],[<AcT>][,<cause_type>,<reject_cause>]]

OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_cell_id_gprs module

AT+CGREG?

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_cell_id_gprs.**GetCellIdGprs**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Example outputs:

+CGREG: <n>,<stat>[,[<lac>],[<ci>],[<AcT>],[<rac>][,<cause_type>,<reject_cause>]]

OK or +CGREG: <n>,<stat>[,[<lac>],[<ci>],[<AcT>],[<rac>][,[<cause_type>],[<reject_cause>]...
...[,[<Active-Time>],[<Periodic-RAU>],[<GPRS-READY-timer>]]]]

OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_cell_id_lte module

AT+CEREG?

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_cell_id_lte.**GetCellIdLte**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.at.genericat.GenericAtCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo.

+CEREG: <n>,<stat>[,[<lac>],[<ci>],[<AcT>][,<cause_type>,<reject_cause>]]

OK or +CEREG: <n>,<stat>[,[<lac>],[<ci>],[<AcT>][,[<cause_type>],[<reject_cause>]...
...[,[<Active-Time>],[<Periodic-RAU>],[<GPRS-READY-timer>]]]]

OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_cell_id_nr module

AT+C5GREG?

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_cell_id_nr.**GetCellIdNr**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo.

+C5GREG: <n>,<stat>[,[<tac>],[<ci>],[<AcT>],[<Allowed_NSSAI_length>],[<Allowed_NSSAI>]...
... [,<cause_type>,<reject_cause>]][,<cag_stat>][,<caginfo>]

OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_imei module

AT+CGSN .

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_imei.**GetImei**(*connection=None*, *sn_type='default'*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get product serial number identification. Example output:

AT+CGSN 490154203237518

AT+CGSN=1 +CGSN: "490154203237518" OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

is_end_of_cmd_output(*line*)

Checks if end of command is reached.

AT+CGSN and AT+CGSN=0 are not finished by OK, so such cmd is finished when it detects serial_number

Parameters **line** – Line from device.

Returns

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo.

490154203237518

or

+CGSN: “490154203237518” OK

Write your own implementation but don’t forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_imsi module

AT+CIMI .

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_imsi.**GetImsi**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get IMSI. Example output:

AT+CIMI 49009123123123 OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo.

49009123123123 OK

Write your own implementation but don’t forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_ip module

AT+CGPADDR

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_ip.**GetIp**(*context_identifier*, *connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get IP. Example output:

+CGPADDR: 1,0.0.0.0,0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0

OK

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo.

+CGPADDR: 1,"40.1.1.105"

OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_manufacturer_id module

AT+CGMI .

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_manufacturer_id.**GetManufacturerId**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get manufacturer identification. Example output:

AT+CGMI QUALCOMM INCORPORATED

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

is_end_of_cmd_output(*line*)
Checks if end of command is reached.

AT+CGMI is not finished by OK, so it is finished when it detects manufacturer

Parameters **line** – Line from device.

Returns

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo.

QUALCOMM INCORPORATED

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_product_info module

ATI

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_product_info.**GetProductInfo**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get product information. Example output:

Manufacturer: QUALCOMM INCORPORATED Model: 334 Revision: OEM_VER: RTL6300_NOKIA_V0.0.3_201116.1_m OEM_BLD: master@dailybuild2, 11/16/2020 05:56:34 QC_VER: MPSS.HI.2.0.c3-00246-SDX55_CPEALL_PACK-1 IMEI: 352569090027192 +GCAP: +CGSM

OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo.

Manufacturer: QUALCOMM INCORPORATED Model: 334 Revision: OEM_VER: RTL6300_NOKIA_V0.0.3_201116.1_m OEM_BLD: master@dailybuild2, 11/16/2020 05:56:34 QC_VER: MPSS.HI.2.0.c3-00246-SDX55_CPEALL_PACK-1 IMEI: 352569090027192 +GCAP: +CGSM

OK

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.get_revision_id module

AT+CGMR .

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.get_revision_id.**GetRevisionId**(*connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to get revision identification. Example output:

AT+CGMR MPSS.HE.1.5.2-00368-SM8150_GENFUSION_PACK-1 1 [Aug 07 2019 21:00:00]

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

is_end_of_cmd_output(*line*)
Checks if end of command is reached.

AT+CGMR is not finished by OK, so it is finished when it detects revision

Parameters **line** – Line from device.

Returns

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo.

MPSS.HE.1.5.2-00368-SM8150_GENFUSION_PACK-1 1 [Aug 07 2019 21:00:00]

Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.at.plink_serial module

Run plink -serial command

class moler.cmd.at.plink_serial.**PlinkSerial**(*connection*, *serial_devname*, *prompt=None*, *newline_chars=None*, *target_newline='n'*, *runner=None*)
Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

build_command_string()
Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.at.set_apn module

AT+CGDCONT

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.set_apn.**SetApn**(*apn_name*, *context_identifier='1'*, *pdp_type='IPV4V6'*, *connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to set apn.

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

moler.cmd.at.set_mode module

AT+COPS

AT commands specification: google for: 3gpp specification 27.007 (always check against latest version of standard)

class moler.cmd.at.set_mode.**SetMode**(*selected_mode*, *connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to set mode (automatic, lte, gsm, wcdma).

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

mode2cops_value = {'automatic': '0', 'gsm': '0,0,0,3', 'lte': '0,0,0,7', 'wcdma':

moler.cmd.at.quectel_lock_nrearfcn module

AT+QNWCFG="nr5g_earfcn_lock",(0..32),earfcn1:scs1:. . . :earfcnN:scsN

AT commands specification: google for: Quectel RG50xQRM5xxQ (R11 release and later) This is internal Quectel AT command (always check against the latest vendor release notes)

class moler.cmd.at.quectel_lock_nr_earfcn.**QuectelLockNrEarfcn**(*earfcn*, *scs*, *connection=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.at.genericat.GenericAtCommand*

Command to lock NR EARFCN. Example output:

AT+QNWCFG="nr5g_earfcn_lock",1,433970:15 OK

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

Module contents

Package for implementing AT commands.

AT commands specification: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1515 (always check against latest version of standard)

moler.cmd.juniper package

Subpackages

moler.cmd.juniper.cli package

Submodules

moler.cmd.juniper.cli.configure module

Configure command module.

class moler.cmd.juniper.cli.configure.**Configure**(*connection*, *prompt=None*, *expected_prompt='^admin@switch#'*, *newline_chars=None*, *runner=None*, *target_newline='rn'*)

Bases: *moler.cmd.juniper.genericjuniper.GenericJuniperCommand*

Configure command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

Module contents

Package for implementing Juniper/CLI commands.

moler.cmd.juniper.configure package

Submodules

moler.cmd.juniper.configure.exit_configure module

Exitconfigure command module.

class moler.cmd.juniper.configure.exit_configure.**ExitConfigure**(*connection*, *prompt=None*, *expected_prompt='^admin@switch>'*, *newline_chars=None*, *runner=None*, *target_newline='rn'*)

Bases: *moler.cmd.juniper.genericjuniper.GenericJuniperCommand*

Configure command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

Module contents

Package for implementing Juniper/CONFIGURE commands.

Submodules

moler.cmd.juniper.genericjuniper module

Generic juniper module.

class moler.cmd.juniper.genericjuniper.**GenericJuniperCommand**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

Genericjunipercommand command class.

Module contents

Package for implementing Juniper commands.

moler.cmd.juniper_ex package

Subpackages

moler.cmd.juniper_ex.cli package

Module contents

Package for implementing JuniperEX/CLI commands.

moler.cmd.juniper_ex.configure package

Module contents

Package for implementing JuniperEX/CONFIGURE commands.

Submodules

moler.cmd.juniper_ex.genericjuniperex module

Generic juniperex module.

class moler.cmd.juniper_ex.genericjuniperex.**GenericJuniperEXCommand**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

Genericjuniperexcommand command class.

Module contents

Package for implementing JuniperEX commands.

moler.cmd.pdu_aten package

Subpackages

moler.cmd.pdu_aten.pdu package

Submodules

moler.cmd.pdu_aten.pdu.exit_telnet module

Exit telnet command

class moler.cmd.pdu_aten.pdu.exit_telnet.**ExitTelnet**(*connection*, *newline_chars=None*, *prompt=None*, *runner=None*, *expected_prompt='moler_bash#'*, *set_timeout=None*, *set_prompt=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *prompt_after_login=None*)

Bases: *moler.cmd.unix.exit_telnet.ExitTelnet*

moler.cmd.pdu_aten.pdu.generic_pdu module

PDU generic module for commands in state PDU.

class moler.cmd.pdu_aten.pdu.generic_pdu.**GenericPdu**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.pdu_aten.generic_pdu_aten.GenericPduAten*

is_failure_indication(*line*)

Method to detect if passed line contains part indicating failure of command

Parameters **line** – Line from command output on device

Returns Match object if find regex in line, None otherwise.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.pdu_aten.pdu.quit module

Quit command module.

class moler.cmd.pdu_aten.pdu.quit.**Quit**(*connection*, *prompt=None*, *expected_prompt='moler_bash#'*, *newline_chars=None*, *runner=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

build_command_string()

Returns a string with command.

Returns String with the command.

moler.cmd.pdu_aten.pdu.read_meter module

Command read status.

class moler.cmd.pdu_aten.pdu.read_meter.**ReadMeter**(*connection*, *target*, *measurement*, *outlet=None*, *output_format=None*, *prompt='>'*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.pdu_aten.pdu.generic_pdu.GenericPdu*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.pdu_aten.pdu.read_status module

Command read status.

class moler.cmd.pdu_aten.pdu.read_status.**ReadStatus**(*connection*, *outlet*, *output_format=None*, *prompt='>'*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.pdu_aten.pdu.generic_pdu.GenericPdu*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.pdu_aten.pdu.sw module

Command sw.

class moler.cmd.pdu_aten.pdu.sw.**Sw**(*connection*, *outlet*, *control*, *option=None*, *prompt='>'*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.pdu_aten.pdu.generic_pdu.GenericPdu*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

Module contents

Package for implementing PDU Aten commands in state PDU.

Submodules

moler.cmd.pdu_aten.generic_pdu_aten module

PDU generic module for commands in state PDU.

class moler.cmd.pdu_aten.generic_pdu_aten.**GenericPduAten**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

Module contents

Package for implementing Pdu Atel commands.

moler.cmd.scpi package

Subpackages

moler.cmd.scpi.scpi package

Submodules

moler.cmd.scpi.scpi.exit_telnet module

Exit telnet command

class moler.cmd.scpi.scpi.exit_telnet.**ExitTelnet**(*connection*, *newline_chars=None*, *prompt=None*, *runner=None*, *expected_prompt='moler_bash#'*, *set_timeout=None*, *set_prompt=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *prompt_after_login=None*)

Bases: *moler.cmd.unix.exit_telnet.ExitTelnet*

moler.cmd.scpi.scpi.genericscpistate module

SCPI generic module for commands in state SCPI.

class moler.cmd.scpi.scpi.genericscpistate.**GenericScpiState**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.scpi.genricscpi.GenericScpi*

moler.cmd.scpi.scpi.idn module

SCPI command idn module.

class moler.cmd.scpi.scpi.idn.**Idn**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.scpi.scpi.genericscpistate.GenericScpiState*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class in most cases.

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – True if new line character was removed from line, False otherwise

Returns None

moler.cmd.scpi.scpi.read module

SCPI command READ.

class moler.cmd.scpi.scpi.read.**Read**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.scpi.scpi.genericscpistate.GenericScpiState*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class in most cases.

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – True if new line character was removed from line, False otherwise

Returns None

moler.cmd.scpi.scpi.run_command module

SCPI command to support command without interesting output.

class moler.cmd.scpi.scpi.run_command.**RunCommand**(*connection*, *command*, *error_regex=re.compile('ERROR', re.IGNORECASE)*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.scpi.scpi.genericscpistate.GenericScpiState*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class in most cases.

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – True if new line character was removed from line, False otherwise

Returns None

Module contents

Package for implementing SCPI in state SCPI commands.

Submodules

moler.cmd.scpi.genricscpi module

SCPI generic module for commands in all states.

class moler.cmd.scpi.genricscpi.**GenericScpi**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

Module contents

Package for implementing SCPI commands.

moler.cmd.unix package

Submodules

moler.cmd.unix.bash module

Bash command module.

class moler.cmd.unix.bash.**Bash**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *bash='TERM=xterm-mono bash'*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns String representation of command to send over connection to device.

moler.cmd.unix.cat module

Cat command module.

class moler.cmd.unix.cat.**Cat**(*connection*, *path*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *failure_only_in_first_line=True*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds string with command.

Returns String with command.

is_failure_indication(*line*)

Method to detect if passed line contains part indicating failure of command.

Parameters **line** – Line from command output on device

Returns Match object if find regex in line, None otherwise.

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

set_exception(*exception*)

Set exception object as failure for command object.

Parameters **exception** – An exception object to set.

Returns None.

moler.cmd.unix.cd module

cd command module.

class moler.cmd.unix.cd.**Cd**(*connection*, *path=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *expected_prompt=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.chgrp module

Chgrp command module.

class moler.cmd.unix.chgrp.**Chgrp**(*connection*, *files*, *group*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.chmod module

chmod command module.

class moler.cmd.unix.chmod.**Chmod**(*connection*, *permission*, *filename*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.chown module

chown command module.

class moler.cmd.unix.chown.**Chown**(*connection*, *param*, *filename*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.cp module

Cp command module.

class moler.cmd.unix.cp.**Cp**(*connection*, *src*, *dst*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ctrl_c module

Ctrl+C command module.

class moler.cmd.unix.ctrl_c.**CtrlC**(*connection*, *prompt=None*, *expected_prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix ctrl+c command

build_command_string()
Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

command_string
Getter for command_string.

Returns String with command_string

on_new_line(*line*, *is_full_line*)
Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.cut module

cut command module.

class moler.cmd.unix.cut.**Cut**(*connection*, *options=None*, *path=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix command cut.

build_command_string()
Build command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.date module

Date command module.

class moler.cmd.unix.date.**Date**(*connection*, *options=None*, *date_table_output=True*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.devmem module

Devmem command module.

class moler.cmd.unix.devmem.**Devmem**(*connection*, *address*, *size=None*, *value=None*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Devmem command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.df module

Df command module.

class moler.cmd.unix.df.**Df**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.dmesg module

Dmesg command module.

class moler.cmd.unix.dmesg.**Dmesg**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.du module

du command module.

class moler.cmd.unix.du.**Du**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix command du.

build_command_string()
Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.echo module

Echo command module.

class moler.cmd.unix.echo.**Echo**(*connection*, *options=None*, *text=None*, *write_mode='>'*, *output_file=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *text_in_quotation=True*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.enter module

enter command module.

class moler.cmd.unix.enter.**Enter**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns String representation of command to send over connection to device.

moler.cmd.unix.env module

Env command module.

class moler.cmd.unix.env.**Env**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ethtool module

Ethtool command module.

class moler.cmd.unix.ethtool.**Ethtool**(*connection*, *interface*, *prompt=None*, *newline_chars=None*, *options=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.exit module

Exit command module.

class moler.cmd.unix.exit.**Exit**(*connection*, *prompt=None*, *expected_prompt='>'*, *newline_chars=None*, *runner=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *start_command_immediately=True*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

build_command_string()

Returns a string with command.

Returns String with the command.

moler.cmd.unix.exit_telnet module

Exit telnet command

class moler.cmd.unix.exit_telnet.**ExitTelnet**(*connection*, *newline_chars=None*, *prompt=None*, *runner=None*, *expected_prompt='moler_bash#'*, *set_timeout=None*, *set_prompt=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *prompt_after_login=None*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.export module

Export command module.

class moler.cmd.unix.export.**Export**(*connection*, *prompt=None*, *newline_chars=None*, *ps1_param=None*, *set_param=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed

- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.find module

Find command module.

class moler.cmd.unix.find.**Find**(*connection*, *paths=None*, *prompt=None*, *newline_chars=None*, *options=None*, *operators=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.genericctelnetssh module

Base class for telnet and ssh commands.

class moler.cmd.unix.genericctelnetssh.**GenericTelnetSsh**(*connection*, *host*, *login=None*, *password=None*, *newline_chars=None*, *prompt=None*, *runner=None*, *port=0*, *expected_prompt='^>\s*'*, *set_timeout='export TMOUT=\"2678400\"'*, *set_prompt=None*, *term_mono='TERM=xterm-mono'*, *encrypt_password=True*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *repeat_password=True*, *failure_exceptions_indication=None*, *prompt_after_login=None*, *send_enter_after_connection=True*, *username=None*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

is_failure_indication(*line*)

Checks if line contains information that command fails.

Parameters **line** – Line from device

Returns Match object or None

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

Raises ParsingDone if any line matched the regex.

moler.cmd.unix.genericunix module

Generic Unix/Linux module

class moler.cmd.unix.genericunix.**GenericUnixCommand**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

is_failure_indication(*line*)

Method to detect if passed line contains part indicating failure of command

Parameters **line** – Line from command output on device

Returns Match object if find regex in line, None otherwise.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.grep module

Grep command module.

class moler.cmd.unix.grep.**Grep**(*connection*, *options*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.gunzip module

Gunzip command module.

class moler.cmd.unix.gunzip.**Gunzip**(*connection*, *archive_name*, *output_file_name=None*, *options=None*, *overwrite=False*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.gzip module

Gzip command module.

class moler.cmd.unix.gzip.**Gzip**(*connection*, *file_name*, *compressed_file_name=None*, *options=None*, *overwrite=False*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.hciconfig module

Hciconfig command module.

class moler.cmd.unix.hciconfig.**Hciconfig**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.head module

Head command module.

class moler.cmd.unix.head.**Head**(*connection*, *path*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Head command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of the command to send over a connection to the device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.hexdump module

Hexdump command module.

class moler.cmd.unix.hexdump.**Hexdump**(*connection*, *files*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.history module

History command module.

class moler.cmd.unix.history.**History**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix command history.

build_command_string()

Build command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.hostname module

Hostname command module.

class moler.cmd.unix.hostname.**Hostname**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.id module

id command module.

class moler.cmd.unix.id.**Id**(*connection*, *user=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.ifconfig module

ifconfig command module.

class moler.cmd.unix.ifconfig.**Ifconfig**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ip_addr module

ip addr command module.

class moler.cmd.unix.ip_addr.**IpAddr**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ip_link module

Ip link command module.

class moler.cmd.unix.ip_link.**IpLink**(*connection*, *action*, *prompt=None*, *newline_chars=None*, *options=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ip_neigh module

ip addr command module.

class moler.cmd.unix.ip_neigh.**IpNeigh**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ip_route module

ip route command module.

class moler.cmd.unix.ip_route.**IpRoute**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *is_ipv6=False*, *addr_get=None*, *addr_from=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix command ip route

build_command_string()
Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

get_default_route()
Helpful method to find default route from command output.

Returns Default route or None.

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.iperf module

Iperf command module.

+———————————————————————- | Deprecated module. Don't develop any more. Develop Iperf2 instead. +———————————————————————-

Deprecation rationale: This command returned value requires additional parsing and doesn't provide final report.

class moler.cmd.unix.iperf.**Iperf**(*connection*, *options*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Run iperf command and return its statistics

Statistics are given as list of dicts like:

```
{'Interval':                '0.0- 1.0 sec',
 'Transfer Raw':            '1.17 MBytes',
 'Transfer':                1226833,
 'Bandwidth Raw':           '9.84 Mbits/sec',
 'Bandwidth':               1230000,
 'Jitter':                  '1.830 ms',
 'Lost_vs_Total_Datagrams': '0/ 837   (0%)'}
```

Above dict represents iperf output like:

```
[ ID]   Interval         Transfer        Bandwidth         Jitter        Lost/
↪Total Datagrams
[904]   0.0- 1.0 sec   1.17 MBytes   9.84 Mbits/sec   1.830 ms   0/ 837   (0%)
```

Please note that numeric values are normalized to Bytes: - Transfer is in Bytes - Bandwith is in Bytes/sec

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.iperf2 module

Iperf2 command module.

It is refactored Iperf module changing data format returned. New format doesn't require additional post processing of values inside returned dict (it was the case with old one)

Moreover, new format provides final report - see bellow.

iperf2 was orphaned in the late 2000s at version 2.0.5 Then in 2014, Bob (Robert) McMahon from Broadcom restarted development of iperf2 Official iperf2 releases after 2.0.5: https://sourceforge.net/projects/iperf2/files/ Important changes: - starting from 2.0.8 -b may be used to limit bandwidth at TCP - as a consequence -b doesn't force -u

class moler.cmd.unix.iperf2.**Iperf2**(*connection*, *options*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*, *moler.publisher.Publisher*

Run iperf command, return its statistics and report.

Single line of iperf output may look like:

```
[ ID]   Interval       Transfer      Bandwidth        Jitter   Lost/Total␣
↪Datagrams
[904]   0.0- 1.0 sec   1.17 MBytes   9.84 Mbits/sec   1.830 ms    0/ 837   (0%)
```

It represents data transfer statistics reported per given interval. This line is parsed out and produces statistics record as python dict. (examples can be found at bottom of iperf2.py source code) Some keys inside dict are normalized to Bytes. In such case you will see both: raw and normalized values:

```
'Transfer Raw':     '1.17 MBytes',
'Transfer':         1226833,           # Bytes
'Bandwidth Raw':    '9.84 Mbits/sec',
'Bandwidth':        1230000,           # Bytes/sec
```

Iperf statistics are stored under connection name with format (client_port@client_IP, server_port@server_IP) It represents iperf output line (iperf server example below) like:

```
[  3] local 192.168.0.12 port 5016 connected with 192.168.0.10 port 56262
("56262@192.168.0.10", "5016@192.168.0.12"): [<statistics dicts here>]
```

Iperf returned value has also additional connection named "report connection". It has format (client_IP, server_port@server_IP) So, for above example you should expect structure like:

```
("192.168.0.10", "5016@192.168.0.12"): {'report': {<report dict here>}}
```

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

client

dualtest

interval

is_end_of_cmd_output(*line*)

Checks if end of command is reached.

For iperf server we can't await prompt since at server it is not displayed

Parameters **line** – Line from device.

Returns

on_inactivity()
Call when no data is received on connection within self.life_status.inactivity_timeout seconds.

Returns None

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

parallel_client

protocol

server

singlerun_server

subscribe(*subscriber*)
Subscribe for notifications about iperf statistic as it comes.

Anytime we find iperf statistics line like: [3] 2.0- 3.0 sec 612 KBytes 5010 Kbits/sec 0.022 ms 0/ 426 (0%) such line is parsed and published to subscriber

Subscriber must be function or method with following signature (name doesn't matter):

def iperf_observer(from_client, to_server, data_record=None, report=None): ...

Either data_record is published or report. Report is published on last line of iperf statistics summarizing stats for whole period: [904] 0.0-10.0 sec 11.8 MBytes 9.86 Mbits/sec 2.618 ms 9/ 8409 (0.11%)

Parameters **subscriber** – function to be called to notify about data.

time

works_in_dualtest

moler.cmd.unix.ipsec module

Ipsec command module.

class moler.cmd.unix.ipsec.**Ipsec**(*connection*, *options*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.iptables module

iptables command module.

class moler.cmd.unix.iptables.**Iptables**(*connection*, *options=None*, *v6=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.kill module

kill command module.

class moler.cmd.unix.kill.**Kill**(*connection*, *pid*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.killall module

Killall command module.

class moler.cmd.unix.killall.**Killall**(*connection*, *name*, *is_verbose=False*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ln module

Ln command module.

class moler.cmd.unix.ln.**Ln**(*connection*, *prompt=None*, *newline_chars=None*, *options=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.logout module

Logout command module.

class moler.cmd.unix.logout.**Logout**(*connection*, *prompt=None*, *expected_prompt='>'*, *newline_chars=None*, *runner=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *start_command_immediately=True*)

Bases: *moler.cmd.unix.exit.Exit*

build_command_string()
Returns a string with command.

Returns String with the command.

moler.cmd.unix.ls module

Ls command module.

class moler.cmd.unix.ls.**Ls**(*connection*, *prompt=None*, *newline_chars=None*, *options=None*, *path=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix command ls

build_command_string()
Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

get_dirs()
Returns only directories (folders) from command output

Returns Dict, key is item, value is parsed information about item

get_files()
Returns only files from command output

Returns Dict, key is item, value is parsed information about item

get_links()
Returns only links from command output

Returns Dict, key is item, value is parsed information about item

on_new_line(*line*, *is_full_line*)
Processes line from output form connection/device.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.lsof module

lsof command module.

class moler.cmd.unix.lsof.**Lsof**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix lsof command

build_command_string()
Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.lxc_attach module

LxcAttach command module.

moler.cmd.unix.lxc_attach.**COMMAND_RESULT_2 = {}**

==========HELP=MESSAGE==========
root@0xe000:~ >lxc-attach –help Usage: lxc-attach –name=NAME [– COMMAND]

Execute the specified COMMAND - enter the container NAME

Options :

-n, --name=NAME NAME of the container

-e, --elevated-privileges=PRIVILEGES Use elevated privileges instead of those of the container. If you don't specify privileges to be elevated as OR'd list: CAP, CGROUP and LSM (capabilities, cgroup and restrictions, respectively) then all of them will be elevated. WARNING: This may leak privileges into the container. Use with care.

-a, --arch=ARCH Use ARCH for program instead of container's own architecture.

-s, --namespaces=FLAGS Don't attach to all the namespaces of the container but just to the following OR'd list of flags: MOUNT, PID, UTSNAME, IPC, USER or NETWORK. WARNING: Using -s implies -e with all privileges elevated, it may therefore leak privileges into the container. Use with care.

-R, --remount-sys-proc Remount /sys and /proc if not attaching to the mount namespace when using -s in order to properly reflect the correct namespace context. See the lxc-attach(1) manual page for details.

--clear-env Clear all environment variables before attaching. The attached shell/program will start with only container=lxc set.

--keep-env Keep all current environment variables. This is the current default behaviour, but is likely to change in the future.

-L, --pty-log=FILE Log pty output to FILE

-v, --set-var **Set an additional variable that is seen by the** attached program in the container. May be specified multiple times.

--keep-var Keep an additional environment variable. Only applicable if

–clear-env is specified. May be used multiple times.

-f, --rcfile=FILE Load configuration file FILE

Common options :

-o, --logfile=FILE Output log to FILE instead of stderr

-l, --logpriority=LEVEL Set log priority to LEVEL

-q, --quiet Don't produce any output

-P, --lxcpath=PATH Use specified container path

-?, –help Give this help list

--usage Give a short usage message

--version Print the version number

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

See the lxc-attach man page for further information.

class moler.cmd.unix.lxc_attach.**LxcAttach**(*name*, *connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*, *expected_prompt=None*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

LxcAttach command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.lxc_info module

LxcInfo command module.

moler.cmd.unix.lxc_info.**COMMAND_RESULT_2 = {'RESULT': {'State': 'RUNNING'}}**

==========HELP=MESSAGE==========
root@server~ >lxc-info –help Usage: lxc-info –name=NAME

lxc-info display some information about a container with the identifier NAME

Options :

-n, --name=NAME NAME of the container

-c, --config=KEY show configuration variable KEY from running container

-i, --ips shows the IP addresses

-p, --pid shows the process id of the init container

-S, --stats shows usage stats

-H, --no-humanize shows stats as raw numbers, not humanized

-s, --state shows the state of the container

--rcfile=FILE Load configuration file FILE

Common options :

-o, --logfile=FILE Output log to FILE instead of stderr

-l, --logpriority=LEVEL Set log priority to LEVEL

-q, --quiet Don't produce any output

-P, --lxcpath=PATH Use specified container path

-?, –help Give this help list

--usage Give a short usage message

--version Print the version number

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

See the lxc-info man page for further information.

class moler.cmd.unix.lxc_info.**LxcInfo**(*name*, *connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

LxcInfo command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.lxc_ls module

LxcLs command module.

moler.cmd.unix.lxc_ls.**COMMAND_RESULT_3 = {'RESULT': [['0xe000', '0xe000/0xe000', '0xe000/0x**

==========HELP=MESSAGE==========

root@0xe000:~ >lxc-ls –help Usage: lxc-ls [-P lxcpath] [–active] [–running] [–frozen] [–stopped] [–nesting] [-g groups] [–filter regex] [-1] [-P lxcpath] [–active] [–running] [–frozen] [–stopped] [–nesting] [-g groups] [–filter regex] [-f] [-P lxcpath] [–active] [–running] [–frozen] [–stopped] [–nesting] [-g groups] [–filter regex]

lxc-ls list containers

Options :

-1, --line	show one entry per line
-f, --fancy	use a fancy, column-based output

-F, –fancy-format comma separated list of columns to show in the fancy output valid columns are: NAME, STATE, PID, RAM, SWAP, AUTOSTART, GROUPS, INTERFACE, IPV4 and IPV6

--active	list only active containers
--running	list only running containers
--frozen	list only frozen containers
--stopped	list only stopped containers
--defined	list only defined containers
--nesting=NUM	list nested containers up to NUM (default is 5) levels of nesting
--filter=REGEX	filter container names by regular expression

-g –groups comma separated list of groups a container must have to be displayed

Common options :

-o, --logfile=FILE	Output log to FILE instead of stderr
-l, --logpriority=LEVEL	Set log priority to LEVEL
-q, --quiet	Don't produce any output
-P, --lxcpath=PATH	Use specified container path

-?, –help Give this help list

--usage	Give a short usage message
--version	Print the version number

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

See the lxc-ls man page for further information.

root@0xe000:~ >

class moler.cmd.unix.lxc_ls.**LxcLs**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Lxcls command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.

- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.md5sum module

Md5sum command module.

class moler.cmd.unix.md5sum.**Md5sum**(*connection*, *path*, *options=None*, *prompt=None*, *new-line_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.mkdir module

Mkdir command module.

class moler.cmd.unix.mkdir.**Mkdir**(*connection*, *path*, *options=None*, *prompt=None*, *new-line_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.mount module

Mount command module.

class moler.cmd.unix.mount.**Mount**(*connection*, *options=None*, *device=None*, *directory=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.mpstat module

Mpstat command module.

class moler.cmd.unix.mpstat.**Mpstat**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix mpstat command

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parses the command output.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.mv module

Mv command module.

class moler.cmd.unix.mv.**Mv**(*connection*, *src*, *dst*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.netstat module

class moler.cmd.unix.netstat.**Netstat**(*connection*, *options=''*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Netstat command class.

build_command_string()
Build command string from parameters passed to object.

Returns String representation of the command to send over a connection to the device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.nft module

nft command module.

class moler.cmd.unix.nft.**Nft**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.nmap module

Nmap command module.

class moler.cmd.unix.nmap.**Nmap**(*connection*, *ip*, *is_ping=False*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.ntpq module

Ntpq command module.

class moler.cmd.unix.ntpq.**Ntpq**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.openssl_s_client module

openssl s_client command module.

class moler.cmd.unix.openssl_s_client.**OpensslSClient**(*connection*, *options*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

openssl command s_client

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parse command output.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.openssl_x509_text_in module

openssl_x509_text_in module.

class moler.cmd.unix.openssl_x509_text_in.**OpensslX509TextIn**(*connection*, *path=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Command openssl_x509_text_in.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.passwd module

Passwd command module.

class moler.cmd.unix.passwd.**Passwd**(*connection*, *current_password=None*, *new_password=None*, *user=None*, *options=None*, *encrypt_password=True*, *newline_chars=None*, *runner=None*, *prompt=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.ping module

Ping command module.

class moler.cmd.unix.ping.**Ping**(*connection*, *destination*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.pkill module

Pkill command module.

class moler.cmd.unix.pkill.**Pkill**(*connection*, *name*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ps module

class moler.cmd.unix.ps.**Ps**(*connection=None*, *options=''*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix command ps.

build_command_string()

Builds string with command.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Parses output from command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.

- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.pwd module

pwd command module.

class moler.cmd.unix.pwd.**Pwd**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Processes line from output form connection/device.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.reboot module

Reboot command module.

class moler.cmd.unix.reboot.**Reboot**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. :param line: Line to parse, new lines are trimmed :param is_full_line: False for chunk of line; True on full line (NOTE: new line character removed) :return: None

moler.cmd.unix.rm module

Rm command module.

class moler.cmd.unix.rm.**Rm**(*connection*, *file*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

moler.cmd.unix.route module

Route command module.

class moler.cmd.unix.route.**Route**(*connection*, *prompt=None*, *newline_chars=None*, *options=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.run_script module

Run script command

class moler.cmd.unix.run_script.**RunScript**(*connection*, *script_command*, *error_regex=re.compile('error', re.IGNORECASE)*, *prompt=None*, *newline_chars=None*, *runner=None*, *success_regex=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.run_serial_proxy module

Run moler_serial_proxy command

class moler.cmd.unix.run_serial_proxy.**RunSerialProxy**(*connection*, *serial_devname*, *prompt=None*, *newline_chars=None*, *target_newline='n'*, *runner=None*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.scp module

SCP command module.

class moler.cmd.unix.scp.**Scp**(*connection*, *source*, *dest*, *password=''*, *options=''*, *prompt=None*, *newline_chars=None*, *known_hosts_on_failure='keygen'*, *encrypt_password=True*, *runner=None*, *repeat_password=True*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.sed module

Sed command module.

class moler.cmd.unix.sed.**Sed**(*connection*, *input_files*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*, *scripts=None*, *script_files=None*, *output_file=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.service module

Service command module.

class moler.cmd.unix.service.**Service**(*connection*, *options*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.sftp module

SFTP command module.

class moler.cmd.unix.sftp.**Sftp**(*connection*, *host*, *password*, *user=None*, *confirm_connection=True*, *source_path=None*, *destination_path=None*, *options=None*, *command=None*, *no_result=False*, *prompt=None*, *newline_chars=None*, *timeout=60*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.shasum module

Shasum command module.

class moler.cmd.unix.shasum.**Shasum**(*connection*, *path*, *options=None*, *cmd_kind='shasum'*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string form parameters.

Returns Command string

on_new_line(*line*, *is_full_line*)

Parses every line from connection.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.socat module

Socat command module.

class moler.cmd.unix.socat.**Socat**(*connection*, *input_options=None*, *output_options=None*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.ss module

class moler.cmd.unix.ss.**Ss**(*connection*, *options=''*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Ss command class.

build_command_string()

Build command string from parameters passed to object.

Returns String representation of the command to send over a connection to the device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.ssh module

Ssh command module.

class moler.cmd.unix.ssh.**Ssh**(*connection*, *login=None*, *password=None*, *host='0'*, *prompt=None*, *expected_prompt='>'*, *port=0*, *known_hosts_on_failure='keygen'*, *set_timeout='export TMOUT=\"2678400\"'*, *set_prompt=None*, *term_mono='TERM=xterm-mono'*, *newline_chars=None*, *encrypt_password=True*, *runner=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *repeat_password=True*, *options='-o ServerAliveInterval=7 -o ServerAliveCountMax=2'*, *failure_exceptions_indication=None*, *prompt_after_login=None*, *send_enter_after_connection=True*, *username=None*, *permission_denied_key_pass_keyboard='ssh-keygen -f "~/.ssh/known_hosts" -R "{host}"'*, *allow_override_denied_key_pass_keyboard=True*, *suffix=None*)

Bases: *moler.cmd.unix.genericctelnetssh.GenericTelnetSsh*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.sshkeygen module

Ssh-keygen command module.

class moler.cmd.unix.sshkeygen.**Sshkeygen**(*connection*, *prompt=None*, *newline_chars=None*, *options=None*, *file='/home/ute/.ssh/id_rsa'*, *passphrase=''*, *overwrite=True*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.su module

Su command module.

class moler.cmd.unix.su.**Su**(*connection*, *login=None*, *options=None*, *password=None*, *prompt=None*, *expected_prompt=None*, *newline_chars=None*, *encrypt_password=True*, *target_newline='n'*, *runner=None*, *set_timeout=None*, *allowed_newline_after_prompt=False*, *set_prompt=None*, *prompt_after_login=None*, *cmd_object=None*, *cmd_class_name=None*, *cmd_params=None*, *repeat_password=None*)

Bases: *moler.cmd.unix.sudo.Sudo*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

moler.cmd.unix.sudo module

Sudo command module.

class moler.cmd.unix.sudo.**Sudo**(*connection*, *password=None*, *sudo_params=None*, *cmd_object=None*, *cmd_class_name=None*, *cmd_params=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *encrypt_password=True*, *expected_prompt=None*, *set_timeout=None*, *set_prompt=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *prompt_after_login=None*, *repeat_password=True*)

Bases: *moler.cmd.commandchangingprompt.CommandChangingPrompt*

Unix command sudo

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

start(*timeout=None*, **args*, ***kwargs*)

Start background execution of connection-observer.

moler.cmd.unix.sync module

Sync command module.

class moler.cmd.unix.sync.**Sync**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

moler.cmd.unix.systemctl module

Systemctl command module.

class moler.cmd.unix.systemctl.**Systemctl**(*connection*, *options=None*, *service=None*, *password=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *encrypt_password=True*)

Bases: *moler.cmd.unix.service.Service*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise.

Returns None.

moler.cmd.unix.tail module

Tail command module.

class moler.cmd.unix.tail.**Tail**(*connection*, *path*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *failure_only_in_first_line=True*)

Bases: *moler.cmd.unix.cat.Cat*

build_command_string()

Builds string with command.

Returns String with command.

moler.cmd.unix.tail_latest_file module

Tail latest file from the directory.

class moler.cmd.unix.tail_latest_file.**TailLatestFile**(*connection*, *directory*, *file_pattern='*'*, *prompt=None*, *newline_chars=None*, *runner=None*, *time_for_failure=0.1*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

is_failure_indication(*line*)

Check if line has info about failure indication.

Parameters **line** – Line from device

Returns None if line does not match regex with failure, Match object if line matches the failure regex.

on_new_line(*line*, *is_full_line*)

Parse line from command output.

Parameters

- **line** – Line from device
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.tar module

tar command module.

class moler.cmd.unix.tar.**Tar**(*connection*, *options*, *file*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

moler.cmd.unix.tcpdump module

Tcpdump command module.

class moler.cmd.unix.tcpdump.**Tcpdump**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *break_exec_regex=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Build command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.tee module

Tee command module.

class moler.cmd.unix.tee.**Tee**(*connection*, *path*, *content*, *border='END_OF_FILE'*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unix tee command

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)

Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.telnet module

Telnet command module.

class moler.cmd.unix.telnet.**Telnet**(*connection*, *host*, *login=None*, *password=None*, *port=0*, *prompt=None*, *expected_prompt='^>\s*'*, *set_timeout='export TMOUT=\"2678400\"'*, *set_prompt=None*, *term_mono='TERM=xterm-mono'*, *prefix=None*, *newline_chars=None*, *cmds_before_establish_connection=None*, *cmds_after_establish_connection=None*, *telnet_prompt='^\s*telnet>\s*'*, *encrypt_password=True*, *runner=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *repeat_password=True*, *failure_exceptions_indication=None*, *prompt_after_login=None*, *send_enter_after_connection=True*, *username=None*)

Bases: *moler.cmd.unix.genericctelnetssh.GenericTelnetSsh*

build_command_string()

Builds command string from parameters passed to object.

Returns String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.top module

Top command module.

class moler.cmd.unix.top.**Top**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*, *n=1*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.traceroute module

Traceroute command module.

class moler.cmd.unix.traceroute.**Traceroute**(*connection*, *destination*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.tshark module

Tshark command module.

class moler.cmd.unix.tshark.**Tshark**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.uname module

Uname command module.

class moler.cmd.unix.uname.**Uname**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.unxz module

class moler.cmd.unix.unxz.**Unxz**(*connection*, *xz_file*, *options=''*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unxz command class.

build_command_string()
Build command string from parameters passed to object. :return: String representation of the command to send over a connection to the device.

moler.cmd.unix.unzip module

class moler.cmd.unix.unzip.**Unzip**(*connection*, *zip_file*, *extract_dir=None*, *options=''*, *overwrite=False*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

Unzip command class.

build_command_string()
Build command string from parameters passed to object.

Returns String representation of the command to send over a connection to the device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None.

moler.cmd.unix.uptime module

Uptime command module.

class moler.cmd.unix.uptime.**Uptime**(*connection*, *options=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Builds command string from parameters passed to object. :return: String representation of command to send over connection to device.

on_new_line(*line*, *is_full_line*)
Put your parsing code here. :param line: Line to process, can be only part of line. New line chars are removed from line. :param is_full_line: True if line had new line chars, False otherwise :return: None

moler.cmd.unix.useradd module

Useradd command module.

class moler.cmd.unix.useradd.**Useradd**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*, *defaults=False*, *user=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.userdel module

Userdel command module.

class moler.cmd.unix.userdel.**Userdel**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*, *options=None*, *user=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.w module

class moler.cmd.unix.w.**W**(*connection*, *options=''*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

W command class.

build_command_string()

Build command string from parameters passed to object. Usage of paramter -h in options is going to be ignored.

Returns String representation of the command to send over a connection to the device.

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.unix.wget module

Wget command module.

class moler.cmd.unix.wget.**Wget**(*connection*, *options*, *log_progress_bar=False*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.which module

Which command module.

class moler.cmd.unix.which.**Which**(*connection*, *names*, *show_all=None*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.whoami module

Whoami command module.

class moler.cmd.unix.whoami.**Whoami**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)
Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed

- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

moler.cmd.unix.zip module

Zip command module.

class moler.cmd.unix.zip.**Zip**(*connection*, *options*, *file_name*, *zip_file*, *timeout=60*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.cmd.unix.genericunix.GenericUnixCommand*

build_command_string()

Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)

Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – False for chunk of line; True on full line (NOTE: new line character removed)

Returns None

Module contents

Package for implementing Unix/Linux commands.

Submodules

moler.cmd.commandchangingprompt module

Generic command class for commands change prompt

class moler.cmd.commandchangingprompt.**CommandChangingPrompt**(*connection*, *prompt*, *expected_prompt*, *newline_chars=None*, *runner=None*, *set_timeout=None*, *set_prompt=None*, *target_newline='n'*, *allowed_newline_after_prompt=False*, *prompt_after_login=None*)

Bases: *moler.cmd.commandtextualgeneric.CommandTextualGeneric*

Base class for textual commands to change prompt.

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

on_new_line(*line*, *is_full_line*)
Parses the output of the command.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.cmd.commandtextualgeneric module

Generic class for all command with textual output.

class moler.cmd.commandtextualgeneric.**CommandTextualGeneric**(*connection*, *prompt=None*, *newline_chars=None*, *runner=None*)

Bases: *moler.command.Command*

Base class for textual commands.

break_cmd(*silent=False*, *force=False*)
Send ctrl+c to device to break command execution.

Parameters

- **silent** – set False to log info the break is not sent
- **force** – set True to break cmd even if the command does not run

Returns None

break_exec_regex
Getter for break_exec_regex

Returns Regex object or None

build_command_string()
Returns string with command constructed with parameters of object.

Returns String with command.

cancel()
Called by framework to cancel the command.

Returns False if already cancelled or already done, True otherwise.

command_string
Getter for command_string.

Returns String with command_string.

data_received(*data*, *recv_time*)
Called by framework when any data are sent by device.

Parameters

- **data** – List of strings sent by device.

- **recv_time** – time stamp with the moment when the data was read from connection. Time is given as datetime.datetime instance.

Returns None.

has_any_result()
Checks if any result was already set by command.

Returns True if current_ret has collected any data. Otherwise False.

has_endline_char(*line*)
Method to check if line has chars of new line at the right side.

Parameters **line** – String to check.

Returns True if any new line char was found, False otherwise.

is_end_of_cmd_output(*line*)
Checks if end of command is reached.

Parameters **line** – Line from device.

Returns True if end of command is reached, False otherwise.

on_done()
Callback called by framework when command is just about to finish.

Returns None

on_failure()
Callback called by framework when command is just about to finish with failure. Set ret is called.

Returns None

on_new_line(*line*, *is_full_line*)
Method to parse command output. Will be called after line with command echo. Write your own implementation but don't forget to call on_new_line from base class in most cases.

Parameters

- **line** – Line to parse, new lines are trimmed
- **is_full_line** – True if new line character was removed from line, False otherwise

Returns None

on_success()
Callback called by framework when command is just about to finish with success. Set ret is called.

Returns None

on_timeout()
Callback called by framework when timeout occurs.

Returns None.

send_command()
Sends command string over connection.

Returns None

send_enter()
Sends enter over connection. :return: None

set_exception(*exception*)
Set exception object as failure for command object.

Parameters **exception** – An exception object to set.

Returns None.

Module contents

Package for implementing different commands based on Moler Command.

class moler.cmd.**RegexHelper**
Bases: object

Class to help with working with regular expressions.

get_match()
Returns match object.

Returns Match object.

group(*number*)
Returns group from match object.

Parameters **number** – Number or name of match object.

Returns Match object.

groupdict()
Returns groupdict from match object.

Returns Groupdict from match object.

groups()
Returns groups from match object.

Returns Groups from match object.

match(*pattern*, *string*, *flags=0*)
Matches for passed pattern in passed string.

Parameters

- **pattern** – Pattern to find. Regular expression.
- **string** – String to scan through to find the pattern.
- **flags** – Flags for search.

Returns Match object.

match_compiled(*compiled*, *string*, *raise_if_compiled_is_none=False*)
Matches for passed pattern in passed string.

Parameters

- **compiled** – Compiled regular expression pattern to find.
- **string** – String to scan through to find the pattern.
- **raise_if_compiled_is_none** – set True to raise a WrongUsage if compiled is None. If False then return None.

Returns Match object.

search(*pattern*, *string*, *flags=0*)
Searches for passed pattern in passed string.

Parameters

- **pattern** – Pattern to find. Regular expression.
- **string** – String to scan through to find the pattern.
- **flags** – Flags for search.

Returns Match object.

search_compiled(*compiled*, *string*, *raise_if_compiled_is_none=False*)
Searches for passed pattern in passed string.

Parameters

- **compiled** – Compiled regular expression pattern to find.
- **string** – String to scan through to find the pattern.
- **raise_if_compiled_is_none** – set True to raise a WrongUsage if compiled is None. If False then return None.

Returns Match object.

moler.config package

Submodules

moler.config.connections module

Connections related configuration

moler.config.connections.**clear**()
Cleanup configuration related to connections

moler.config.connections.**define_connection**(*name*, *io_type*, ***constructor_kwargs*)
Assign name to connection specification.

You should provide name that is meaningful in context of your application. Let's say you have 3 servers hosting HTTP under 10.20.30.41 .. 43 Then you may name/define your connections like:

```
www_svr1  io_type=tcp, host=10.20.30.41, port=80
www_svr2  io_type=tcp, host=10.20.30.42, port=80
www_svr3  io_type=tcp, host=10.20.30.43, port=80
```

Thanks to such naming you could establish connection to server like:

```
svr1_conn = get_connection(name="www_svr_1")
svr1_conn.open()
```

moler.config.connections.**mlr_conn_no_encoding**(*moler_conn_class*, *name*)

moler.config.connections.**mlr_conn_no_encoding_partial_clean_vt100**(*moler_conn_class*, *name*)
Cleans some VT100 control-codes

moler.config.connections.**mlr_conn_utf8**(*moler_conn_class*, *name*)

moler.config.connections.**mlr_conn_utf8_with_clean_vt100**(*moler_conn_class*, *name*)

moler.config.connections.**register_builtin_connections**(*connection_factory*, *moler_conn_class*)

moler.config.connections.**set_default_variant**(*io_type*, *variant*)
Set variant to use as default when requesting 'io_type' connection

moler.config.connections.**set_defaults**()
Set defaults for connections configuration

moler.config.devices module

Package Open Source functionality of Moler.

moler.config.devices.**clear**()
Cleanup configuration related to devices

moler.config.devices.**define_device**(*name*, *device_class*, *connection_desc*, *connection_hops*, *initial_state=None*, *lazy_cmds_events=False*, *additional_params=None*)
Assign name to device specification.

moler.config.devices.**set_default_connection**(*io_type*, *variant*)
Set connection to use as default when requesting 'device' without connection specification

moler.config.loggers module

Configure logging for Moler's needs

class moler.config.loggers.**MolerMainMultilineWithDirectionFormatter**(*fmt*, *datefmt=None*)
Bases: *moler.config.loggers.MultilineWithDirectionFormatter*

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class moler.config.loggers.**MultilineWithDirectionFormatter**(*fmt=None*, *datefmt=None*)
Bases: logging.Formatter

We want logs to have non-overlapping areas timestamp area TTTTTTTTTTT transfer direction area > (shows send '>' or receive '<') log message area MMMMMMMMMM It should look like: TTTTTTTTTTTTTTT D MMMMMMMMMMMMMMM 01 00:36:09.581 >|cat my_file.txt 01 00:36:09.585 <|This is

|multiline |content

This formatter allows to use %(transfer_direction)s inside format

format(*record*)
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class moler.config.loggers.**RawDataFormatter**
Bases: object

format(*record*)
We want to take data from log_record.msg as bytes

class moler.config.loggers.**RawFileHandler**(**args*, ***kwargs*)
Bases: logging.FileHandler

emit(*record*)

Emit a record. We don't want base class implementation since we don't want to do: stream.write(self.terminator) We are not adding any

to bytes-message from record.

class moler.config.loggers.**RawTraceFormatter**
Bases: *moler.config.loggers.RawDataFormatter*

format(*record*)
We want to see info about binary data log-record

class moler.config.loggers.**SpecificLevelFilter**(*level*)
Bases: object

filter(*logRecord*)

class moler.config.loggers.**TracedIn**(*logger_name*)
Bases: object

Decorator to allow for tracing method/function invocation It sends function name and parameters into logger given as decorator parameter sends with loglevel=TRACE Decorator is active only when environment variable MOLER_DEBUG_LEVEL = TRACE ex.: @TracedIn('moler') def method(self, arg1, arg2):

moler.config.loggers.**change_logging_suffix**(*suffix=None*, *logger_name=None*)
Change logging suffix. :param suffix: new suffix for log files. None for no suffix. :param logger_name: name of logger. None for all loggers. :return: None

moler.config.loggers.**configure_debug_level**(*level=None*)
Configure debug_level based on environment variable MOLER_DEBUG_LEVEL We use additional env variable besides MOLER_CONFIG to allow for quick/temporary change since debug level is intended also for troubleshooting

moler.config.loggers.**configure_device_logger**(*connection_name*, *propagate=False*)
Configure logger with file storing connection's log

moler.config.loggers.**configure_moler_main_logger**()
Configure main logger of Moler

moler.config.loggers.**configure_moler_threads_logger**()
Configure threads logger of Moler

moler.config.loggers.**configure_runner_logger**(*runner_name*)
Configure logger with file storing runner's log

moler.config.loggers.**create_logger**(*name*, *log_file=None*, *log_level=4*, *log_format='%(asctime)s %(levelname)-10s: |%(message)s'*, *datefmt=None*)
Creates Logger with (optional) file writer :param name: Logger name :param log_file: Path to logfile. Final logfile location is logging_path + log_file :param log_level: only log records with equal and greater level will be accepted for storage in log :param log_format: layout of log file, default is "%(asctime)s %(levelname)-10s: |%(message)s" :param datefmt: format the creation time of the log record :return: None

moler.config.loggers.**debug_level_or_info_level**()
If debugging is active we want to have details inside logs otherwise we want to keep them small

moler.config.loggers.**get_error_log_stack**()
Get how many functions stack you want to log when error is logged. :return: True to get log of all

moler.config.loggers.**get_logging_path**()

moler.config.loggers.**reconfigure_logging_path**(*log_path*)
Set up new logging path when Moler script is running :param log_path: new log path when logs will be stored :return: None

moler.config.loggers.**set_backup_count**(*backup_count*)

Set maximum number of files of logs to rotate for rotating logging. If parameter is not an int number then the function does not change any value.

Parameters **backup_count** – int number of how many files to keep to rotate logs.

Returns None

moler.config.loggers.**set_compress_after_rotation**(*compress_after_rotation*)
Set True to compress file after log rotation. :param compress_after_rotation: True to compress, False otherwsie :return: None

moler.config.loggers.**set_compress_command**(*compress_command*)
Set compress command. :param compress_command: String with compress command with two fields {compressed} and {log_input} :return: None

moler.config.loggers.**set_compressed_file_extension**(*compressed_file_extension*)
Set compressed file extension. :param compressed_file_extension: String with file extension, for example ".zip" :return: None

moler.config.loggers.**set_date_format**(*format*)

moler.config.loggers.**set_error_log_stack**(*error_log_stack=False*)
Set how many functions stack you want to log when error is logged. :param error_log_stack: True to get all functions, False to get the last one. :return: None

moler.config.loggers.**set_interval**(*interval*)
Set interval for rotating logging. If parameter is not an int number then the function does not change any value. :param interval: int number in bytes or seconds :return: None

moler.config.loggers.**set_kind**(*kind*)
Set kind of logging. :param kind: None for plain logger, 'time' to time rotating, 'size' for size rotating. :return: None

moler.config.loggers.**set_logging_path**(*path*)

moler.config.loggers.**set_write_mode**(*mode*)

moler.config.loggers.**setup_new_file_handler**(*logger_name*, *log_level*, *log_filename*, *formatter*, *filter=None*)
Sets up new file handler for given logger :param logger_name: name of logger to which filelogger is added :param log_level: logging level :param log_filename: path to log file :param formatter: formatter for file logger :param filter: filter for file logger :return: logging.FileHandler object

moler.config.loggers.**want_debug_details**()
Check if we want to have debug details inside logs

moler.config.loggers.**want_raw_logs**()

moler.config.runners module

Runners related configuration

moler.config.runners.**clear**()
Cleanup configuration related to runners

moler.config.runners.**register_builtin_runners**(*runner_factory*)

moler.config.runners.**set_default_variant**(*variant*)
Set variant to use as default when requesting runner

Module contents

Moler related configuration

moler.config.**clear**()
Cleanup Moler's configuration

moler.config.**configs_are_same**(*config_list*, *config_to_find*)
Utility function to check if two configs are identical (deep comparison)

Parameters

- **config_list** – list of configs to compare
- **config_to_find** – second config to compare

Returns bool, True if config_to_find is in config_list, False otherwise.

moler.config.**load_config**(*config=None*, *from_env_var=None*, **args*, ***kwargs*)
Load Moler's configuration from config file.

Load Moler's configuration from config file. :param config: either dict or config filename directly provided (overwrites 'from_env_var' if both given). :type config: dict or str :param from_env_var: name of environment variable storing config filename (file is in YAML format) :return: None

moler.config.**load_connection_from_config**(*config*)

moler.config.**load_device_from_config**(*config*, *add_only=False*)

moler.config.**load_logger_from_config**(*config*)

moler.config.**read_configfile**(*path*)
Context manager that reads content of configuration file into string

Parameters **path** – location of configuration file

Returns configuration file content as string

moler.config.**read_yaml_configfile**(*path*)
Read and convert YAML into dictionary

Parameters **path** – location of yaml file

Returns configuration as a python dictionary

moler.config.**reconfigure_logging_path**(*logging_path*)
Set up new logging path when Moler script is running :param logging_path: new log path when logs will be stored :return:

moler.device package

Submodules

moler.device.abstract_device module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.abstract_device.**AbstractDevice**
Bases: object

add_neighbour_device(*neighbour_device*, *bidirectional=True*)
Adds neighbour device to this device.

Parameters

- **neighbour_device** – device object or string with device name.
- **bidirectional** – If True then this device will be added to f_device.

Returns None

configure_logger(*name*, *propagate*)
Configures logger.

Parameters

- **name** – Name of logger
- **propagate** – Set True if you want to propagate logs, False otherwise (recommended)

Returns None

current_state
Getter of current state.

Returns String with the name of current state.

establish_connection()
Establishes connection to real device.

Returns None

get_cmd(*cmd_name*, *cmd_params=None*, *check_state=True*, *for_state=None*)
Returns instance of command connected with the device.

Parameters

- **cmd_name** – name of commands, name of class (without package), for example "cd".
- **cmd_params** – dict with command parameters.
- **check_state** – if True then before execute of command the state of device will be check if the same as when command was created. If False the device state is not checked.
- **for_state** – if None then command object for current state is returned, otherwise object for for_state is returned.

Returns Instance of command

get_event(*event_name*, *event_params=None*, *check_state=True*, *for_state=None*)
Return instance of event connected with the device.

Parameters

- **event_name** – name of event, name of class (without package).
- **event_params** – dict with event parameters.
- **check_state** – if True then before execute of event the state of device will be check if the same as when event was created. If False the device state is not checked.
- **for_state** – if None then event object for current state is returned, otherwise object for for_state is returned.

Returns Event object

get_neighbour_devices(*device_type*)
Returns list of neighbour devices of passed type.

Parameters **device_type** – type of device. If None then all neighbour devices will be returned.

Returns list of devices.

goto_state(*state*, *timeout=-1*, *rerun=0*, *send_enter_after_changed_state=False*, *log_stacktrace_on_fail=True*, *keep_state=True*)
Goes to state.

Parameters

- **state** – name of state to go.
- **timeout** – Time in seconds when break transitions if still not success.
- **rerun** – How many times try to rerun command(s) when device is still not in requested state.
- **send_enter_after_changed_state** – Set True to send enter after enters proper state.
- **log_stacktrace_on_fail** – Set True to log exceptions if command to enter state failed.
- **keep_state** – if True and state is changed without goto_state then device tries to change state to state

defined by goto_state. :return: None

has_established_connection()
Checks if connection is established to real device.

Returns True if connection is established, False otherwise.

name
Getter of device name.

Returns String with the device name.

on_connection_lost(*connection*)
Method to call by Moler framework when connection is lost.

Parameters **connection** – Connection object.

Returns None

on_connection_made(*connection*)
Method to call by Moler framework when connection is established.

Parameters **connection** – Connection object.

Returns None

public_name
Getter for publicly used device name.

Internal name of device (.name attribute) may be modified by device itself in some circumstances (to not overwrite logs). However, public_name is guaranteed to be preserved as it was set by external/client code.

Returns String with the device alias name.

register_device_removal_callback(*callback*)
Registers callable to be called (notified) when device is removed.

Parameters **callback** – callable to call when device is being removed.

Returns None

remove()
Closes device, if any command or event is attached to this device they will be finished.

Returns None

run(*cmd_name*, ***kwargs*)
Wrapper for simple use command: creates command, runs it and waits till it ends.

Parameters

- **cmd_name** – name of class of command.
- **kwargs** – dict with parameters for command constructor.

Returns object from command get_result (mainly dict).

start(*cmd_name*, ***kwargs*)
Wrapper for simple use command: creates command and runs it.

Parameters

- **cmd_name** – name of class of command.
- **kwargs** – dict with parameters for command constructor.

Returns command object

moler.device.adbremote module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.adbremote.**AdbRemote**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)
Bases: *moler.device.unixremote.UnixRemote*

AdbRemote device class.

Example of device in yaml configuration file: -without PROXY_PC:

ADB_1: DEVICE_CLASS: moler.device.adbremote.AdbRemote CONNECTION_HOPS:

UNIX_LOCAL:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

UNIX_REMOTE:

ADB_SHELL: execute_command: adb_shell # default value; default command is: adb shell command_params:

serial_number: 'f57e6b7d' # to create: adb -s f57e6b7d shell expected_prompt: 'shell@adbhost: $'

ADB_SHELL:

UNIX_REMOTE: execute_command: exit # default value

```
adb_shell = 'ADB_SHELL'
adb_shell_root = 'ADB_SHELL_ROOT'
```

moler.device.adbremote2 module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.adbremote2.**AdbRemote2**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.unixremote2.UnixRemote2*

AdbRemote device class.

Example of device in yaml configuration file:

- with PROXY_PC and io "terminal": ADB_1:

  DEVICE_CLASS: moler.device.adbremote2.AdbRemote2 CONNECTION_HOPS:

  UNIX_LOCAL:

  PROXY_PC: execute_command: ssh # default value command_params:

  expected_prompt: proxy_pc_prompt host: host_ip login: login password: password

  PROXY_PC:

  UNIX_REMOTE: execute_command: ssh # default value command_params:

  expected_prompt: unix_remote_prompt host: host_ip login: login password: password

  UNIX_REMOTE:

  ADB_SHELL: execute_command: adb_shell # default value command_params:

  serial_number: 'f57e6b7d'

- with PROXY_PC and remote-access-io like "sshshell": ADB_1:

  DEVICE_CLASS: moler.device.adbremote2.AdbRemote2 CONNECTION_DESC:

  io_type: sshshell host: host_ip login: login password: password

  CONNECTION_HOPS:

  PROXY_PC:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

UNIX_REMOTE:

ADB_SHELL: execute_command: adb_shell # default value command_params:

serial_number: 'f57e6b7d'

-without PROXY_PC and io "terminal":

ADB_1: DEVICE_CLASS: moler.device.adbremote2.AdbRemote2 CONNECTION_HOPS:

UNIX_LOCAL:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

UNIX_REMOTE:

ADB_SHELL: execute_command: adb_shell # default value command_params:

serial_number: 'f57e6b7d'

-without PROXY_PC and remote-access-io like "sshshell":

ADB_1: DEVICE_CLASS: moler.device.adbremote2.AdbRemote2 CONNECTION_DESC:

io_type: sshshell host: host_ip login: login password: password

CONNECTION_HOPS:

UNIX_REMOTE:

ADB_SHELL: execute_command: adb_shell # default value command_params:

serial_number: 'f57e6b7d'

moler.device.atremote module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.atremote.**AtRemote**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.unixremote.UnixRemote*

AtRemote device class.

Example of device in yaml configuration file: -without PROXY_PC:

AT_1: DEVICE_CLASS: moler.device.atremote.AtRemote CONNECTION_HOPS:

UNIX_LOCAL:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

UNIX_REMOTE:

AT_REMOTE: execute_command: plink_serial # default value command_params:

serial_devname: 'COM5'

AT_REMOTE:

UNIX_REMOTE: execute_command: ctrl_c # default value

```
at_remote = 'AT_REMOTE'
```

moler.device.device module

Package Open Source functionality of Moler.

class moler.device.device.**DeviceFactory**

Bases: object

classmethod create_all_devices(*ignore_exception=False*)

Creates all devices from config.

Parameters **ignore_exception** – Set False to raise an exception when cannot create device. True to ignore and create other devices.

Returns None

classmethod differences_between_devices_descriptions(*already_device_name*, *requested_device_def*)

Checks if two device description are the same.

Parameters

- **already_device_name** – Name of device already created by Moler
- **requested_device_def** – Description od device provided to create. The name is the same as above.

Returns Empty string if descriptions are the same, if not the string with differences.

classmethod forget_device_handler(*device_name*)

Function to call to forget device.

Parameters **device_name** – Name of device

Returns None

classmethod get_cloned_device(*source_device*, *new_name*, *initial_state=None*, *establish_connection=True*, *lazy_cmds_events=False*, *io_connection=None*, *additional_params=None*)

Creates (if necessary) and returns new device based on existed device.

Parameters

- **source_device** – Reference to base device or name of base device.
- **new_name** – Name of new device.
- **initial_state** – Initial state of created device. If None then state of source device will be used.
- **establish_connection** – True to open connection, False if it does not matter.

- **lazy_cmds_events** – set False to load all commands and events when device is initialized, set True to load commands and events when they are required for the first time.
- **io_connection** – connection for device. Ignored if device is already created.
- **additional_params** – dict with parameter(s) specific for the device. Will be passed to the constructor. If no specific parameter then set to None.

Returns Device object.

classmethod get_device(*name=None*, *device_class=None*, *connection_desc=None*, *connection_hops=None*, *initial_state=None*, *establish_connection=True*, *lazy_cmds_events=False*, *io_connection=None*, *additional_params=None*)

Return connection instance of given io_type/variant.

Parameters

- **name** – name of device defined in configuration.
- **device_class** – 'moler.device.unixlocal.UnixLocal', 'moler.device.unixremote.UnixRemote', ...
- **connection_desc** – 'io_type' and 'variant' of device connection. Ignored if device already created or argument io_connection is passed.
- **connection_hops** – connection hops to create device SM.
- **initial_state** – initial state for device e.g. UNIX_REMOTE.
- **establish_connection** – True to open connection, False if it does not matter.
- **lazy_cmds_events** – set False to load all commands and events when device is initialized, set True to load commands and events when they are required for the first time.
- **io_connection** – connection for device. Ignored if device is already created.
- **additional_params** – dict with parameter(s) specific for the device. Will be passed to the constructor. If no specific parameter then set to None.

Returns requested device.

classmethod get_devices_by_type(*device_type*)

Returns list of devices filtered by device_type.

Parameters **device_type** – type of device. If None then return all devices.

Returns List of devices. Can be an empty list.

classmethod remove_all_devices(*clear_device_history=False*)

Remove all created devices.

Parameters **clear_device_history** – set True to clear the history of devices. Caution: you may overwrite your logs!

Returns None

classmethod remove_device(*name*)

Removes device. All commands and events are being finished when device is removed.

Parameters **name** – name of device..

Returns None.

classmethod was_any_device_deleted()
Checks if any device was deleted. :return: True if any device was deleted, False otherwise

moler.device.juniper_ex module

JuniperEX module.

class moler.device.juniper_ex.**JuniperEX**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)
Bases: *moler.device.junipergeneric.JuniperGeneric*

Juniperex device class.

Example of device in yaml configuration file: - with PROXY_PC:

JUNIPER_EX_PROXY_PC: DEVICE_CLASS: moler.device.juniper_ex.JuniperEX CONNECTION_HOPS:

PROXY_PC:

CLI: execute_command: ssh command_params:

host: cli_host login: cli_login password: password

UNIX_LOCAL:

PROXY_PC: execute_command: ssh command_params:

expected_prompt: "proxy_pc#" host: proxy_pc_host login: proxy_pc_login password: password

CLI:

PROXY_PC: execute_command: exit command_params:

expected_prompt: "proxy_pc#"

- without PROXY_PC: JUNIPER_EX:

DEVICE_CLASS: moler.device.juniper_ex.JuniperEX CONNECTION_HOPS:

UNIX_LOCAL:

CLI: execute_command: ssh # default value command_params:

host: cli_host login: cli_login password: password

moler.device.junipergeneric module

Juniper Generic module.

class moler.device.junipergeneric.**JuniperGeneric**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)
Bases: *moler.device.proxy_pc.ProxyPc*

Junipergeneric device class.

cli = 'CLI'

configure = 'CONFIGURE'

moler.device.pdu_aten module

PduAten device class.

class moler.device.pdu_aten.**PduAten**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.proxy_pc.ProxyPc*

PDU Aten device class.

Example of device in yaml configuration file: - with PROXY_PC:

PDU_1: DEVICE_CLASS: moler.device.pdu_aten.PduAten CONNECTION_HOPS:

PROXY_PC:

PDU: execute_command: telnet # default value command_params:

host: 10.0.0.1

PDU:

PROXY_PC: execute_command: exit_telnet # default value command_params:

expected_prompt: proxy_pc.*>

UNIX_LOCAL:

PROXY_PC: execute_command: ssh # default value command_params:

expected_prompt: proxy_pc.*> host: 10.0.0.2 login: user password: password

-without PROXY_PC:

PDU_1: DEVICE_CLASS: moler.device.pdu_aten.PduAten CONNECTION_HOPS:

UNIX_LOCAL:

PDU: execute_command: telnet # default value command_params:

host: 10.0.0.1

pdu = 'PDU'

moler.device.proxy_pc module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.proxy_pc.**ProxyPc**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.unixlocal.UnixLocal*

proxy_pc = 'PROXY_PC'

moler.device.proxy_pc2 module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.proxy_pc2.**ProxyPc2**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.unixlocal.UnixLocal*

goto_state(*state*, **args*, ***kwargs*)
Goes to specific state.

on_connection_lost(*connection*)
Execute action when connection lost. :param connection: device connection. :return: Nothing.

on_connection_made(*connection*)
Execute action when connection made. :param connection: device connection. :return: Nothing.

moler.device.proxy_pc2.**want_local_unix_state**(*io_type=None*, *io_connection=None*)
Check if device is intended to work with local machine or remote ones only. :return: True for local.

moler.device.scpi module

SCPI device class

class moler.device.scpi.**Scpi**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.proxy_pc.ProxyPc*

Scpi device class.

Example of device in yaml configuration file: - with PROXY_PC:

SCPI_1: DEVICE_CLASS: moler.device.scpi.Scpi CONNECTION_HOPS:

PROXY_PC:

SCPI: execute_command: telnet # default value command_params:

expected_prompt: SCPI> host: 10.0.0.1 port: 99999

SCPI:

PROXY_PC: execute_command: exit_telnet # default value command_params:

expected_prompt: proxy_pc.*>

UNIX_LOCAL:

PROXY_PC: execute_command: ssh # default value command_params:

expected_prompt: proxy_pc.*> host: 10.0.0.2 login: user password: password

-without PROXY_PC:

SCPI_1: DEVICE_CLASS: moler.device.scpi.Scpi CONNECTION_HOPS:

UNIX_LOCAL:

SCPI: execute_command: telnet # default value command_params:

expected_prompt: SCPI> host: 10.0.0.1 port: 99999

scpi = 'SCPI'

moler.device.state_machine module

class moler.device.state_machine.**StateMachine**(*model='self'*, *states=None*, *initial='initial'*, *transitions=None*, *send_event=False*, *auto_transitions=True*, *ordered_transitions=False*, *ignore_invalid_triggers=None*, *before_state_change=None*, *after_state_change=None*, *name=None*, *queued=False*, *prepare_event=None*, *finalize_event=None*, ***kwargs*)

Bases: transitions.core.Machine

set_state(*state*, *model=None*)

Sets state of StateMachine.

Parameters

- **state** – name of state to set.
- **model** – model.

Returns None.

moler.device.textualdevice module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.textualdevice.**TextualDevice**(*sm_params=None*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.abstract_device.AbstractDevice*

add_neighbour_device(*neighbour_device*, *bidirectional=True*)

Adds neighbour device to this device.

Parameters

- **neighbour_device** – device object or string with device name.
- **bidirectional** – If True then this device will be added to f_device.

Returns None

await_goto_state(*timeout=10*)
Waits till goto_state chain is empty. :param timeout: timeout in seconds. :return: None :raise DeviceChangeStateFailure: if the goto_state chain is not empty and timeout occurs.

build_trigger_to_state(*state*)

calc_timeout_for_command(*passed_timeout*, *configurations*)

cmds = 'cmd'

configure_logger(*name*, *propagate*)
Configures logger.

Parameters

- **name** – Name of logger
- **propagate** – Set True if you want to propagate logs, False otherwise (recommended)

Returns None

connection_hops = 'CONNECTION_HOPS'

current_state
Getter of current state.

Returns String with the name of current state.

disable_logging()
Disable logging incoming data. :return: None

enable_logging()
Enable logging incoming data. :return: None

establish_connection()
Establishes real connection to device. You have to call this method before device is full operable.

Returns None

events = 'event'

exchange_io_connection(*io_connection*)

classmethod from_named_connection(*connection_name*)

get_cmd(*cmd_name*, *cmd_params=None*, *check_state=True*, *for_state=None*)
Returns instance of command connected with the device. :param cmd_name: name of commands, name of class (without package), for example "cd". :param cmd_params: dict with command parameters. :param check_state: if True then before execute of command the state of device will be check if the same

as when command was created. If False the device state is not checked.

Parameters **for_state** – if None then command object for current state is returned, otherwise object for for_state is returned.

Returns Instance of command

get_configurations(*source_state*, *dest_state*)

get_event(*event_name*, *event_params=None*, *check_state=True*, *for_state=None*)
Return instance of event connected with the device. :param event_name: name of event, name of class (without package). :param event_params: dict with event parameters. :param check_state: if True then before execute of event the state of device will be check if the same

as when event was created. If False the device state is not checked.

Parameters **for_state** – if None then event object for current state is returned, otherwise object for for_state is returned.

Returns Event object

get_neighbour_devices(*device_type*)
Returns list of neighbour devices of passed type.

Parameters **device_type** – type of device. If None then all neighbour devices will be returned.

Returns list of devices.

get_observer(*observer_name*, *observer_type*, *observer_exception*, *check_state=True*, *for_state=None*, ***kwargs*)

get_prompt()

goto_state(*state*, *timeout=-1*, *rerun=0*, *send_enter_after_changed_state=False*, *log_stacktrace_on_fail=True*, *keep_state=False*, *timeout_multiply=1.0*)
Goes to specific state.

Parameters

- **state** – Name to state to change on the device.
- **timeout** – Timeout for changing state, if negative then timeout from commands are taken.
- **rerun** – How many times rerun the procedure before it fails.
- **send_enter_after_changed_state** – If True then enter is sent after state is changed. False nothing is sent.
- **log_stacktrace_on_fail** – Set True to have stacktrace in logs when failed, otherwise False.
- **keep_state** – if True and state is changed without goto_state then device tries to change state to state

defined by goto_state. :param timeout_multiply: how many times multiply passed timeout to get final timeout. :return: None :raise: DeviceChangeStateFailure if cannot change the state of device.

goto_state_bg(*state*, *keep_state=False*)
Starts to go to state. Returns immediately. :param state: name of state. :param keep_state: if True and state is changed without goto_state then device tries to change state to state defined by goto_state. :return: None

has_established_connection()
Checks if connection is established to real device.

Returns True if connection is established, False otherwise.

name
Getter of device name.

Returns String with the device name.

not_connected = 'NOT_CONNECTED'

on_connection_lost(*connection*)
Method to call by Moler framework when connection is lost.

Parameters **connection** – Connection object.

Returns None

on_connection_made(*connection*)
Method to call by Moler framework when connection is established.

Parameters **connection** – Connection object.

Returns None

public_name
Getter for publicly used device name.

Internal name of device (.name attribute) may be modified by device itself in some circumstances (to not overwrite logs). However, public_name is guaranteed to be preserved as it was set by external/client code.

Returns String with the device alias name.

remove()
Closes device, if any command or event is attached to this device they will be finished.

Returns None

run(*cmd_name*, ***kwargs*)
Wrapper for simple use:

return ux.run('cd', path="/home/user/")

Command/observer object is created locally

set_all_prompts_on_line(*value=True*)
Set True to check all prompts on line. False to interrupt after 1st prompt (default). :param value: True to check all prompts on line. False to interrupt after 1st prompt (default). :return: None

set_logging_suffix(*suffix*)
Set logging suffix. :param suffix: Suffix for device log. Use None for no suffix. :return: None

start(*cmd_name*, ***kwargs*)
Wrapper for simple use:

localhost_ping = ux.start('ping', destination="localhost", options="-c 5") ... result = localhost_ping.await_done()

result = await localhost_ping # py3 notation

Command/observer object is created locally

moler.device.unixlocal module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.unixlocal.**UnixLocal**(*sm_params=None*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)
Bases: *moler.device.textualdevice.TextualDevice*

UnixLocal device class.

Example of device in yaml configuration file: UNIX_1:

DEVICE_CLASS: moler.device.unixlocal.UnixLocal

UNIX_2: DEVICE_CLASS: moler.device.unixlocal.UnixLocal 'CONNECTION_HOPS': {

'UNIX_LOCAL': {

'UNIX_LOCAL_ROOT': {

"command_params": { "password": "root_password", "expected_prompt": r'local_root_prompt',

}

}

}

}

on_connection_lost(*connection*)

Execute action when connection lost. :param connection: device connection. :return: None.

on_connection_made(*connection*)

Execute action when connection made. :param connection: device connection. :return: None.

unix_local = 'UNIX_LOCAL'

unix_local_root = 'UNIX_LOCAL_ROOT'

moler.device.unixremote module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.unixremote.**UnixRemote**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.proxy_pc.ProxyPc*

UnixRemote device class.

Example of device in yaml configuration file: - with PROXY_PC:

UNIX_1: DEVICE_CLASS: moler.device.unixremote.UnixRemote CONNECTION_HOPS:

PROXY_PC:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

UNIX_REMOTE:

PROXY_PC: execute_command: exit # default value command_params:

expected_prompt: proxy_pc_prompt

UNIX_LOCAL:

PROXY_PC: execute_command: ssh # default value command_params:

expected_prompt: proxy_pc_prompt host: host_ip login: login password: password

-without PROXY_PC:

UNIX_1: DEVICE_CLASS: moler.device.unixremote.UnixRemote CONNECTION_HOPS:

UNIX_LOCAL:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

```
unix_remote = 'UNIX_REMOTE'
unix_remote_root = 'UNIX_REMOTE_ROOT'
```

moler.device.unixremote2 module

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

class moler.device.unixremote2.**UnixRemote2**(*sm_params*, *name=None*, *io_connection=None*, *io_type=None*, *variant=None*, *io_constructor_kwargs=None*, *initial_state=None*, *lazy_cmds_events=False*)

Bases: *moler.device.proxy_pc2.ProxyPc2*

UnixRemote2 device class.

Example of device in yaml configuration file: - with PROXY_PC and io "terminal":

UNIX_1: DEVICE_CLASS: moler.device.unixremote2.UnixRemote2 CONNECTION_HOPS:

UNIX_LOCAL:

PROXY_PC: execute_command: ssh # default value command_params:

expected_prompt: proxy_pc_prompt host: host_ip login: login password: password

PROXY_PC:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

- with PROXY_PC and remote-access-io like "sshshell": UNIX_1:

DEVICE_CLASS: moler.device.unixremote2.UnixRemote2 CONNECTION_DESC:

io_type: sshshell host: host_ip login: login password: password

CONNECTION_HOPS:

PROXY_PC:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

-without PROXY_PC and io "terminal":

UNIX_1: DEVICE_CLASS: moler.device.unixremote2.UnixRemote2 CONNECTION_HOPS:

UNIX_LOCAL:

UNIX_REMOTE: execute_command: ssh # default value command_params:

expected_prompt: unix_remote_prompt host: host_ip login: login password: password

-without PROXY_PC and remote-access-io like "sshshell":

UNIX_1: DEVICE_CLASS: moler.device.unixremote2.UnixRemote2 CONNECTION_DESC:

io_type: sshshell host: host_ip login: login password: password

(no need for CONNECTION_HOPS since we jump directly from NOT_CONNECTED to UNIX_REMOTE using sshshell)

on_connection_made(*connection*)
Execute action when connection made. :param connection: device connection. :return: Nothing.

Module contents

Moler's device has 2 main responsibilities: - be the factory that returns commands of that device - be the state machine that controls which commands may run in given state

moler.events package

Subpackages

moler.events.juniper package

Submodules

moler.events.juniper.genericshared_lineevent module

Generic Juniper module

class moler.events.juniper.genericshared_lineevent.**GenericJuniperLineEvent**(*detect_patterns*, *connection=None*, *till_occurs_times=-1*, *match='any'*, *runner=None*)

Bases: *moler.events.lineevent.LineEvent*

moler.events.juniper.genericshared_textualevent module

Generic Juniper module

class moler.events.juniper.genericshared_textualevent.**GenericJuniperEvent**(*connection=None*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.textualevent.TextualEvent*

Module contents

Package for implementing Juniper events.

moler.events.juniper_ex package

Submodules

moler.events.juniper_ex.genericjuniper_ex_lineevent module

Generic Juniper_Ex module

class moler.events.juniper_ex.genericjuniper_ex_lineevent.**GenericJuniperExLineEvent**(*detect_pa*
con-
nec-
tion=Non
till_occur
1,
match='a
run-
ner=Non

Bases: *moler.events.lineevent.LineEvent*

moler.events.juniper_ex.genericjuniper_ex_textualevent module

Generic Juniper_Ex module

class moler.events.juniper_ex.genericjuniper_ex_textualevent.**GenericJuniperExTextualEvent**(*c*
t
r
r

Bases: *moler.events.textualevent.TextualEvent*

Module contents

Package for implementing JuniperEX events.

moler.events.pdu_aten package

Module contents

Package for implementing PDU Aten events.

moler.events.scpi package

Submodules

moler.events.scpi.genericscpi_lineevent module

Generic Scpi module

class moler.events.scpi.genericscpi_lineevent.**GenericScpiLineEvent**(*detect_patterns*, *connection=None*, *till_occurs_times=-1*, *match='any'*, *runner=None*)

Bases: *moler.events.lineevent.LineEvent*

moler.events.scpi.genericscpi_textualevent module

Generic Scpi module

class moler.events.scpi.genericscpi_textualevent.**GenericScpiTextualEvent**(*connection=None*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.textualevent.TextualEvent*

Module contents

Package for implementing SCPI events.

moler.events.shared package

Submodules

moler.events.shared.genericshared_lineevent module

Generic Shared module

class moler.events.shared.genericshared_lineevent.**GenericSharedLineEvent**(*detect_patterns*, *connection=None*, *till_occurs_times=-1*, *match='any'*, *runner=None*)

Bases: *moler.events.lineevent.LineEvent*

moler.events.shared.genericshared_textualevent module

Generic Shared module

class moler.events.shared.genericshared_textualevent.**GenericSharedTextualEvent**(*connection=None*, *till_occurs_times=1*, *runner=None*)

Bases: *moler.events.textualevent.TextualEvent*

moler.events.shared.password_prompt module

class moler.events.shared.password_prompt.**PasswordPrompt**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.shared.genericshared_textualevent.GenericSharedTextualEvent*

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.events.shared.wait4 module

class moler.events.shared.wait4.**Wait4**(*detect_patterns*, *connection*, *match='any'*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.shared.genericshared_lineevent.GenericSharedLineEvent*

Module contents

Package for implementing Shared events.

moler.events.unix package

Submodules

moler.events.unix.advise_to_change_your_password module

class moler.events.unix.advise_to_change_your_password.**AdviseToChangeYourPassword**(*connection*, *till_occurs_t*, *1*, *runner=None*)

Bases: *moler.events.unix.genericunix_textualevent.GenericUnixTextualEvent*

on_new_line(*line*, *is_full_line*)
Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.events.unix.failed_login_counter module

class moler.events.unix.failed_login_counter.**FailedLoginCounter**(*connection*, *till_occurs_times=-1*, *runner=None*)
Bases: *moler.events.unix.genericunix_lineevent.GenericUnixLineEvent*

moler.events.unix.genericunix_lineevent module

Generic Unix/Linux module

class moler.events.unix.genericunix_lineevent.**GenericUnixLineEvent**(*detect_patterns*, *connection=None*, *till_occurs_times=-1*, *match='any'*, *runner=None*)
Bases: *moler.events.lineevent.LineEvent*

moler.events.unix.genericunix_textualevent module

Generic Unix/Linux module

class moler.events.unix.genericunix_textualevent.**GenericUnixTextualEvent**(*connection=None*, *till_occurs_times=-1*, *runner=None*)
Bases: *moler.events.textualevent.TextualEvent*

moler.events.unix.last_failed_login module

class moler.events.unix.last_failed_login.**LastFailedLogin**(*connection*, *till_occurs_times=-1*, *runner=None*)
Bases: *moler.events.unix.last_login.LastLogin*

moler.events.unix.last_login module

class moler.events.unix.last_login.**LastLogin**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_textualevent.GenericUnixTextualEvent*

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.events.unix.ping_no_response module

class moler.events.unix.ping_no_response.**PingNoResponse**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_lineevent.GenericUnixLineEvent*

moler.events.unix.ping_response module

class moler.events.unix.ping_response.**PingResponse**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_lineevent.GenericUnixLineEvent*

moler.events.unix.private_system module

class moler.events.unix.private_system.**PrivateSystem**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_lineevent.GenericUnixLineEvent*

moler.events.unix.shutdown module

class moler.events.unix.shutdown.**Shutdown**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_lineevent.GenericUnixLineEvent*

moler.events.unix.u_boot_crtm module

class moler.events.unix.u_boot_crtm.**UBootCrtm**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_textualevent.GenericUnixTextualEvent*

on_new_line(*line*, *is_full_line*)

Put your parsing code here.

Parameters

- **line** – Line to process, can be only part of line. New line chars are removed from line.
- **is_full_line** – True if line had new line chars, False otherwise

Returns None

moler.events.unix.wait4prompt module

class moler.events.unix.wait4prompt.**Wait4prompt**(*connection*, *prompt*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.shared.wait4.Wait4*

moler.events.unix.wait4prompts module

class moler.events.unix.wait4prompts.**Wait4prompts**(*connection*, *prompts*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_textualevent.GenericUnixTextualEvent*

change_prompts(*prompts*)

on_new_line(*line*, *is_full_line*)

Method to parse output from device. Write your own implementation to do something useful :param line: Line to parse, new lines are trimmed :param is_full_line: True if new line character was removed from line, False otherwise :return: None

moler.events.unix.warning_default_password module

class moler.events.unix.warning_default_password.**WarningDefaultPassword**(*connection*, *till_occurs_times=-1*, *runner=None*)

Bases: *moler.events.unix.genericunix_lineevent.GenericUnixLineEvent*

Module contents

Package for implementing Unix/Linux commands.

Submodules

moler.events.lineevent module

class moler.events.lineevent.**LineEvent**(*detect_patterns*, *connection=None*, *till_occurs_times=-1*, *match='any'*, *runner=None*)

Bases: *moler.events.textualevent.TextualEvent*

compile_patterns(*patterns*)

get_long_desc()
Returns string with description of event.

Returns String with description.

get_short_desc()
Returns string with description of event.

Returns String with description.

on_new_line(*line*, *is_full_line*)
Method to parse output from device. Write your own implementation to do something useful :param line: Line to parse, new lines are trimmed :param is_full_line: True if new line character was removed from line, False otherwise :return: None

start(*timeout=None*, **args*, ***kwargs*)
Start background execution of command.

moler.events.textualevent module

class moler.events.textualevent.**TextualEvent**(*connection=None*, *till_occurs_times=-1*, *runner=None*)
Bases: *moler.event.Event*

data_received(*data*, *recv_time*)
Called by framework when any data are sent by device.

Parameters

- **data** – List of strings sent by device.
- **recv_time** – time stamp with the moment when the data was read from connection.

Returns None.

event_occurred(*event_data*)
Sets event_data as new item of occurrence ret. :param event_data: data to set as value of occurrence. :return: None

is_new_line(*line*)
Method to check if line has chars of new line at the right side :param line: String to check :return: True if any new line char was found, False otherwise

on_new_line(*line*, *is_full_line*)
Method to parse output from device. Write your own implementation to do something useful :param line: Line to parse, new lines are trimmed :param is_full_line: True if new line character was removed from line, False otherwise :return: None

pause()
Pauses the event. Do not process till resume.

Returns None.

resume()
Resumes processing output from connection by the event.

Returns None.

Module contents

moler.io package

Subpackages

moler.io.asyncio package

Submodules

moler.io.asyncio.tcp module

External-IO connections based on asyncio.

The only 3 requirements for these connections are: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

class moler.io.asyncio.tcp.**AsyncioInThreadTcp**(*moler_connection*, *port*, *host='localhost'*, *receive_buffer_size=262144*, *logger=None*)

Bases: *moler.io.io_connection.IOConnection*

Implementation of TCP connection using asyncio running in dedicated thread.

close()
Close TCP connection.

Connection should allow for calling close on closed/not-open connection.

open()
Open TCP connection.

class moler.io.asyncio.tcp.**AsyncioTcp**(*moler_connection*, *port*, *host='localhost'*, *receive_buffer_size=262144*, *logger=None*)

Bases: *moler.io.io_connection.IOConnection*

Implementation of TCP connection using asyncio.

close()
Close TCP connection.

Connection should allow for calling close on closed/not-open connection.

forward_connection_read_data()

open()
Open TCP connection.

send(*data*)
Send data via TCP service.

Parameters **data** (*str*) – data

moler.io.asyncio.terminal module

External-IO connections based on asyncio.

The only 3 requirements for these connections are: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

class moler.io.asyncio.terminal.**AsyncioInThreadTerminal**(*moler_connection*, *cmd=None*, *first_prompt=None*, *dimensions=(100, 300)*, *logger=None*)

Bases: *moler.io.io_connection.IOConnection*

Implementation of Terminal connection using asyncio running in dedicated thread.

close()
Close TCP connection.

Connection should allow for calling close on closed/not-open connection.

logger

name

notify(*callback*, *when*)
Adds subscriber to list of functions to call :param callback: reference to function to call when connection is open/established :param when: connection state change :return: None

open()
Open TCP connection.

class moler.io.asyncio.terminal.**AsyncioTerminal**(*moler_connection*, *cmd=None*, *first_prompt=None*, *dimensions=(100, 300)*, *logger=None*)

Bases: *moler.io.io_connection.IOConnection*

Implementation of Terminal connection using asyncio.

close()
Close AsyncioTerminal connection.

Connection should allow for calling close on closed/not-open connection.

data_received(*data*, *recv_time*)
Await initial prompt of started shell command.

After that - do real forward

name

open()
Open AsyncioTerminal connection & start running it inside asyncio loop.

send(*data*)
Write data into AsyncioTerminal connection.

class moler.io.asyncio.terminal.**PtySubprocessProtocol**(*pty_fd=None*)

Bases: asyncio.protocols.SubprocessProtocol

connection_made(*transport*)
Called when a connection is made.

The argument is the transport representing the pipe connection. To receive data, wait for data_received() calls. When the connection is closed, connection_lost() is called.

data_received(*data*, *recv_time*)

on_process_exited(*return_code*)

on_pty_close(*exc*)

on_pty_open()

pipe_connection_lost(*fd*, *exc*)
Called when a file descriptor associated with the child process is closed.

fd is the int file descriptor that was closed.

pipe_data_received(*fd*, *data*)
Called when the subprocess writes data into stdout/stderr pipe.

fd is int file descriptor. data is bytes object.

process_exited()
Called when subprocess has exited.

send(*data*)

moler.io.asyncio.terminal.**open_terminal**(*dimensions*)
Open pseudo-Terminal and configure it's dimensions

Parameters **dimensions** – terminal dimensions (rows, columns)

Returns (master, slave) file descriptors of Pty

moler.io.asyncio.terminal.**run_command**(*cmd*, *cwd*)

moler.io.asyncio.terminal.**start_reading_pty**(*protocol*, *pty_fd*)
Make asyncio to read file descriptor of Pty

Parameters

- **protocol** – protocol of subprocess speaking via Pty
- **pty_fd** – file descriptor of Pty (dialog with subprocess goes that way)

Returns

moler.io.asyncio.terminal.**start_subprocess_in_terminal**(*protocol_class*, *cmd=None*, *cwd=None*, *env=None*, *dimensions=(100, 300)*)
Start subprocess that will run cmd inside terminal.

Some commands run differently when they detect "I'm running at terminal" (stdin/stdout/stderr are bound to terminal device) They assume human interaction so, for example they display "Password:" prompt.

Parameters

- **protocol_class** –
- **cmd** – command to be run at terminal
- **cwd** – working directory when to start that command
- **env** – environment for command
- **dimensions** – terminal dimensions (rows, columns)

Returns

moler.io.asyncio.terminal.**terminal_io_test**()

Module contents

External-IO connections based on asyncio

The only 3 requirements for these connections is: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

moler.io.raw package

Submodules

moler.io.raw.memory module

External-IO connections based on memory buffer.

The only 3 requirements for these connections are: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

Logging inside ext-IO is mainly focused on connection establishment/close/drop. Data transfer aspects of connection are logged by embedded Moler's connection.

class moler.io.raw.memory.**FifoBuffer**(*moler_connection*, *echo=True*, *name=None*, *logger_name=''*)

Bases: *moler.io.io_connection.IOConnection*

FIFO-in-memory.:

```
inject              |\
          +---------+ \  read
write     +---------+ /
                    |/
```

Usable for unit tests (manually inject what is expected).

inject(*input_bytes*, *delay=0.0*)

Add bytes to FIFO with injection-delay before each data

Parameters

- **input_bytes** – iterable of bytes to inject
- **delay** – delay before each inject

Returns None

inject_response(*input_bytes*, *delay=0.0*)

Injection is activated by nearest write()

Parameters

- **input_bytes** – iterable of bytes to inject
- **delay** – delay before each inject

Returns None

name
Get name of connection

read(*bufsize=None*)
Remove bytes from front of buffer

receive(*bufsize=None*)
Remove bytes from front of buffer

send(*input_bytes*)
What is written to connection comes back on read() only if we simulate echo service of remote end.

write(*input_bytes*)
What is written to connection comes back on read() only if we simulate echo service of remote end.

class moler.io.raw.memory.**ThreadedFifoBuffer**(*moler_connection*, *echo=True*, *name=None*, *logger_name=''*)
Bases: *moler.io.raw.memory.FifoBuffer*

FIFO-in-memory connection inside dedicated thread.

This is external-IO usable for Moler since it has it's own runner (thread) that can work in background and pull data from FIFO-mem connection. Usable for integration tests.

close()
Stop pulling thread.

inject(*input_bytes*, *delay=0.0*)
Add bytes to end of buffer

Parameters

- **input_bytes** – iterable of bytes to inject
- **delay** – delay before each inject

Returns None

open()
Start thread pulling data from FIFO buffer.

pull_data(*pulling_done*)
Pull data from FIFO buffer.

moler.io.raw.sshshell module

External-IO connections based on Paramiko.

The only 3 requirements for these connections are: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

class moler.io.raw.sshshell.**SshShell**(*host*, *port=22*, *username=None*, *login=None*, *password=None*, *receive_buffer_size=262144*, *logger=None*, *existing_client=None*)
Bases: object

Implementation of 'remote shell over Ssh' connection using python Paramiko module

This connection is not intended for one-shot actions like execute_command of paramiko. It's purpose is to provide continuous stream of bytes from remote shell. Moreover, it works with Pty assigned to remote shell to enable interactive dialog like asking for login or password.

close()
Close SshShell connection. Close channel of that connection.

If SshShell was created with "reused ssh transport" then closing will close only ssh channel of remote shell. Ssh transport will be closed after it's last channel is closed.

classmethod from_sshshell(*sshshell*, *logger=None*)
Build new sshshell based on existing one - it will reuse its transport

No need to provide host, port and login credentials - they will be reused. You should use this constructor if you are connecting towards same host/port using same credentials.

Parameters

- **sshshell** – existing connection to reuse it's ssh transport
- **logger** – new logger for new connection

Returns instance of new sshshell connection with reused ssh transport

open()
Open Ssh channel to remote shell.

If SshShell was created with "reused ssh transport" then no new transport is created - just shell channel. (such connection establishment is quicker) Else - before creating channel we create ssh transport and perform full login with provided credentials.

May be used as context manager: with connection.open():

receive(*timeout=30.0*)
Receive data.

Parameters **timeout** (*float*) – max time to await for data, default 30 sec

send(*data*, *timeout=1*)
Send data via SshShell connection.

Parameters

- **data** (*bytes*) – data
- **timeout** (*float*) – max time to spend on sending all data, default 1 sec

class moler.io.raw.sshshell.**ThreadedSshShell**(*moler_connection*, *host*, *port=22*, *username=None*, *login=None*, *password=None*, *receive_buffer_size=262144*, *name=None*, *logger_name=''*, *existing_client=None*)
Bases: *moler.io.io_connection.IOConnection*

SshShell connection feeding Moler's connection inside dedicated thread.

This is external-IO usable for Moler since it has it's own runner (thread) that can work in background and pull data from SshShell connection.

close()
Close SshShell connection. Close channel of that connection & stop pulling thread.

If SshShell was created with "reused ssh transport" then closing will close only ssh channel of remote shell. Ssh transport will be closed after it's last channel is closed.

classmethod from_sshshell(*moler_connection*, *sshshell*, *name=None*, *logger_name=''*)
Build new sshshell based on existing one - it will reuse its transport

No need to provide host, port and login credentials - they will be reused. You should use this constructor if you are connecting towards same host/port using same credentials.

Parameters

- **moler_connection** – moler-connection may not be reused; we need fresh one
- **sshshell** – existing connection to reuse it's ssh transport
- **logger_name** – new logger for new connection

Returns instance of new sshshell connection with reused ssh transport

name
Get name of connection

open()
Open Ssh channel to remote shell & start thread pulling data from it.

If SshShell was created with "reused ssh transport" then no new transport is created - just shell channel. (such connection establishment is quicker) Else - before creating channel we create ssh transport and perform full login with provided credentials.

May be used as context manager: with connection.open():

receive()
Pull data bytes from external-IO:

data = io_connection.receive()

data is intended to forward into Moler's connection:

self.moler_connection.data_received(data)

send(*data*, *timeout=1*)
Send data via SshShell connection.

Parameters

- **data** (*bytes*) – data
- **timeout** (*float*) – max time to spend on sending all data, default 1 sec

moler.io.raw.subprocess module

External-IO connections based on pyhon subprocess module.

class moler.io.raw.subprocess.**Subprocess**
Bases: object

data_received(*data*, *recv_time*)

read_subprocess_output()

receive(*timeout=30*)

send(*data*)

start()

stop()

class moler.io.raw.subprocess.**ThreadedSubprocess**
Bases: *moler.io.raw.subprocess.Subprocess*

close()

open()

pull_data(*pulling_done*)

moler.io.raw.tcp module

External-IO connections based on raw sockets.

The only 3 requirements for these connections are: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

class moler.io.raw.tcp.**Tcp**(*port*, *host='localhost'*, *receive_buffer_size=262144*, *logger=None*)
Bases: object

Implementation of TCP connection using python builtin modules.:

```
socket.send    /|           |\
              / +-----------+ \
             /    host:port    \  TCP server
socket.recv  \                 /
              \ +-----------+ /
               \|           |/
```

close()
Close TCP connection.

Connection should allow for calling close on closed/not-open connection.

open()
Open TCP connection.

Should allow for using as context manager: with connection.open():

receive(*timeout=30*)
Receive data.

Parameters **timeout** (*float*) – time-out, default 30 sec

send(*data*)
Send data via TCP service.

Parameters **data** (*str*) – data

class moler.io.raw.tcp.**ThreadedTcp**(*moler_connection*, *port*, *host='localhost'*, *receive_buffer_size=262144*, *logger=None*)
Bases: *moler.io.raw.tcp.Tcp*

TCP connection feeding Moler's connection inside dedicated thread.

This is external-IO usable for Moler since it has it's own runner (thread) that can work in background and pull data from TCP connection.

close()
Close TCP connection & stop pulling thread.

open()
Open TCP connection & start thread pulling data from it.

pull_data(*pulling_done*)
Pull data from TCP connection.

moler.io.raw.tcpserverpiped module

TCP server based on raw sockets and multiprocessing with services usable for integration tests.

It has backdoor connection over pipe that forms control-service which can: - inject data that should be server's response for client's request - send asynchronous data towards client (without awaiting client's request) - close client connection - return history of server activity (client connected, data received, ...) - shutdown server

class moler.io.raw.tcpserverpiped.**TcpServerPiped**(*host*, *port*, *pipe_in*, *buffer_size=1024*, *delay=0*, *use_stderr_logger=False*)
Bases: multiprocessing.context.Process

check_and_handle_client_socket()
Handle data coming from client connection

check_and_handle_server_socket()
Check if client is knocking :-)

configure_stderr_logger()
Configure logging output on stderr

do_close_connection(***kwargs*)
Handles following pipe message ("close connection", {}) Force server to close connection to client

do_get_history(***kwargs*)
Handles following pipe message ("get history", {}) Retrieve server's history

do_inject_response(***kwargs*)
Handles following pipe message ("inject response", {'req': rpc_startup_msg, 'resp': response_for_rpc_startup_msg}) It defines response ('resp') that should be send by server in reaction to received client data ('req').

do_send_async_msg(***kwargs*)
Handles following pipe message ("send async msg", {'msg': msg_payload}) It injects message to be send by server 'just now' (without awaiting any data from client)

do_set_delay(***kwargs*)
Handles following pipe message ("set delay", {'delay': 2.5}) It defines how much we should delay sending response

do_shutdown(***kwargs*)
Handles following pipe message ("shutdown", {}) Force server to shut down

handle_controlling_action()
Handle actions coming from server-controlling pipe

handle_data(*data*)
Handle data that came from client

handle_incomming_client()
Accept incoming client - make socket connection for it

handle_leaving_client()
Handle client leaving server

interpret_pipe_msg(*msg*)
Interpret message that came from controlling pipe

prepare_server_socket()
Create, configure and start server-listening socket

run()
Run main loop of server process

moler.io.raw.tcpserverpiped.**tcp_server_piped**(*port=19543*, *use_stderr_logger=False*)
TCP server context-manager used for integration tests.

It starts server on localhost/given-port during context-manager start() and performs server-cleanup during context-manager stop()

Parameters

- **port** – where to start server at
- **use_stderr_logger** – configure logging to output into stderr?

Returns pair (server, inter-process-pipe used to control server)

moler.io.raw.terminal module

class moler.io.raw.terminal.**ThreadedTerminal**(*moler_connection*, *cmd='/bin/bash'*, *select_timeout=0.002*, *read_buffer_size=4096*, *first_prompt='[%$#]+'*, *target_prompt='moler_bash#'*, *set_prompt_cmd='export PS1="moler_bash# "\n'*, *dimensions=(100, 300)*, *terminal_delayafterclose=0.2*)
Bases: *moler.io.io_connection.IOConnection*

Works on Unix (like Linux) systems only!

ThreadedTerminal is shell working under Pty

close()
Close ThreadedTerminal connection & stop pulling thread.

open()
Open ThreadedTerminal connection & start thread pulling data from it.

pull_data(*pulling_done*)
Pull data from ThreadedTerminal connection.

send(*data*)
Write data into ThreadedTerminal connection.

Module contents

External-IO connections based on raw libraries like: sockets, subprocess, ...

The only 3 requirements for these connections is: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

class moler.io.raw.**TillDoneThread**(*done_event*, *target=None*, *name=None*, *kwargs=None*)
Bases: threading.Thread

join(*timeout=None*)
Wait until the thread terminates. Set event indicating "I'm done" before awaiting.

Submodules

moler.io.io_connection module

External-IO connections.

The only 3 requirements for these connections is: (1) store Moler's connection inside self.moler_connection attribute (2) forward IO received data into self.moler_connection.data_received(data) (3) provide Moler connection with method to send outgoing data:

self.moler_connection.how2send = self.send

We want all connections of this subpackage to have open()/close() API even that for memory connection that may have no sense but for majority (tcp, udp, ssh, process . . .) it applies, so lets have common API. Moreover, we will map it to context manager API.

send()/receive() is required but we cant force naming here since it's better to keep native/well-known naming of external-IO (send/recv for socket, transport.write/data_received for asyncio, . . .) Specific external-IO implementation must do data-forwarding and send-plugin as stated above. NOTE: send/receive works on bytes since encoding/decoding is responsibility of moler_connection

So, below class is not needed to work as base class. It may, but rather it is generic template how to glue external-IO with Moler's connection.

class moler.io.io_connection.**IOConnection**(*moler_connection*)
Bases: object

External-IO connection.

close()
Close established connection.

data_received(*data*, *recv_time*)
Having been given data bytes from external-IO:

just forward it to Moler's connection:

disable_logging()
Disable logging incoming data. :return: None

enable_logging()
Enable logging incoming data. :return: None

name

notify(*callback*, *when*)
Adds subscriber to list of functions to call :param callback: reference to function to call when connection is open/established :param when: connection state change :return: None

open()
Take 'how to establish connection' info from constructor and open that connection.

Return context manager to allow for: with connection.open() as conn:

receive()
Pull data bytes from external-IO:

data = io_connection.receive()

and then forward it to Moler's connection:

self.moler_connection.data_received(data)

send(*data*)
Send data bytes over external-IO.

Because of plugin done in constructor it will be called by moler_connection when it needs.

subscribe_on_connection_lost(*subscriber*)
Adds subscriber to list of functions to call when connection is closed/disconnected :param subscriber: reference to function to call when connection is closed/disconnected :return: None

subscribe_on_connection_made(*subscriber*)
Adds subscriber to list of functions to call when connection is open/established (also reopen after close) :param subscriber: reference to function to call when connection is open/established :return: None

unsubscribe_on_connection_lost(*subscriber*)
Remove subscriber from list of functions to call when connection is closed/disconnected :param subscriber: reference to function registered by method subscribe_on_connection_lost :return: None

unsubscribe_on_connection_made(*subscriber*)
Remove subscriber from list of functions to call when connection is open/established (also reopen after close) :param subscriber: reference to function registered by method subscribe_on_connection_made :return: None

moler.io.io_exceptions module

External-IO exceptions.

exception moler.io.io_exceptions.**ConnectionBaseError**
Bases: Exception

exception moler.io.io_exceptions.**ConnectionTimeout**
Bases: *moler.io.io_exceptions.ConnectionBaseError*

exception moler.io.io_exceptions.**RemoteEndpointDisconnected**(*err_code=None*)
Bases: *moler.io.io_exceptions.ConnectionBaseError*

exception moler.io.io_exceptions.**RemoteEndpointNotConnected**
Bases: *moler.io.io_exceptions.ConnectionBaseError*

Module contents

External-IO connections.

The only 3 requirements for these connections is: (1) store Moler's connection inside self.moler_connection attribute (2) plugin into Moler's connection the way IO outputs data to external world:

self.moler_connection.how2send = self.send

(3) forward IO received data into self.moler_connection.data_received(data)

moler.parser package

Submodules

moler.parser.table_text module

Last modification: adding support for EPC tables and partially empty tables changes till 23.05.2018 added adjustements for project requirements

Commad TableText is parsing Linux Command to according to headers and columns (adjusted) For initialization it is getting 4 parameters (2 required 2 optional) Required parameters: _header_regexps –> regexp array for finding headers _header_keys –> array of keys assigned for found columns Optional parameters: _skip –> regexp for skipping line _finish –> regexp for finishing parsing (next parse execution will not give any line) Value_splitter –> value for splitting values in line by default 1+ spaces WARNING ==> in current version no special signs or spaces are allowed in Name field (header key regexp)

class moler.parser.table_text.**TableText**(*_header_regexps*, *_header_keys*, *_skip=''*, *_finish=''*, *value_splitter='\s+'*)

Bases: object

build_hdr_groups()

Function for building group of headers. If not enough keys defined name COL_x will be added at the end :return: regexp allowing to read headers group with column names in format (?P<name>regexp) what allows to read assign correct name for matched value

static convert_data_to_type(*data*)

get_value_for_header(*current_header_position*, *values*, *start_value_index*)

Parameters

- **current_header_position** – matched headers positions (get with re.search(regexp, line)) for which search will take place
- **values** – values connected into list
- **start_value_index** – current start from which search will take place

Returns Returns found value for specific header

parse(*data*)

Parameters **data** – accepts one parameter which is line for parsing. Without headers found it will look for it

Returns returns result dictionary according to your header regexps set during initialization returns None when nothing was found or headers when values were found

static split_with_end_position(*values*, *data*)

Parameters

- **values** – values splitted from data
- **data** – data full line with values in raw form

Returns returns list of dictionary where each dict contains 2 keys 'value' and 'end' which is end position of string

Module contents

moler.util package

Submodules

moler.util.cmds_events_doc module

Perform command autotest for selected command(s).

moler.util.cmds_events_doc.**check_cmd_or_event**(*path2cmds*)

moler.util.cmds_events_doc.**check_if_documentation_exists**(*path2cmds*)
Check if documentation exists and has proper structure.

Parameters **path2cmds** (*str*) – relative path to comands directory

Returns True if all checks passed

Return type bool

moler.util.connection_observer module

Utilities related to connection-observers

class moler.util.connection_observer.**exception_stored_if_not_main_thread**(*connection_observer*, *logger=None*)
Bases: object

Context manager storing exception inside connection-observer for non-main threads

moler.util.connection_observer.**inside_main_thread**()

moler.util.connection_observer_life_status module

class moler.util.connection_observer_life_status.**ConnectionObserverLifeStatus**
Bases: object

moler.util.converterhelper module

Units converter

class moler.util.converterhelper.**ConverterHelper**
Bases: object

static get_converter_helper()

to_bytes(*str_bytes*, *binary_multipliers=True*)
Method to convert size with unit to size in bytes :param str_bytes: String with bytes and optional unit :param binary_multipliers: If True then binary multipliers will be used, if False then decimal :return: 3 values: int value in bytes, float value in units (parsed form string), String unit

to_number(*value*, *raise_exception=True*)
Convert number to int or float.

Parameters

- **value** – string with number inside
- **raise_exception** – if True then raise exception if cannot convert to number, If False then return 0.

Returns int or float with value.

to_seconds(*value*, *unit*)
Method to convert number of :param value: numeric value in units :param unit: Unit of time, h for hour :return: number of seconds

to_seconds_str(*str_time*)
Method to convert string time with unit to seconds. :param str_time: String with time and unit. :return: 3 values: float value in seconds, float value in units (parsed form string), String unit

moler.util.devices_SM module

Perform devices SM autotest.

class moler.util.devices_SM.**RemoteConnection**(*moler_connection*, *echo=True*, *name=None*, *logger_name=''*)
Bases: *moler.io.raw.memory.ThreadedFifoBuffer*

remote_inject_response(*input_strings*)
Simulate remote endpoint that sends response. Response is given as strings.

send(*input_bytes*)
What is written to connection comes back on read() only if we simulate echo service of remote end.

set_device(*device*)
Need to get actual state of device when sending cmds response.

write(*input_bytes*)
What is written to connection comes back on read() only if we simulate echo service of remote end.

moler.util.devices_SM.**get_cloned_device**(*src_device*, *new_name*, *new_connection*)

moler.util.devices_SM.**get_device**(*name*, *connection*, *device_output*, *test_file_path*)

moler.util.devices_SM.**get_memory_device_connection**()

moler.util.devices_SM.**iterate_over_device_states**(*device*, *max_time=None*, *tests_per_device=300*, *max_no_of_threads=11*, *rerun=0*, *timeout_multiply=3.0*)
Check all states in device under test. :param device: device :param max_time: maximum time of check. None for infinity. If execution time is greater then max_time then test is

interrupted.

Parameters

- **tests_per_device** – how many tests should be performed in one thread.
- **max_no_of_threads** – maximum number of threads that can be started.
- **rerun** – rerun for goto_state
- **timeout_multiply** – timeout_multiply for goto_state

Returns None

moler.util.loghelper module

Utilities related to logging

moler.util.loghelper.**debug_into_logger**(*logger*, *msg*, *extra=None*, *levels_to_go_up=0*)

moler.util.loghelper.**disabled_logging**(*from_level_and_below=20*)

moler.util.loghelper.**error_into_logger**(*logger*, *msg*, *extra=None*, *levels_to_go_up=0*)

moler.util.loghelper.**find_caller**(*levels_to_go_up=0*)
Find the stack frame of the caller so that we can note the source file name, line number and function name.

Based on findCaller() from logging module but allows to go higher back on stack

Parameters **levels_to_go_up** – 0 - info about 'calling location' of caller of findCaller(); 1 - 'calling -1 location'

Returns

moler.util.loghelper.**info_into_logger**(*logger*, *msg*, *extra=None*, *levels_to_go_up=0*)

moler.util.loghelper.**log_into_logger**(*logger*, *level*, *msg*, *extra=None*, *levels_to_go_up=0*)
Log into specific logger

No support for logging exceptions

Parameters

- **logger** – logger to send log record into
- **level** – logging level
- **msg** – message to be logged
- **levels_to_go_up** – 0 - info about function using log_into_logger(), 1 - caller of that function, ...

Returns None

moler.util.loghelper.**warning_into_logger**(*logger*, *msg*, *extra=None*, *levels_to_go_up=0*)

moler.util.moler_serial_proxy module

class moler.util.moler_serial_proxy.**AtConsoleProxy**(*port*, *verbose=False*, *at_echo=False*)
Bases: object

Class to proxy AT commands console into stdin/stdout

close()
Close underlying serial connection.

open()
Open underlying serial connection.

Return context manager to allow for: with connection.open() as conn:

send(*cmd*)
Send data over underlying serial connection

class moler.util.moler_serial_proxy.**AtToStdout**(*prefix*, *verbose=False*)
Bases: serial.threaded.LineReader

ATserial->stdout

await_response_event(*timeout=4*)
Await till reading thread sets found event

connection_lost(*exc*)
Forget transport

connection_made(*transport*)
Store transport

handle_line(*line*)
Process one line - to be overridden by subclassing

send(*line*)

send_and_await_response(*line*, *response*, *timeout=4*)
Await till data comes

class moler.util.moler_serial_proxy.**IOSerial**(*port*, *baudrate=115200*, *stopbits=1*, *parity='N'*, *timeout=2*, *xonxoff=1*)
Bases: object

Serial-IO connection.

close()
Close established connection.

open()
Take 'how to establish connection' info from constructor and open that connection.

Return context manager to allow for: with connection.open() as conn:

read()
Returns data read from serial connection

send(*cmd*)
Sends data over serial connection

class moler.util.moler_serial_proxy.**IoThreadedSerial**(*port*, *protocol_factory*, *baudrate=115200*, *stopbits=1*, *parity='N'*, *timeout=2*, *xonxoff=1*)
Bases: *moler.util.moler_serial_proxy.IOSerial*

Threaded Serial-IO connection.

close()
Close the serial port and exit reader thread

open()
Take 'how to establish connection' info from constructor and open that connection.

Return context manager to allow for: with connection.open() as conn:

send(*cmd*)
Sends data over serial connection

send_and_await_response(*line*, *response*, *timeout=4*)

moler.util.moler_test module

Utility/common code of library.

class moler.util.moler_test.**MolerTest**
Bases: object

classmethod error(*msg*, *raise_exception=False*, *dump=None*)
Makes an error (fail the test) and (optional) continue the test flow. :param msg: Message to show. :param raise_exception: If True then raise an exception (if not in try except block then test will be

terminated), if False then only show msg and mark error in logs.

Parameters **dump** – If defined then dump object.

Returns None.

classmethod info(*msg*, *dump=None*)
Shows the message :param msg: Message to show. :param dump: If defined then dump object. :return: None.

classmethod raise_background_exceptions(*decorated='function'*, *check_steps_end=False*)
Decorates the function, method or class. :param decorated: Function, method or class to decorate. :param check_steps_end: If True then check if steps_end was called before return the method, if False then do

not check

Returns Decorated callable

classmethod sleep(*seconds*, *quiet=False*)
Add sleep functionality TODO: add support to asyncio when runner ready :param seconds: Time to sleep (in seconds) :param quiet: If True then no info to log about sleeping, if False then sleep info will be logged :return:

classmethod steps_end()
You should call this function at the end of your test code with Moler. :return: None

classmethod stop_python(*force=False*)
Stops current Python. :return: None

classmethod warning(*msg*, *dump=None*)
Shows the message as warning. :param msg: Message to show. :param dump: If defined then dump object. :return: None

Module contents

2.1.2 Submodules

2.1.3 moler.abstract_moler_connection module

2.1.4 moler.asyncio_runner module

Asyncio Runner

class moler.asyncio_runner.**AsyncioEventThreadsafe**(*, *loop=None*)
Bases: asyncio.locks.Event

clear()
Reset the internal flag to false. Subsequently, coroutines calling wait() will block until set() is called to set the internal flag to true again.

set()
Set the internal flag to true. All coroutines waiting for it to become true are awakened. Coroutine that call wait() once the flag is true will not block at all.

class moler.asyncio_runner.**AsyncioInThreadRunner**
Bases: *moler.asyncio_runner.AsyncioRunner*

shutdown()
Cleanup used resources.

submit(*connection_observer*)
Submit connection observer to background execution. Returns Future that could be used to await for connection_observer done.

wait_for(*connection_observer*, *connection_observer_future*, *timeout=None*)
Await for connection_observer running in background or timeout.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().
- **timeout** – Max time (in float seconds) to await before give up. None - use connection_observer.timeout

Returns

wait_for_iterator(*connection_observer*, *connection_observer_future*)
Version of wait_for() intended to be used by Python3 to implement iterable/awaitable object.

Note: we don't have timeout parameter here. If you want to await with timeout please do use timeout machinery of selected parallelism.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().

Returns iterator

class moler.asyncio_runner.**AsyncioLoopThread**(*name='Asyncio'*)
Bases: *moler.io.raw.TillDoneThread*

join(*timeout=None*)
Closing asyncio loop must be done from MainThread to allow for removing signal handlers

run_async_coroutine(*coroutine_to_run*, *timeout*)
Start coroutine in dedicated thread and await its result with timeout

start()
We wan't this method to not return before it ensures that thread and it's enclosed loop are really running.

start_async_coroutine(*coroutine_to_run*)
Start coroutine in dedicated thread, don't await its result

class moler.asyncio_runner.**AsyncioRunner**(*logger_name='moler.runner.asyncio'*)
Bases: *moler.runner.ConnectionObserverRunner*

feed(*connection_observer*, *subscribed_data_receiver*, *observer_lock*)
Feeds connection_observer by transferring data from connection and passing it to connection_observer. Should be called from background-processing of connection observer.

last_runner_id = 0

runner_lock = <unlocked _thread.lock object>

shutdown()
Cleanup used resources.

submit(*connection_observer*)
Submit connection observer to background execution. Returns Future that could be used to await for connection_observer done.

timeout_change(*timedelta*)
Call this method to notify runner that timeout has been changed in observer :param timedelta: delta timeout in float seconds :return: None

wait_for(*connection_observer*, *connection_observer_future*, *timeout=None*)
Await for connection_observer running in background or timeout.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().
- **timeout** – Max time (in float seconds) to await before give up. If None then taken from connection_observer

Returns

wait_for_iterator(*connection_observer*, *connection_observer_future*)
Version of wait_for() intended to be used by Python3 to implement awaitable object.

Note: we don't have timeout parameter here. If you want to await with timeout please do use asyncio machinery. For ex.: await asyncio.wait_for(connection_observer, timeout=10)

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().

Returns iterator

class moler.asyncio_runner.**LoudEventLoop**(**args*)
Bases: asyncio.unix_events._UnixSelectorEventLoop

stop()
Stop running the event loop.

Every callback already scheduled will still run. This simply informs run_forever to stop looping after a complete iteration.

class moler.asyncio_runner.**LoudEventLoopPolicy**
Bases: asyncio.unix_events._UnixDefaultEventLoopPolicy

moler.asyncio_runner.**cancel_remaining_feeders**(*loop*, *logger_name='moler.runner.asyncio'*, *in_shutdown=False*)

moler.asyncio_runner.**check_system_resources_limit**(*connection_observer*, *observer_lock*, *logger*)

moler.asyncio_runner.**cleanup_remaining_tasks**(*loop*, *logger*)

moler.asyncio_runner.**cleanup_selected_tasks**(*tasks2cancel*, *loop*, *logger*)

moler.asyncio_runner.**feeder_callback**(*future*)
Used to recognize asyncio task as AsyncioRunner feeder

moler.asyncio_runner.**get_asyncio_loop_thread**()

moler.asyncio_runner.**handle_cancelled_feeder**(*connection_observer*, *observer_lock*, *subscribed_data_receiver*, *logger*, *future*)

moler.asyncio_runner.**is_feeder**(*task*)
We recognize asyncio task to be feeder if it has feeder_callback attached

moler.asyncio_runner.**system_resources_usage**()

moler.asyncio_runner.**system_resources_usage_msg**(*curr_fds_open*, *curr_threads_nb*)

moler.asyncio_runner.**thread_secure_get_event_loop**(*logger_name='moler.runner.asyncio'*)
Need securing since asyncio.get_event_loop() when called from new thread may raise sthg like: RuntimeError: There is no current event loop in thread 'Thread-3' It is so since MainThread has preinstalled loop but other threads must setup own loop by themselves.

Returns loop of current thread + info if it was newly created

2.1.5 moler.command module

Command is a type of ConnectionObserver. Additionally: - it starts by sending command string over connection - starting CMD(*) - it focuses on parsing the output caused by that CMD - it stores string starting that CMD inside .command_string attribute

(*) we use naming CMD to differentiate from Command class naming: - CMD - command started on some device like 'ls -l' that has its own output - Command - Python code automating its startup/parsing/completion

class moler.command.**Command**(*connection=None*, *runner=None*)
Bases: *moler.connection_observer.ConnectionObserver*

get_long_desc()

get_short_desc()

is_command()

Returns True if instance of ConnectionObserver is a command. False if not a command.

send_command()
Sends command string over connection.

Returns None

2.1.6 moler.command_scheduler module

Scheduler for commands and events.

class moler.command_scheduler.**CommandScheduler**
Bases: object

Scheduler for commands and events.

static dequeue_running_on_connection(*connection_observer*)
Remove command from queue and/or current executed on connection.

Parameters **connection_observer** – Command object to remove from connection

Returns None

static enqueue_starting_on_connection(*connection_observer*)
Wait for free slot and runs command when no other command is in run mode.

If connection_observer is not a command then runs immediately. :param connection_observer: Object of ConnectionObserver to run. Maybe a command or an observer. :return: None

static is_waiting_for_execution(*connection_observer*)
Check if connection_observer waits in queue before passed to runner.

Parameters **connection_observer** – ConnectionObserver object.

Returns True if connection_observer waits in queue and False if it does not wait.

2.1.7 moler.connection module

2.1.8 moler.connection_factory module

One of Moler's goals is to be IO-agnostic. So it can be used under twisted, asyncio, curio any any other IO system.

Moler's connection is very thin layer binding Moler's ConnectionObserver with external IO system. Connection responsibilities: - have a means for sending outgoing data via external IO - have a means for receiving incoming data from external IO - perform data encoding/decoding to let external IO use pure bytes - have a means allowing multiple observers to get it's received data (data dispatching)

class moler.connection_factory.**ConnectionFactory**
Bases: object

ConnectionFactory creates plugin-system: external code can register "construction recipe" that will be used to create specific connection.

"Construction recipe" means: class to be used or any other callable that can produce instance of connection.

Specific means type/variant pair. Type is: memory, tcp, udp, ssh, ... Variant is: threaded, asyncio, twisted, ...

Connection means here: external-IO-connection + moler-connection. Another words - fully operable connection doing IO and data dispatching, ready to be used by ConnectionObserver.

ConnectionFactory responsibilities: - register "recipe" how to build given type/variant of connection - return connection instance created via utilizing registered "recipe"

classmethod available_variants(*io_type*)
Return variants available for given io_type

Parameters **io_type** – 'tcp', 'memory', 'ssh', ...

Returns list of variants, ex. ['threaded', 'twisted']

classmethod get_connection(*io_type*, *variant*, ***constructor_kwargs*)
Return connection instance of given io_type/variant

Parameters

- **io_type** – 'tcp', 'memory', 'ssh', ...
- **variant** – implementation variant, ex. 'threaded', 'twisted', 'asyncio', ...
- **constructor_kwargs** – arguments specific for given io_type

Returns requested connection

classmethod register_construction(*io_type*, *variant*, *constructor*)
Register constructor that will return "connection construction recipe"

Parameters

- **io_type** – 'tcp', 'memory', 'ssh', ...

- **variant** – implementation variant, ex. 'threaded', 'twisted', 'asyncio', ...
- **constructor** – callable building connection object

Returns None

moler.connection_factory.**get_connection**(*name=None*, *io_type=None*, *variant=None*, ***constructor_kwargs*)

Return connection instance of given io_type/variant

Parameters

- **name** – name of connection defined in configuration
- **io_type** – 'tcp', 'memory', 'ssh', ...
- **variant** – implementation variant, ex. 'threaded', 'twisted', 'asyncio', ...
- **constructor_kwargs** – arguments specific for given io_type

Returns requested connection

You may provide either 'name' or 'io_type' but not both. If you provide 'name' then it is searched inside configuration to find io_type and constructor_kwargs assigned to that name.

If variant is not given then it is taken from configuration.

2.1.9 moler.connection_observer module

class moler.connection_observer.**ConnectionObserver**(*connection=None*, *runner=None*)

Bases: object

await_done(*timeout=None*)

Await completion of connection-observer.

CAUTION: if you call it from asynchronous code (async def) you may block events loop for long time. You should rather await it via: result = await connection_observer or (to have timeout) result = await asyncio.wait_for(connection_observer, timeout=10) or you may delegate blocking call execution to separate thread, see: https://pymotw.com/3/asyncio/executors.html

Parameters **timeout** –

Returns observer result

cancel()

Cancel execution of connection-observer.

cancelled()

Return True if the connection-observer has been cancelled.

connection_closed_handler()

Called by Moler (ThreadedMolerConnection) when connection is closed.

Returns None

data_received(*data*, *recv_time*)

Entry point where feeders pass data read from connection Here we perform data parsing to conclude in result setting.

Parameters

- **data** – List of strings sent by device.
- **recv_time** – time stamp with the moment when the data was read from connection. Time is given as datetime.datetime instance.

Returns None.

done()
Return True if the connection-observer is already done.

extend_timeout(*timedelta*)

get_logger_name()

get_long_desc()

get_short_desc()

static get_unraised_exceptions(*remove=True*)

is_command()

Returns True if instance of ConnectionObserver is a command. False if not a command.

observer_name = 'connection_observer'

on_inactivity()
Callback called when no data is received on connection within self.life_status.inactivity_timeout seconds

Returns None

on_timeout()
Callback called when observer times out

result()
Retrieve final result of connection-observer

running()
Return True if the connection-observer is currently executing.

set_end_of_life()
Set end of life of object. Dedicated for runners only!

Returns None

set_exception(*exception*)
Should be used to indicate some failure during observation.

Parameters **exception** – Exception to set

Returns None

set_result(*result*)
Should be used to set final result

start(*timeout=None*, **args*, ***kwargs*)
Start background execution of connection-observer.

start_time

terminating_timeout

timeout

2.1.10 moler.event module

class moler.event.**Event**(*connection=None*, *till_occurs_times=-1*, *runner=None*)
Bases: *moler.connection_observer.ConnectionObserver*

add_event_occurred_callback(*callback*, *callback_params=None*)

disable_log_occurrence()
Disables to log every occurrence of the event.

Returns None

enable_log_occurrence()
Enables to log every occurrence of the event.

Returns None

event_occurred(*event_data*)
Sets event_data as new item of occurrence ret. :param event_data: data to set as value of occurrence. :return: None

get_last_occurrence()
Returns ret value from last occurrence.

Returns ret value form last occurrence or None if there is no occurrence.

get_long_desc()
Returns string with description of event.

Returns String with description.

get_short_desc()
Returns string with description of event.

Returns String with description.

notify()
Notifies (call callback).

Returns None

pause()
Pauses the event. Do not process till resume.

Returns None.

remove_event_occurred_callback()
Removes callback from the event.

Returns None

resume()
Resumes processing output from connection by the event.

Returns None.

start(*timeout=None*, **args*, ***kwargs*)
Start background execution of command.

2.1.11 moler.event_awaiter module

Event awaiter

class moler.event_awaiter.**EventAwaiter**
Bases: object

static cancel_all_events(*events*)

Parameters **events** – list of events to cancel

Returns None

static separate_done_events(*events*)

Parameters **events** – list of events to check and separate

Returns tuple. 0th element is list of done events, 1st element is list of non done events

static wait_for_all(*timeout*, *events*, *interval=0.001*)

Wait for all events are done or timeout occurs :param timeout: time in seconds :param events: list of events to check :param interval: interval in seconds between checking events :return: True if all events are done, False otherwise

static wait_for_any(*timeout*, *events*, *interval=0.001*)

Parameters

- **timeout** – time in seconds
- **events** – list of events to check
- **interval** – interval in seconds between checking events

Returns True if any event is done, False otherwise

2.1.12 moler.exceptions module

exception moler.exceptions.**CancelledError**
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**CommandFailure**(*command*, *message*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**CommandTimeout**(*connection_observer*, *timeout*, *kind='run'*, *passed_time=''*)
Bases: *moler.exceptions.ConnectionObserverTimeout*

exception moler.exceptions.**CommandWrongState**(*command*, *expected_state*, *current_state*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**ConnectionObserverNotStarted**(*connection_observer*)
Bases: *moler.exceptions.InvalidStateError*

exception moler.exceptions.**ConnectionObserverTimeout**(*connection_observer*, *timeout*, *kind='run'*, *passed_time=''*)
Bases: *moler.exceptions.MolerTimeout*

exception moler.exceptions.**DeviceChangeStateFailure**(*device*, *exception*, *device_name=None*)
Bases: *moler.exceptions.DeviceFailure*

exception moler.exceptions.**DeviceFailure**(*device*, *message*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**EventWrongState**(*event*, *expected_state*, *current_state*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**ExecutionException**(*msg*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**InvalidStateError**
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**MolerException**
Bases: Exception

exception moler.exceptions.**MolerTimeout**(*timeout*, *kind='run'*, *passed_time=0*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**NoCommandStringProvided**(*command*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**NoConnectionProvided**(*connection_observer*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**NoDetectPatternProvided**(*command*)
Bases: *moler.exceptions.MolerException*

exception moler.exceptions.**NoResultSinceCancelCalled**(*connection_observer*)
Bases: *moler.exceptions.CancelledError*

exception moler.exceptions.**ParsingDone**
Bases: *moler.exceptions.MolerException*

Indicate that given part of output has been fully parsed and requires no further processing.

exception moler.exceptions.**ResultAlreadySet**(*connection_observer*)
Bases: *moler.exceptions.InvalidStateError*

exception moler.exceptions.**ResultNotAvailableYet**(*connection_observer*)
Bases: *moler.exceptions.InvalidStateError*

exception moler.exceptions.**WrongUsage**
Bases: *moler.exceptions.MolerException*

Wrong usage of library

2.1.13 moler.helpers module

Utility/common code of library.

class moler.helpers.**ClassProperty**
Bases: property

class moler.helpers.**ForwardingHandler**(*target_logger_name*)
Bases: logging.Handler

Take log record and pass it to target_logger

emit(*record*)
Emit a record.

Output the record to the target_logger, catering for rollover as described in doRollover().

moler.helpers.**all_chars_to_hex**(*source*)
Converts input string into hex for all chars. :param source: input string. :return: output string witch exchanged chars.

moler.helpers.**call_base_class_method_with_same_name**(*obj*)
Run base class method.

Parameters **obj** – class object which methods will be decorated.

Returns class object with decorated methods

moler.helpers.**camel_case_to_lower_case_underscore**(*string*)
Split string by upper case letters. F.e. useful to convert camel case strings to underscore separated ones. @return words (list)

moler.helpers.**compare_objects**(*first_object*, *second_object*, *ignore_order=False*, *report_repetition=False*, *significant_digits=None*, *exclude_paths=None*, *exclude_types=None*, *verbose_level=2*)
Return difference between two objects. :param first_object: first object to compare :param second_object: second object to compare :param ignore_order: ignore difference in order :param report_repetition: report when is repetition :param significant_digits: use to properly compare numbers(float arithmetic error) :param exclude_paths: path which be excluded from comparison :param exclude_types: types which be excluded from comparison :param verbose_level: higher verbose level shows you more details - default 0. :return: difference between two objects

moler.helpers.**convert_to_int**(*obj*)
Convert element of object structure to int if it's possible. :param obj: object to convert

moler.helpers.**convert_to_number**(*value*)
Convert value to Python number type. :param value: value to convert :return: converted value if possible, otherwise original

moler.helpers.**copy_dict**(*src*, *deep_copy=False*)
Copies dict, if None then returns empty dict :param src: List to copy :param deep_copy: if False then shallow copy, if True then deep copy :return: Copied dict

moler.helpers.**copy_list**(*src*, *deep_copy=False*)
Copies list, if None then returns empty list :param src: List to copy :param deep_copy: if False then shallow copy, if True then deep copy :return: Copied list

moler.helpers.**create_object_from_name**(*full_class_name*, *constructor_params*)

moler.helpers.**instance_id**(*instance*)
Return id of instance in hex form. Helps in logs/debugs/development troubleshooting.

moler.helpers.**is_digit**(*value*)
Check that value is digit. :param value: value to check :return: True if value is digit, otherwise False

moler.helpers.**mark_to_call_base_class_method_with_same_name**(*func*)
Mark method which base class method with same name will be call. :param func: function to mark. :return: marked function

moler.helpers.**non_printable_chars_to_hex**(*source*)
Converts input string into hex for all non printable chars, printable chars remain unchanged. :param source: input string. :return: output string witch exchanged chars.

moler.helpers.**regexp_without_anchors**(*regexp*)
Remove anchors from beginning (^) and ending ($) of the regexp. :param regexp: compiled regexp :return: compiled regexp without anchors

moler.helpers.**remove_all_known_special_chars**(*line*)

Parameters **line** – line from terminal

Returns line without all known special chars

moler.helpers.**remove_cursor_visibility_codes**(*multiline*)

Parameters **multiline** – string from terminal holding single or multiple lines

Returns line(s) without terminal escape codes related to cursor visibility

moler.helpers.**remove_escape_codes**(*line*)

Parameters **line** – line from terminal

Returns line without terminal escape codes

moler.helpers.**remove_fill_spaces_right_codes**(*multiline*)

Parameters **multiline** – string from terminal holding single or multiple lines

Returns line(s) without spaces added till right VT-screen margin

moler.helpers.**remove_overwritten_left_write**(*multiline*)

Parameters **multiline** – string from terminal holding single or multiple lines

Returns line without spaces added till right VT-screen margin

moler.helpers.**remove_terminal_last_cmd_status**(*line*)

Parameters **line** – line from terminal

Returns line without terminal last cmd status

moler.helpers.**remove_text_formatting_codes**(*multiline*)

Parameters **multiline** – string from terminal holding single or multiple lines

Returns line(s) without terminal escape codes related to text formatting

moler.helpers.**remove_window_title_codes**(*multiline*)

Parameters **multiline** – string from terminal holding single or multiple lines

Returns line(s) without terminal escape codes setting console window/icon title

moler.helpers.**update_dict**(*target_dict*, *expand_dict*)

2.1.14 moler.instance_loader module

moler.instance_loader.**create_class_instance**(*class_object*, *constructor_params*)

Factory method that creates class instance object according to its definition given in parameters.

Parameters

- **class_object** – class object to be instantiated
- **constructor_params** – to be passed into instance constructor

Returns instance of requested class

moler.instance_loader.**create_instance_from_class_fullname**(*class_fullname*, *constructor_parameters*)

Factory method that creates class instance object according to its definition given in parameters.

Parameters

- **class_fullname** – full name of class in dotted notation like 'package1.module1.ClassName1'
- **constructor_parameters** – to be passed into instance constructor

Returns instance of requested class

moler.instance_loader.**load_class_from_class_fullname**(*class_fullname*)

Factory method that loads class object according to its fullname.

Parameters **class_fullname** – full name of class in dotted notation like 'package1.module1.ClassName1'

Returns requested class object

2.1.15 moler.observable_connection module

One of Moler's goals is to be IO-agnostic. So it can be used under twisted, asyncio, curio any any other IO system.

Moler's connection is very thin layer binding Moler's ConnectionObserver with external IO system. Connection responsibilities: - have a means for sending outgoing data via external IO - have a means for receiving incoming data from external IO - perform data encoding/decoding to let external IO use pure bytes - have a means allowing multiple observers to get it's received data (data dispatching)

class moler.observable_connection.**ObservableConnection**(**args*, ***kwargs*)
Bases: *moler.threaded_moler_connection.ThreadedMolerConnection*

2.1.16 moler.observer_thread_wrapper module

Wrapper for observer registered in ThreadedMolerConnection (old name: ObservableConnection).

class moler.observer_thread_wrapper.**ObserverThreadWrapper**(*observer*, *observer_self*, *logger*)
Bases: object

Wrapper for observer registered in ThreadedMolerConnection (old name: ObservableConnection).

feed(*data*, *recv_time*)
Put data here.

Parameters

- **data** – data to put.
- **recv_time** – time when data is received/read from connection.

Returns None

request_stop()
Call if you want to stop feed observer.

Returns None

class moler.observer_thread_wrapper.**ObserverThreadWrapperForConnectionObserver**(*observer*, *observer_self*, *logger*)
Bases: *moler.observer_thread_wrapper.ObserverThreadWrapper*

2.1.17 moler.publisher module

Moler implementation of Publisher-Subscriber Design Pattern.

Main characteristic: - allow Subscribers to be garbage collected while still subscribed inside Publisher

This is required since: - both parties may exist in different threads - both parties may keep references to themselves creating reference cycles

class moler.publisher.**Publisher**
Bases: object

Allows objects to subscribe for notification about data.

Subscription is made by registering function to be called with this data (may be object's method). Function should have signature like:

def subscriber(data): # handle that data

handle_subscriber_exception(*subscriber_owner*, *subscriber_function*, *raised_exception*)
Handle exception raised by subscriber during publishing.

Parameters

- **subscriber_owner** – instance of class whose method was subscribed (or None)
- **subscriber_function** – subscribed class method or raw function
- **raised_exception** – exception raised by subscriber during publishing

Returns None

notify_subscribers(**args*, ***kwargs*)
Notify all subscribers passing them notification parameters.

subscribe(*subscriber*)
Subscribe for 'data notification'.

Parameters **subscriber** – function to be called to notify about data.

unsubscribe(*subscriber*)
Unsubscribe from 'data notification'.

Parameters **subscriber** – function that was previously subscribed

2.1.18 moler.runner module

Runner abstraction goal is to hide concurrency machinery used to make it exchangeable (threads, asyncio, twisted, curio)

class moler.runner.**CancellableFuture**(*future*, *observer_lock*, *stop_running*, *is_done*, *stop_timeout=0.5*)
Bases: object

cancel(*no_wait=False*)
Cancel embedded future :param no_wait: if True - just set self._stop_running event to let thread exit loop :return:

class moler.runner.**ConnectionObserverRunner**
Bases: object

feed(*connection_observer*)
Feeds connection_observer with data to let it become done. This is a place where runner is a glue between words of connection and connection-observer. Should be called from background-processing of connection observer.

is_in_shutdown()
Call this method to check if runner is in shutdown mode. :return: Is in shutdown

shutdown()
Cleanup used resources.

submit(*connection_observer*)
Submit connection observer to background execution. Returns Future that could be used to await for connection_observer done.

timeout_change(*timedelta*)
Call this method to notify runner that timeout has been changed in observer :param timedelta: delta timeout in float seconds :return: None

wait_for(*connection_observer*, *connection_observer_future*, *timeout=10.0*)
Await for connection_observer running in background or timeout.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().
- **timeout** – Max time (in float seconds) you want to await before you give up.

Returns

wait_for_iterator(*connection_observer*, *connection_observer_future*)
Version of wait_for() intended to be used by Python3 to implement iterable/awaitable object.

Note: we don't have timeout parameter here. If you want to await with timeout please do use timeout machinery of selected parallelism.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().

Returns iterator

class moler.runner.**ThreadPoolExecutorRunner**(*executor=None*)
Bases: *moler.runner.ConnectionObserverRunner*

feed(*connection_observer*, *subscribed_data_receiver*, *stop_feeding*, *feed_done*, *observer_lock*)
Feeds connection_observer by transferring data from connection and passing it to connection_observer. Should be called from background-processing of connection observer.

is_in_shutdown()
Call this method to check if runner is in shutdown mode. :return: Is in shutdown

shutdown()
Cleanup used resources.

submit(*connection_observer*)
Submit connection observer to background execution. Returns Future that could be used to await for connection_observer done.

timeout_change(*timedelta*)
Call this method to notify runner that timeout has been changed in observer :param timedelta: delta timeout in float seconds :return: None

wait_for(*connection_observer*, *connection_observer_future*, *timeout=None*)
Await for connection_observer running in background or timeout.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().
- **timeout** – Max time (in float seconds) you want to await before you give up. If None then taken from connection_observer

Returns

wait_for_iterator(*connection_observer*, *connection_observer_future*)
Version of wait_for() intended to be used by Python3 to implement iterable/awaitable object.

Note: we don't have timeout parameter here. If you want to await with timeout please do use timeout machinery of selected parallelism.

Parameters

- **connection_observer** – The one we are awaiting for.
- **connection_observer_future** – Future of connection-observer returned from submit().

Returns iterator

moler.runner.**await_future_or_eol**(*connection_observer*, *remain_time*, *start_time*, *timeout*, *logger*)

moler.runner.**his_remaining_time**(*prefix*, *timeout*, *from_start_time*)
Calculate remaining time of "he" object assuming that "he" has .life_status.start_time attribute

Parameters

- **prefix** – string to be used inside 'remaining time description'
- **timeout** – max lifetime of object
- **from_start_time** – start of lifetime for the object

Returns remaining time as float and related description message

moler.runner.**result_for_runners**(*connection_observer*)
When runner takes result from connection-observer it should not modify ConnectionObserver._not_raised_exceptions

Parameters **connection_observer** – observer to get result from

Returns result or raised exception

moler.runner.**time_out_observer**(*connection_observer*, *timeout*, *passed_time*, *runner_logger*, *kind='background_run'*)
Set connection_observer status to timed-out

2.1.19 moler.runner_factory module

Runner abstraction goal is to hide concurrency machinery used to make it exchangeable (threads, asyncio, twisted, curio)

class moler.runner_factory.**RunnerFactory**
Bases: object

RunnerFactory creates plugin-system: external code can register "construction recipe" that will be used to create specific runner.

"Construction recipe" means: class to be used or any other callable that can produce instance of runner.

Specific means runner variant like: threaded, asyncio, twisted, ...

ConnectionFactory responsibilities: - register "recipe" how to build given variant of runner - return runner instance created via utilizing registered "recipe"

classmethod available_variants()
Return available variants of runners

Returns list of variants, ex. ['threaded', 'twisted']

classmethod get_runner(*variant*, *reuse_last=True*, ***constructor_kwargs*)
Return runner instance of given variant

Parameters

- **variant** – implementation variant, ex. 'threaded', 'twisted', 'asyncio', ...
- **reuse_last** – should we return cached last runner of given variant
- **constructor_kwargs** – arguments specific for given variant

Returns requested runner

classmethod register_construction(*variant*, *constructor*)
Register constructor that will return "runner construction recipe"

Parameters

- **variant** – implementation variant, ex. 'threaded', 'twisted', 'asyncio', ...
- **constructor** – callable building runner object

Returns None

moler.runner_factory.**get_runner**(*variant=None*, *reuse_last=True*, ***constructor_kwargs*)
Return runner instance of given variant

Parameters

- **variant** – implementation variant, ex. 'threaded', 'twisted', 'asyncio', ...
- **reuse_last** – should we return cached last runner of given variant
- **constructor_kwargs** – arguments specific for given variant

Returns requested runner

If variant is not given then it is taken from configuration.

2.1.20 moler.scheduler module

class moler.scheduler.**DecoratedCallable**(*callback*, *cancel_on_exception*)
Bases: object

call(***kwargs*)

class moler.scheduler.**Job**(*job*)
Bases: object

cancel()
Method to stop the job :return: None

start()
Method to start the job. :return: None

class moler.scheduler.**MolerAsyncioScheduler**(*gconfig={}*, ***options*)
Bases: apscheduler.schedulers.asyncio.AsyncIOScheduler

forwarding_handler = None

class moler.scheduler.**MolerThreadScheduler**(*gconfig={}*, ***options*)
Bases: apscheduler.schedulers.background.BackgroundScheduler

forwarding_handler = None

class moler.scheduler.**Scheduler**(*scheduler_type=None*)
Bases: object

static change_kind(*scheduler_type=None*)
Static method to change type of scheduler :param scheduler_type: type of new scheduler. Allowed thread (default) or asyncio. If None then default multi

threading model will be used.

Returns None. If scheduler_type is not supported then it raises object of type moler.exceptions.WrongUsage

static get_job(*callback*, *interval*, *callback_params=None*, *cancel_on_exception=False*, *misfire_grace_time=0*)
Static method to create job. :param callback: Reference to callable object (i.e. function, method) :param interval: time in float seconds when fun is called. If time of one execution is longer than interval then

some callbacks are missed. For example: interval is 2s and time of execution is 3s then callback will be called when job ios created after 2s, after 4s will not be executed because still the first excection is running, then after 6s of is called.

Parameters

- **callback_params** – dict of params of fun
- **cancel_on_exception** – set True if you want to break next execution of this callback if previous raises an exception
- **misfire_grace_time** (*int*) – seconds after the designated runtime that the job is still allowed to be run

Returns Instance of Job.

2.1.21 moler.threaded_moler_connection module

One of Moler's goals is to be IO-agnostic. So it can be used under twisted, asyncio, curio any any other IO system.

Moler's connection is very thin layer binding Moler's ConnectionObserver with external IO system. Connection responsibilities: - have a means for sending outgoing data via external IO - have a means for receiving incoming data from external IO - perform data encoding/decoding to let external IO use pure bytes - have a means allowing multiple observers to get it's received data (data dispatching)

class moler.threaded_moler_connection.**ThreadedMolerConnection**(*how2send=None*, *encoder=<function identity_transformation>*, *decoder=<function identity_transformation>*, *name=None*, *newline='n'*, *logger_name=''*)
Bases: moler.abstract_moler_connection.AbstractMolerConnection

Allows objects to subscribe for notification about connection's data-received. Subscription is made by registering function to be called with this data (may be object's method). Function should have signature like:

def observer(data): # handle that data

data_received(*data*, *recv_time*)
Incoming-IO API: external-IO should call this method when data is received

notify_observers(*data*, *recv_time*)
Notify all subscribed observers about data received on connection. :param data: data to send to all registered subscribers. :param recv_time: time of data really read form connection. :return None

shutdown()
Closes connection with notifying all observers about closing. :return: None

subscribe(*observer*, *connection_closed_handler*)
Subscribe for 'data-received notification'

Parameters

- **observer** – function to be called to notify when data received.
- **connection_closed_handler** – callable to be called when connection is closed.

unsubscribe(*observer*, *connection_closed_handler*)
Unsubscribe from 'data-received notification' :param observer: function that was previously subscribed :param connection_closed_handler: callable to be called when connection is closed.

2.1.22 Module contents

CHAPTER 3

Code Examples

This section of the documentation contains long-form code examples. These are intended as references for developers that would like to get an understanding of how Moler fits in with various Python I/O frameworks.

3.1 Layer 1

3.1.1 AsyclO - network down detector

```
# -*- coding: utf-8 -*-
"""
asyncio.network_down_detector.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using asyncio. Requires Python 3.6+.

This example demonstrates basic concept of connection observer - entity
that is fully responsible for:
- observing data coming from connection till it catches what it is waiting for
- parsing that data to have "caught event" stored in expected form
- storing that result internally for later retrieval

Please note that this example is LAYER-1 usage which means:
- observer can't run by its own, must be fed with data (passive observer)
- observer can't be awaited, must be queried for status before asking for data
Another words - low level manual combining of all the pieces.
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import asyncio
```

(continues on next page)

(continued from previous page)

```
import logging
import sys
import os
import time

from moler.threaded_moler_connection import ThreadedMolerConnection

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow␣
↪finding modules in examples/

from network_toggle_observers import NetworkDownDetector

# ====================== Moler's connection-observer usage ======================


async def ping_observing_task(address):
    logger = logging.getLogger('moler.user.app-code')

    # Lowest layer of Moler's usage (you manually glue all elements):
    # 1. create observer
    net_down_detector = NetworkDownDetector('10.0.2.15')
    # 2. ThreadedMolerConnection is a proxy-glue between observer (speaks str)
    #                                    and asyncio-connection (speaks bytes)
    moler_conn = ThreadedMolerConnection(decoder=lambda data: data.decode("utf-8"))
    # 3a. glue from proxy to observer
    moler_conn.subscribe(net_down_detector.data_received)

    logger.debug('waiting for data to observe')
    async for connection_data in tcp_connection(address):
        # 3b. glue to proxy from external-IO (asyncio tcp client connection)
        #   (client code has to pass it's received data into Moler's connection)
        moler_conn.data_received(connection_data)
        # 4. Moler's client code must manually check status of observer ...
        if net_down_detector.done():
            # 5. ... to know when it can ask for result
            net_down_time = net_down_detector.result()
            timestamp = time.strftime("%H:%M:%S", time.localtime(net_down_time))
            logger.debug('Network is down from {}'.format(timestamp))
            break


# ============================================================================
async def tcp_connection(address):
    """Async generator reading from tcp network transport layer"""
    logger = logging.getLogger('asyncio.tcp-connection')
    logger.debug('... connecting to tcp://{}:{}'.format(*address))
    reader, writer = await asyncio.open_connection(*address)
    try:
        while True:
            data = await reader.read(128)
            if data:
                logger.debug('<<< {!r}'.format(data))
                yield data
            else:
                break
    finally:
        logger.debug('... closing')
```

(continues on next page)

(continued from previous page)

```
        writer.close()


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers

    logging.basicConfig(
        level=logging.DEBUG,
        format='%(asctime)s |%(name)40s |%(message)s',
        datefmt='%H:%M:%S',
        stream=sys.stderr,
    )
    event_loop = asyncio.get_event_loop()
    server_address = ('127.0.0.1', 5678)
    servers = start_ping_servers([(server_address, '10.0.2.15')])
    try:
        event_loop.run_until_complete(ping_observing_task(server_address))
    finally:
        stop_ping_servers(servers)
        event_loop.close()

'''
LOG OUTPUT

16:56:30 |                   asyncio |Using selector: SelectSelector
16:56:30 |  asyncio.ping.tcp-server |Ping Sim started at tcp://127.0.0.1:5678
16:56:30 |  asyncio.ping.tcp-server |WARNING - I'll be tired too much just after␣
↪first client!
16:56:30 |      moler.user.app-code |waiting for data to observe
16:56:30 |   asyncio.tcp-connection |... connecting to tcp://127.0.0.1:5678
16:56:30 |  asyncio.ping.tcp-server |connection accepted - client at tcp://127.0.0.
↪1:56556
16:56:30 |   asyncio.tcp-connection |<<< b'\n'
16:56:31 |   asyncio.tcp-connection |<<< b'greg@debian:~$ ping 10.0.2.15\n'
16:56:32 |   asyncio.tcp-connection |<<< b'PING 10.0.2.15 (10.0.2.15) 56(84) bytes of␣
↪data.\n'
16:56:33 |   asyncio.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=1 ttl=64␣
↪time=0.080 ms\n'
16:56:34 |   asyncio.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=2 ttl=64␣
↪time=0.037 ms\n'
16:56:35 |   asyncio.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=3 ttl=64␣
↪time=0.045 ms\n'
16:56:36 |   asyncio.tcp-connection |<<< b'ping: sendmsg: Network is unreachable\n'
16:56:36 |  moler.net-down-detector |Network is down!
16:56:36 |      moler.user.app-code |Network is down from 16:56:36
16:56:36 |   asyncio.tcp-connection |... closing
16:56:38 |  asyncio.ping.tcp-server |Ping Sim: I'm tired after this client ... will␣
↪do sepuku
'''
```

3.1.2 Curio - network down detector

```
# -*- coding: utf-8 -*-
"""
```

(continues on next page)

(continued from previous page)

```
curio.network_down_detector.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using curio. Requires Python 3.6+.

This example demonstrates basic concept of connection observer - entity
that is fully responsible for:
- observing data coming from connection till it catches what it is waiting for
- parsing that data to have "caught event" stored in expected form
- storing that result internally for later retrieval

Please note that this example is LAYER-1 usage which means:
- observer can't run by its own, must be fed with data (passive observer)
- observer can't be awaited, must be queried for status before asking for data
Another words - low level manual combining of all the pieces.
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import logging
import sys
import os
import time

import curio
from moler.threaded_moler_connection import ThreadedMolerConnection

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow␣
↪finding modules in examples/

from network_toggle_observers import NetworkDownDetector

# ====================== Moler's connection-observer usage =======================


async def ping_observing_task(address):
    logger = logging.getLogger('moler.user.app-code')

    # Lowest layer of Moler's usage (you manually glue all elements):
    # 1. create observer
    net_down_detector = NetworkDownDetector('10.0.2.15')
    # 2. ThreadedMolerConnection is a proxy-glue between observer (speaks str)
    #                                 and curio-connection (speaks bytes)
    moler_conn = ThreadedMolerConnection(decoder=lambda data: data.decode("utf-8"))
    # 3a. glue from proxy to observer
    moler_conn.subscribe(net_down_detector.data_received)

    logger.debug('waiting for data to observe')
    async with curio.meta.finalize(tcp_connection(address)) as tcp_conn:
        async for connection_data in tcp_conn:
            # 3b. glue to proxy from external-IO (curio tcp client connection)
            #    (client has to pass it's received data into Moler's connection)
            moler_conn.data_received(connection_data)
            # 4. Moler's client code must manually check status of observer ...
            if net_down_detector.done():
```

(continues on next page)

(continued from previous page)

```
                # 5. ... to know when it can ask for result
                net_down_time = net_down_detector.result()
                timestamp = time.strftime("%H:%M:%S",
                                          time.localtime(net_down_time))
                logger.debug('Network is down from {}'.format(timestamp))
                break


# ============================================================================

async def tcp_connection(address):
    """Async generator reading from tcp network transport layer"""
    logger = logging.getLogger('curio.tcp-connection')
    logger.debug('... connecting to tcp://{}:{}'.format(*address))
    host, port = address
    sock = await curio.open_connection(host, port)
    async with sock:
        while True:
            data = await sock.recv(128)
            if data:
                logger.debug('<<< {!r}'.format(data))
                yield data
            else:
                break


async def main(host, port):
    # Starting the client
    cli_task = await curio.spawn(ping_observing_task, (host, port))
    await cli_task.join()


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers
    logging.basicConfig(
        level=logging.DEBUG,
        format='%(asctime)s |%(name)25s |%(message)s',
        datefmt='%H:%M:%S',
        stream=sys.stderr,
    )
    # ----------------------------------------------------------------
    import platform
    import selectors
    selector = selectors.DefaultSelector()
    if platform.system() == 'Windows':
        # need workaround:   https://github.com/dabeaz/curio/issues/75
        import socket as stdlib_socket
        dummy_socket = stdlib_socket.socket(stdlib_socket.AF_INET,
                                            stdlib_socket.SOCK_DGRAM)
        selector.register(dummy_socket, selectors.EVENT_READ)
    # ----------------------------------------------------------------
    servers = start_ping_servers([(('localhost', 5679), '10.0.2.15')])
    curio.run(main, 'localhost', 5679, selector=selector)
    stop_ping_servers(servers)

'''
```

(continues on next page)

(continued from previous page)

```
LOG OUTPUT

16:57:07 |    curio.ping.tcp-server |Ping Sim started at tcp://:5679
16:57:07 |    curio.ping.tcp-server |WARNING - I'll be tired too much just after first client!
16:57:07 |      moler.user.app-code |waiting for data to observe
16:57:07 |     curio.tcp-connection |... connecting to tcp://:5679
16:57:09 |    curio.ping.tcp-server |connection accepted - client at tcp://192.168.56.1:56578
16:57:09 |     curio.tcp-connection |<<< b'\n'
16:57:10 |     curio.tcp-connection |<<< b'greg@debian:~$ ping 10.0.2.15\n'
16:57:11 |     curio.tcp-connection |<<< b'PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.\n'
16:57:12 |     curio.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=1 ttl=64 time=0.080 ms\n'
16:57:13 |     curio.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=2 ttl=64 time=0.037 ms\n'
16:57:14 |     curio.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=3 ttl=64 time=0.045 ms\n'
16:57:15 |     curio.tcp-connection |<<< b'ping: sendmsg: Network is unreachable\n'
16:57:15 |  moler.net-down-detector |Network is down!
16:57:15 |      moler.user.app-code |Network is down from 16:57:15
16:57:15 |    curio.ping.tcp-server |Ping Sim: You are right - I'm tired after this client
16:57:15 |             curio.kernel |Kernel <curio.kernel.Kernel object at 0x000000000303E780> shutting down
'''
```

3.1.3 Thread - network down detector

```
# -*- coding: utf-8 -*-
"""
threaded.network_down_detector.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using socket & threading.
Works on Python 2.7 as well as on 3.6

This example demonstrates basic concept of connection observer - entity
that is fully responsible for:
- observing data coming from connection till it catches what it is waiting for
- parsing that data to have "caught event" stored in expected form
- storing that result internally for later retrieval

Please note that this example is LAYER-1 usage which means:
- observer can't run by its own, must be fed with data (passive observer)
- observer can't be awaited, must be queried for status before asking for data
Another words - low level manual combining of all the pieces.
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import logging
```

(continues on next page)

(continued from previous page)

```
import socket
import sys
import os
import threading
import time
from contextlib import closing

from moler.threaded_moler_connection import ThreadedMolerConnection

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow finding modules in examples/

from network_toggle_observers import NetworkDownDetector


# ==================== Moler's connection-observer usage ======================


def ping_observing_task(address):
    logger = logging.getLogger('moler.user.app-code')

    # Lowest layer of Moler's usage (you manually glue all elements):
    # 1. create observer
    net_down_detector = NetworkDownDetector('10.0.2.15')
    # 2. ThreadedMolerConnection is a proxy-glue between observer (speaks str)
    #                                    and threaded-connection (speaks bytes)
    moler_conn = ThreadedMolerConnection(decoder=lambda data: data.decode("utf-8"))
    # 3a. glue from proxy to observer
    moler_conn.subscribe(net_down_detector.data_received)

    logger.debug('waiting for data to observe')
    for connection_data in tcp_connection(address):
        # 3b. glue to proxy from external-IO (threaded tcp client connection)
        #   (client code has to pass it's received data into Moler's connection)
        moler_conn.data_received(connection_data)
        # 4. Moler's client code must manually check status of observer ...
        if net_down_detector.done():
            # 5. ... to know when it can ask for result
            net_down_time = net_down_detector.result()
            timestamp = time.strftime("%H:%M:%S", time.localtime(net_down_time))
            logger.debug('Network is down from {}'.format(timestamp))
            break


# ============================================================================
def tcp_connection(address):
    """Generator reading from tcp network transport layer"""
    logger = logging.getLogger('threaded.tcp-connection')
    logger.debug('... connecting to tcp://{}:{}'.format(*address))
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(address)

    with closing(client_socket):
        while True:
            data = client_socket.recv(128)
            if data:
                logger.debug('<<< {!r}'.format(data))
```

(continues on next page)

(continued from previous page)

```
                yield data
            else:
                logger.debug("... closed")
                break


def main(address):
    # Starting the client
    client_thread = threading.Thread(target=ping_observing_task, args=(address,))
    client_thread.start()
    client_thread.join()


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers

    logging.basicConfig(
        level=logging.DEBUG,
        format='%(asctime)s |%(name)25s |%(message)s',
        datefmt='%H:%M:%S',
        stream=sys.stderr,
    )
    local_address = ('localhost', 5670)
    servers = start_ping_servers([(local_address, '10.0.2.15')])
    main(local_address)
    stop_ping_servers(servers)

'''
LOG OUTPUT

16:58:04 | threaded.ping.tcp-server |Ping Sim started at tcp://localhost:5670
16:58:04 | threaded.ping.tcp-server |WARNING - I'll be tired too much just after␣
↪first client!
16:58:04 |      moler.user.app-code |waiting for data to observe
16:58:04 |  threaded.tcp-connection |... connecting to tcp://localhost:5670
16:58:04 | threaded.ping.tcp-server |connection accepted - client at tcp://127.0.0.
↪1:56582
16:58:04 |  threaded.tcp-connection |<<< b'\n'
16:58:05 |  threaded.tcp-connection |<<< b'greg@debian:~$ ping 10.0.2.15\n'
16:58:06 |  threaded.tcp-connection |<<< b'PING 10.0.2.15 (10.0.2.15) 56(84) bytes of␣
↪data.\n'
16:58:07 |  threaded.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=1 ttl=64␣
↪time=0.080 ms\n'
16:58:08 |  threaded.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=2 ttl=64␣
↪time=0.037 ms\n'
16:58:09 |  threaded.tcp-connection |<<< b'64 bytes from 10.0.2.15: icmp_req=3 ttl=64␣
↪time=0.045 ms\n'
16:58:10 |  threaded.tcp-connection |<<< b'ping: sendmsg: Network is unreachable\n'
16:58:10 |  moler.net-down-detector |Network is down!
16:58:10 |      moler.user.app-code |Network is down from 16:58:10
16:58:10 | threaded.ping.tcp-server |Ping Sim: I'm tired after this client ... bye
16:58:12 | threaded.ping.tcp-server |Connection closed
'''
```

```
# -*- coding: utf-8 -*-
```

(continues on next page)

(continued from previous page)

```
"""
threaded.network_down_multi-detectors.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using socket & threading.
Works on Python 2.7 as well as on 3.6

This example demonstrates multiple connection observers working
on multiple connections.
Shows following concepts:
- multiple observers may observe single connection
- each one is focused on different data (processing decomposition)
- client code may run observers on different connections
- client code may "start" observers in sequence
- external-IO-connection must be given Moler's connection for data forwarding
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import logging
import socket
import sys
import os
import threading
import time
from contextlib import closing

from moler.threaded_moler_connection import ThreadedMolerConnection

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow␣
↪finding modules in examples/

from network_toggle_observers import NetworkDownDetector, NetworkUpDetector


# ==================== Moler's connection-observer usage ====================


def ping_observing_task(address, ping_ip):
    logger = logging.getLogger('moler.user.app-code')
    net_addr = 'tcp://{}:{}'.format(*address)

    # Lowest layer of Moler's usage (you manually glue all elements):
    # 1. create observers
    net_down_detector = NetworkDownDetector(ping_ip)
    net_drop_found = False
    net_up_detector = NetworkUpDetector(ping_ip)
    moler_conn = ThreadedMolerConnection(decoder=lambda data: data.decode("utf-8"))
    # 2. virtually "start" observer by making it data-listener
    moler_conn.subscribe(net_down_detector.data_received)

    info = '{} on {} using {}'.format(ping_ip, net_addr, net_down_detector)
    logger.debug('observe ' + info)
    for _ in tcp_connection(address, moler_conn):
        # anytime new data comes it may change status of observer
```

(continues on next page)

(continued from previous page)

```
        if not net_drop_found and net_down_detector.done():
            net_drop_found = True
            net_down_time = net_down_detector.result()
            timestamp = time.strftime("%H:%M:%S", time.localtime(net_down_time))
            logger.debug('Network {} is down from {}'.format(ping_ip, timestamp))
            # 3. virtually "stop" that observer
            moler_conn.unsubscribe(net_down_detector.data_received)
            # 4. and start subsequent one (to know when net is back "up")
            info = '{} on {} using {}'.format(ping_ip, net_addr, net_up_detector)
            logger.debug('observe ' + info)
            moler_conn.subscribe(net_up_detector.data_received)
        if net_up_detector.done():
            net_up_time = net_up_detector.result()
            timestamp = time.strftime("%H:%M:%S", time.localtime(net_up_time))
            logger.debug('Network {} is back "up" from {}'.format(ping_ip, timestamp))
            # 5. virtually "stop" that observer
            moler_conn.unsubscribe(net_up_detector.data_received)
            break


# ============================================================================
def tcp_connection(address, moler_conn):
    """Forwarder reading from tcp network transport layer"""
    logger = logging.getLogger('threaded.tcp-connection({}:{})'.format(*address))
    logger.debug('... connecting to tcp://{}:{}'.format(*address))
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(address)

    with closing(client_socket):
        while True:
            data = client_socket.recv(128)
            if data:
                logger.debug('<<< {!r}'.format(data))
                # Forward received data into Moler's connection
                moler_conn.data_received(data)
                yield data
            else:
                logger.debug("... closed")
                break


def main(connections2observe4ip):
    # Starting the clients
    connections = []
    for address, ping_ip in connections2observe4ip:
        client_thread = threading.Thread(target=ping_observing_task,
                                         args=(address, ping_ip))
        client_thread.start()
        connections.append(client_thread)
    # await observers job to be done
    for client_thread in connections:
        client_thread.join()


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers
```

(continues on next page)

(continued from previous page)

```
115
116     logging.basicConfig(
117         level=logging.DEBUG,
118         format='%(asctime)s |%(name)-40s |%(message)s',
119         datefmt='%H:%M:%S',
120         stream=sys.stderr,
121     )
122     connections2observe = [(('localhost', 5671), '10.0.2.15'),
123                            (('localhost', 5672), '10.0.2.16')]
124     servers = start_ping_servers(connections2observe)
125     main(connections2observe)
126     stop_ping_servers(servers)
127
128 '''
129 LOG OUTPUT
130
131 16:37:44 |threaded.ping.tcp-server(5671)           |Ping Sim started at tcp://
    ↪localhost:5671
132 16:37:44 |threaded.ping.tcp-server(5672)           |Ping Sim started at tcp://
    ↪localhost:5672
133 16:37:44 |moler.user.app-code                      |observe 10.0.2.15 on tcp://
    ↪localhost:5671 using NetworkDownDetector(id:46109472)
134 16:37:44 |threaded.tcp-connection(localhost:5671)  |... connecting to tcp://
    ↪localhost:5671
135 16:37:44 |moler.user.app-code                      |observe 10.0.2.16 on tcp://
    ↪localhost:5672 using NetworkDownDetector(id:46184544)
136 16:37:44 |threaded.tcp-connection(localhost:5672)  |... connecting to tcp://
    ↪localhost:5672
137 16:37:44 |threaded.ping.tcp-server(5671 -> 61044)  |connection accepted - client at␣
    ↪tcp://127.0.0.1:61044
138 16:37:44 |threaded.tcp-connection(localhost:5671)  |<<< b'\n'
139 16:37:44 |threaded.ping.tcp-server(5672 -> 61045)  |connection accepted - client at␣
    ↪tcp://127.0.0.1:61045
140 16:37:44 |threaded.tcp-connection(localhost:5672)  |<<< b'\n'
141 16:37:45 |threaded.tcp-connection(localhost:5671)  |<<< b'greg@debian:~$ ping 10.0.2.
    ↪15\n'
142 16:37:45 |threaded.tcp-connection(localhost:5672)  |<<< b'greg@debian:~$ ping 10.0.2.
    ↪16\n'
143 16:37:46 |threaded.tcp-connection(localhost:5671)  |<<< b'PING 10.0.2.15 (10.0.2.15)␣
    ↪56(84) bytes of data.\n'
144 16:37:46 |threaded.tcp-connection(localhost:5672)  |<<< b'PING 10.0.2.16 (10.0.2.16)␣
    ↪56(84) bytes of data.\n'
145 16:37:47 |threaded.tcp-connection(localhost:5671)  |<<< b'64 bytes from 10.0.2.15:␣
    ↪icmp_req=1 ttl=64 time=0.080 ms\n'
146 16:37:47 |threaded.tcp-connection(localhost:5672)  |<<< b'64 bytes from 10.0.2.16:␣
    ↪icmp_req=1 ttl=64 time=0.080 ms\n'
147 16:37:48 |threaded.tcp-connection(localhost:5671)  |<<< b'64 bytes from 10.0.2.15:␣
    ↪icmp_req=2 ttl=64 time=0.037 ms\n'
148 16:37:48 |threaded.tcp-connection(localhost:5672)  |<<< b'64 bytes from 10.0.2.16:␣
    ↪icmp_req=2 ttl=64 time=0.037 ms\n'
149 16:37:49 |threaded.tcp-connection(localhost:5671)  |<<< b'64 bytes from 10.0.2.15:␣
    ↪icmp_req=3 ttl=64 time=0.045 ms\n'
150 16:37:49 |threaded.tcp-connection(localhost:5672)  |<<< b'64 bytes from 10.0.2.16:␣
    ↪icmp_req=3 ttl=64 time=0.045 ms\n'
151 16:37:50 |threaded.tcp-connection(localhost:5671)  |<<< b'ping: sendmsg: Network is␣
    ↪unreachable\n'
152 16:37:50 |moler.NetworkDownDetector(id:46109472)   |Network 10.0.2.15 is down!
```

(continues on next page)

(continued from previous page)

```
16:37:50 |moler.user.app-code                        |Network 10.0.2.15 is down from 16:37:50
16:37:50 |moler.user.app-code                        |observe 10.0.2.15 on tcp://localhost:5671 using NetworkUpDetector(id:46110368)
16:37:50 |threaded.tcp-connection(localhost:5672)   |<<< b'ping: sendmsg: Network is unreachable\n'
16:37:50 |moler.NetworkDownDetector(id:46184544)    |Network 10.0.2.16 is down!
16:37:50 |moler.user.app-code                        |Network 10.0.2.16 is down from 16:37:50
16:37:50 |moler.user.app-code                        |observe 10.0.2.16 on tcp://localhost:5672 using NetworkUpDetector(id:46184488)
16:37:51 |threaded.tcp-connection(localhost:5671)   |<<< b'ping: sendmsg: Network is unreachable\n'
16:37:51 |threaded.tcp-connection(localhost:5672)   |<<< b'ping: sendmsg: Network is unreachable\n'
16:37:52 |threaded.tcp-connection(localhost:5671)   |<<< b'ping: sendmsg: Network is unreachable\n'
16:37:52 |threaded.tcp-connection(localhost:5672)   |<<< b'ping: sendmsg: Network is unreachable\n'
16:37:53 |threaded.tcp-connection(localhost:5671)   |<<< b'64 bytes from 10.0.2.15: icmp_req=7 ttl=64 time=0.123 ms\n'
16:37:53 |moler.NetworkUpDetector(id:46110368)      |Network 10.0.2.15 is up!
16:37:53 |moler.user.app-code                        |Network 10.0.2.15 is back "up" from 16:37:53
16:37:53 |threaded.tcp-connection(localhost:5672)   |<<< b'64 bytes from 10.0.2.16: icmp_req=7 ttl=64 time=0.123 ms\n'
16:37:53 |moler.NetworkUpDetector(id:46184488)      |Network 10.0.2.16 is up!
16:37:53 |moler.user.app-code                        |Network 10.0.2.16 is back "up" from 16:37:53
16:37:53 |threaded.ping.tcp-server(5671)             |Ping Sim: ... bye
16:37:53 |threaded.ping.tcp-server(5672)             |Ping Sim: ... bye
16:37:55 |threaded.ping.tcp-server(5671 -> 61044)   |Connection closed
16:37:55 |threaded.ping.tcp-server(5672 -> 61045)   |Connection closed
'''
```

3.1.4 Twisted - network down detector

```
# -*- coding: utf-8 -*-
"""
twisted.network_down_detector.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using twisted.
Works on Python 2.7 as well as on 3.6

This example demonstrates basic concept of connection observer - entity
that is fully responsible for:
- observing data coming from connection till it catches what it is waiting for
- parsing that data to have "caught event" stored in expected form
- storing that result internally for later retrieval

Please note that this example is LAYER-1 usage which means:
- observer can't run by its own, must be fed with data (passive observer)
- observer can't be awaited, must be queried for status before asking for data
Another words - low level manual combining of all the pieces.
```

(continues on next page)

(continued from previous page)

```
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import logging
import sys
import os
import time
from functools import partial

from moler.threaded_moler_connection import ThreadedMolerConnection
from twisted.internet import reactor, task
from twisted.internet.defer import Deferred
from twisted.internet.protocol import Protocol, ClientFactory

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow␣
↪finding modules in examples/

from network_toggle_observers import NetworkDownDetector

# ==================== Moler's connection-observer usage ======================


def ping_observing_task(address):
    logger = logging.getLogger('moler.user.app-code')
    observer_done = Deferred()

    # Lowest layer of Moler's usage (you manually glue all elements):
    # 1. create observer
    net_down_detector = NetworkDownDetector('10.0.2.15')
    # 2. ThreadedMolerConnection is a proxy-glue between observer (speaks str)
    #                                    and twisted-connection (speaks bytes)
    moler_conn = ThreadedMolerConnection(decoder=lambda data: data.decode("utf-8"))
    # 3a. glue from proxy to observer
    moler_conn.subscribe(net_down_detector.data_received)

    logger.debug('waiting for data to observe')

    def feed_moler(connection_data):
        # 3b. glue to proxy from external-IO (twisted tcp client connection)
        #    (client has to pass it's received data into Moler's connection)
        moler_conn.data_received(connection_data)
        # 4. Moler's client code must manually check status of observer ...
        if net_down_detector.done():
            # 5. ... to know when it can ask for result
            net_down_time = net_down_detector.result()
            timestamp = time.strftime("%H:%M:%S",
                                      time.localtime(net_down_time))
            logger.debug('Network is down from {}'.format(timestamp))
            observer_done.callback(None)  # break tcp client and server

    start_tcp_connection(address, feed_moler)
    return observer_done


```

(continues on next page)

(continued from previous page)

```
# ============================================================================
class TcpConnection(Protocol):
    def __init__(self, forward_data):
        self.forward_data = forward_data
        self.logger = logging.getLogger('twisted.tcp-connection')

    def connectionMade(self):
        conn_info = 'tcp://{}:{}'.format(*self.transport.realAddress)
        self.logger.debug('... connected to ' + conn_info)

    def connectionLost(self, reason):
        self.logger.debug("... closed")

    def dataReceived(self, data):
        self.logger.debug('<<< {!r}'.format(data))
        self.forward_data(data)


def start_tcp_connection(address, forward_data):
    host, port = address
    factory = ClientFactory()
    factory.protocol=partial(TcpConnection, forward_data)
    reactor.connectTCP(host, port, factory)


def main(reactor, address):
    # Starting the client
    processing_done_deferred = ping_observing_task(address)
    return processing_done_deferred


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers

    logging.basicConfig(
        level=logging.DEBUG,
        format='%(asctime)s |%(name)40s |%(message)s',
        datefmt='%H:%M:%S',
        stream=sys.stderr,
    )
    address = ('localhost', 5670)
    servers = start_ping_servers([(address, '10.0.2.15')])
    try:
        task.react(main, argv=[address])
    finally:
        stop_ping_servers(servers)

'''
LOG OUTPUT

10:53:39 |         threaded.ping.tcp-server(5670) |Ping Sim started at tcp://
↪localhost:5670
10:53:39 |               moler.runner.thread-pool |created
10:53:39 |               moler.runner.thread-pool |created own executor <concurrent.
↪futures.thread.ThreadPoolExecutor object at 0x7fde953e93c8>
10:53:39 |                    moler.user.app-code |waiting for data to observe
```

(continues on next page)

(continued from previous page)

```
10:53:39 |                 twisted.tcp-connection |... connected to tcp://127.0.0.
↪1:5670
10:53:39 |threaded.ping.tcp-server(5670 -> 35996) |connection accepted - client at␣
↪tcp://127.0.0.1:35996
10:53:39 |                 twisted.tcp-connection |<<< b'\n'
10:53:39 |                     moler.7fde953e95c0 |
10:53:40 |                 twisted.tcp-connection |<<< b'greg@debian:~$ ping 10.0.2.
↪15\n'
10:53:40 |                     moler.7fde953e95c0 |greg@debian:~$ ping 10.0.2.15
10:53:41 |                 twisted.tcp-connection |<<< b'PING 10.0.2.15 (10.0.2.15)␣
↪56(84) bytes of data.\n'
10:53:41 |                     moler.7fde953e95c0 |PING 10.0.2.15 (10.0.2.15) 56(84)␣
↪bytes of data.
10:53:42 |                 twisted.tcp-connection |<<< b'64 bytes from 10.0.2.15:␣
↪icmp_req=1 ttl=64 time=0.080 ms\n'
10:53:42 |                     moler.7fde953e95c0 |64 bytes from 10.0.2.15: icmp_
↪req=1 ttl=64 time=0.080 ms
10:53:43 |                 twisted.tcp-connection |<<< b'64 bytes from 10.0.2.15:␣
↪icmp_req=2 ttl=64 time=0.037 ms\n'
10:53:43 |                     moler.7fde953e95c0 |64 bytes from 10.0.2.15: icmp_
↪req=2 ttl=64 time=0.037 ms
10:53:44 |                 twisted.tcp-connection |<<< b'64 bytes from 10.0.2.15:␣
↪icmp_req=3 ttl=64 time=0.045 ms\n'
10:53:44 |                     moler.7fde953e95c0 |64 bytes from 10.0.2.15: icmp_
↪req=3 ttl=64 time=0.045 ms
10:53:45 |                 twisted.tcp-connection |<<< b'ping: sendmsg: Network is␣
↪unreachable\n'
10:53:45 |                     moler.7fde953e95c0 |ping: sendmsg: Network is␣
↪unreachable
10:53:45 |               moler.net-down-detector |Network is down!
10:53:45 |                   moler.user.app-code |Network is down from 10:53:45
10:53:45 |                 twisted.tcp-connection |... closed
10:53:45 |         threaded.ping.tcp-server(5670) |Ping Sim: ... bye
10:53:47 |threaded.ping.tcp-server(5670 -> 35996) |Connection closed
10:53:47 |               moler.runner.thread-pool |shutting down
'''
```

3.2 Layer 2

3.2.1 Thread - network down detector

```
# -*- coding: utf-8 -*-
"""
threaded.network_down_detectors_no_runner.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using socket & threading.
Works on Python 2.7 as well as on 3.6

This is Layer_2 (half of it) example:
uses Moler provided external-IO TCP implementation (moler.io.raw.tcp.ThradedTcp)
that integrates with Moler's connection

```

(continues on next page)

(continued from previous page)

```
This example demonstrates multiple connection observers working
on multiple connections.

Shows following concepts:
- multiple observers may observe single connection
- each one is focused on different data (processing decomposition)
- client code may run observers on different connections
- client code may "start" observers in sequence
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import logging
import sys
import os
import threading
import time

from moler.threaded_moler_connection import ThreadedMolerConnection
from moler.io.raw import tcp

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow␣
↪finding modules in examples/

from network_toggle_observers import NetworkDownDetector, NetworkUpDetector


# ====================== Moler's connection-observer usage =======================


def ping_observing_task(ext_io_connection, ping_ip):
    """
    Here external-IO connection is abstract - we don't know its type.
    What we know is just that it has .moler_connection attribute.
    """
    logger = logging.getLogger('moler.user.app-code')
    conn_addr = str(ext_io_connection)

    # Layer 2 of Moler's usage (ext_io_connection):
    # 1. create observers
    net_down_detector = NetworkDownDetector(ping_ip)
    net_drop_found = False
    net_up_detector = NetworkUpDetector(ping_ip)
    moler_conn = ext_io_connection.moler_connection
    # 2. virtually "start" observer by making it data-listener
    moler_conn.subscribe(net_down_detector.data_received)

    info = '{} on {} using {}'.format(ping_ip, conn_addr, net_down_detector)
    logger.debug('observe ' + info)

    with ext_io_connection.open():
        observing_timeout = 10
        start_time = time.time()
        while time.time() < start_time + observing_timeout:
            # anytime new data comes it may change status of observer
```

(continues on next page)

(continued from previous page)

```
            if not net_drop_found and net_down_detector.done():
                net_drop_found = True
                net_down_time = net_down_detector.result()
                timestamp = time.strftime("%H:%M:%S", time.localtime(net_down_time))
                logger.debug('Network {} is down from {}'.format(ping_ip, timestamp))
                # 3. virtually "stop" that observer
                moler_conn.unsubscribe(net_down_detector.data_received)
                # 4. and start subsequent one (to know when net is back "up")
                info = '{} on {} using {}'.format(ping_ip, conn_addr, net_up_detector)
                logger.debug('observe ' + info)
                moler_conn.subscribe(net_up_detector.data_received)
            if net_up_detector.done():
                net_up_time = net_up_detector.result()
                timestamp = time.strftime("%H:%M:%S", time.localtime(net_up_time))
                logger.debug('Network {} is back "up" from {}'.format(ping_ip,␣
↪timestamp))
                # 5. virtually "stop" that observer
                moler_conn.unsubscribe(net_up_detector.data_received)
                break
            time.sleep(0.2)


# ============================================================================
def main(connections2observe4ip):
    # Starting the clients
    connections = []
    for address, ping_ip in connections2observe4ip:
        host, port = address
        # 1. create Moler's connection that knows encoding
        decoder = lambda data: data.decode("utf-8")
        moler_conn = ThreadedMolerConnection(decoder=decoder)
        # 2. create external-IO connection gluing to Moler's connection
        conn_logger_name = 'threaded.tcp-connection({}:{})'.format(*address)
        conn_logger = logging.getLogger(conn_logger_name)
        tcp_connection = tcp.ThreadedTcp(moler_connection=moler_conn,
                                         port=port, host=host,
                                         logger=conn_logger)
        client_thread = threading.Thread(target=ping_observing_task,
                                         args=(tcp_connection, ping_ip))
        client_thread.start()
        connections.append(client_thread)
    # await observers job to be done
    for client_thread in connections:
        client_thread.join()


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers

    logging.basicConfig(
        level=logging.DEBUG,
        format='%(asctime)s |%(name)-40s |%(message)s',
        datefmt='%H:%M:%S',
        stream=sys.stderr,
    )
    connections2observe4ip = [(('localhost', 5671), '10.0.2.15'),
```

(continues on next page)

(continued from previous page)

```
                              (('localhost', 5672), '10.0.2.16')]
    servers = start_ping_servers(connections2observe4ip)
    main(connections2observe4ip)
    stop_ping_servers(servers)

'''
LOG OUTPUT

12:06:50 |threaded.ping.tcp-server(5671)           |Ping Sim started at tcp://
↪localhost:5671
12:06:50 |threaded.ping.tcp-server(5672)           |Ping Sim started at tcp://
↪localhost:5672
12:06:50 |moler.user.app-code                      |observe 10.0.2.15 on tcp://
↪localhost:5671 using NetworkDownDetector(id:46425312)
12:06:50 |moler.user.app-code                      |observe 10.0.2.16 on tcp://
↪localhost:5672 using NetworkDownDetector(id:46425704)
12:06:50 |threaded.tcp-connection(localhost:5671)  |connecting to tcp://localhost:5671
12:06:50 |threaded.tcp-connection(localhost:5672)  |connecting to tcp://localhost:5672
12:06:50 |threaded.tcp-connection(localhost:5671)  |connection tcp://localhost:5671␣
↪is open
12:06:50 |threaded.ping.tcp-server(5671 -> 62735)  |connection accepted - client at␣
↪tcp://127.0.0.1:62735
12:06:50 |threaded.tcp-connection(localhost:5672)  |connection tcp://localhost:5672␣
↪is open
12:06:50 |threaded.tcp-connection(localhost:5671)  |< b'\n'
12:06:50 |threaded.ping.tcp-server(5672 -> 62736)  |connection accepted - client at␣
↪tcp://127.0.0.1:62736
12:06:50 |threaded.tcp-connection(localhost:5672)  |< b'\n'
12:06:51 |threaded.tcp-connection(localhost:5672)  |< b'greg@debian:~$ ping 10.0.2.
↪16\n'
12:06:51 |threaded.tcp-connection(localhost:5671)  |< b'greg@debian:~$ ping 10.0.2.
↪15\n'
12:06:52 |threaded.tcp-connection(localhost:5672)  |< b'PING 10.0.2.16 (10.0.2.16)␣
↪56(84) bytes of data.\n'
12:06:52 |threaded.tcp-connection(localhost:5671)  |< b'PING 10.0.2.15 (10.0.2.15)␣
↪56(84) bytes of data.\n'
12:06:53 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=1 ttl=64 time=0.080 ms\n'
12:06:53 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=1 ttl=64 time=0.080 ms\n'
12:06:54 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=2 ttl=64 time=0.037 ms\n'
12:06:54 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=2 ttl=64 time=0.037 ms\n'
12:06:55 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=3 ttl=64 time=0.045 ms\n'
12:06:55 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=3 ttl=64 time=0.045 ms\n'
12:06:56 |threaded.tcp-connection(localhost:5672)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
12:06:56 |moler.NetworkDownDetector(id:46425704)   |Network 10.0.2.16 is down!
12:06:56 |moler.user.app-code                      |Network 10.0.2.16 is down from␣
↪12:06:56
12:06:56 |moler.user.app-code                      |observe 10.0.2.16 on tcp://
↪localhost:5672 using NetworkUpDetector(id:46426096)
12:06:56 |threaded.tcp-connection(localhost:5671)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
```

(continues on next page)

(continued from previous page)

```
12:06:56 |moler.NetworkDownDetector(id:46425312)   |Network 10.0.2.15 is down!
12:06:56 |moler.user.app-code                      |Network 10.0.2.15 is down from␣
↪12:06:56
12:06:56 |moler.user.app-code                      |observe 10.0.2.15 on tcp://
↪localhost:5671 using NetworkUpDetector(id:46425480)
12:06:57 |threaded.tcp-connection(localhost:5672)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
12:06:57 |threaded.tcp-connection(localhost:5671)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
12:06:58 |threaded.tcp-connection(localhost:5672)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
12:06:58 |threaded.tcp-connection(localhost:5671)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
12:06:59 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=7 ttl=64 time=0.123 ms\n'
12:06:59 |moler.NetworkUpDetector(id:46426096)     |Network 10.0.2.16 is up!
12:06:59 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=7 ttl=64 time=0.123 ms\n'
12:06:59 |moler.NetworkUpDetector(id:46425480)     |Network 10.0.2.15 is up!
12:06:59 |moler.user.app-code                      |Network 10.0.2.15 is back "up"␣
↪from 12:06:59
12:06:59 |threaded.tcp-connection(localhost:5671)  |connection tcp://localhost:5671␣
↪is closed
12:06:59 |moler.user.app-code                      |Network 10.0.2.16 is back "up"␣
↪from 12:06:59
12:06:59 |threaded.tcp-connection(localhost:5672)  |connection tcp://localhost:5672␣
↪is closed
12:07:00 |threaded.ping.tcp-server(5671)           |Ping Sim: ... bye
12:07:00 |threaded.ping.tcp-server(5672)           |Ping Sim: ... bye
12:07:01 |threaded.ping.tcp-server(5672 -> 62736)  |Connection closed
12:07:01 |threaded.ping.tcp-server(5671 -> 62735)  |Connection closed
'''
```

3.2.2 Thread - network down detector

```
# -*- coding: utf-8 -*-
"""
threaded.network_down_detectors.py
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A fully-functional connection-observer using socket & threading.
Works on Python 2.7 as well as on 3.6

This is Layer_2 example:
- uses Moler provided external-IO TCP implementation (moler.io.raw.tcp.ThreadedTcp)
- uses full API of connection observer (it has internal concurrency runner)

This example demonstrates multiple connection observers working
on multiple connections.

Shows following concepts:
- multiple observers may observe single connection
- each one is focused on different data (processing decomposition)
- client code may run observers on different connections
- client code may "start" observers in sequence
```

(continues on next page)

(continued from previous page)

```
"""

__author__ = 'Grzegorz Latuszek'
__copyright__ = 'Copyright (C) 2018, Nokia'
__email__ = 'grzegorz.latuszek@nokia.com'

import logging
import sys
import os
import threading
import time

from moler.threaded_moler_connection import ThreadedMolerConnection
from moler.io.raw import tcp

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "..", ".."))  # allow␣
↪finding modules in examples/

from network_toggle_observers import NetworkDownDetector, NetworkUpDetector

# ==================== Moler's connection-observer usage ======================

def ping_observing_task(ext_io_connection, ping_ip):
    """
    Here external-IO connection is abstract - we don't know its type.
    What we know is just that it has .moler_connection attribute.
    """
    logger = logging.getLogger('moler.user.app-code')
    conn_addr = str(ext_io_connection)

    # Layer 2 of Moler's usage (ext_io_connection + runner):
    # 3. create observers on Moler's connection
    net_down_detector = NetworkDownDetector(ping_ip)
    net_down_detector.connection = ext_io_connection.moler_connection
    net_up_detector = NetworkUpDetector(ping_ip)
    net_up_detector.connection = ext_io_connection.moler_connection

    info = '{} on {} using {}'.format(ping_ip, conn_addr, net_down_detector)
    logger.debug('observe ' + info)

    # 4. start observer (nonblocking, using as future)
    net_down_detector.start()  # should be started before we open connection
    # to not loose first data on connection

    with ext_io_connection.open():
        # 5. await that observer to complete
        net_down_time = net_down_detector.await_done(timeout=10)
        timestamp = time.strftime("%H:%M:%S", time.localtime(net_down_time))
        logger.debug('Network {} is down from {}'.format(ping_ip, timestamp))

        # 6. call next observer (blocking till completes)
        info = '{} on {} using {}'.format(ping_ip, conn_addr, net_up_detector)
        logger.debug('observe ' + info)
        # using as synchronous function (so we want verb to express action)
        detect_network_up = net_up_detector
        net_up_time = detect_network_up()
        timestamp = time.strftime("%H:%M:%S", time.localtime(net_up_time))
```

(continues on next page)

(continued from previous page)

```
        logger.debug('Network {} is back "up" from {}'.format(ping_ip, timestamp))


# ============================================================================
def main(connections2observe4ip):
    # Starting the clients
    connections = []
    for address, ping_ip in connections2observe4ip:
        host, port = address
        # 1. create Moler's connection that knows encoding
        decoder = lambda data: data.decode("utf-8")
        moler_conn = ThreadedMolerConnection(decoder=decoder)
        # 2. create external-IO connection gluing to Moler's connection
        conn_logger_name = 'threaded.tcp-connection({}:{})'.format(*address)
        conn_logger = logging.getLogger(conn_logger_name)
        tcp_connection = tcp.ThreadedTcp(moler_connection=moler_conn,
                                         port=port, host=host,
                                         logger=conn_logger)
        client_thread = threading.Thread(target=ping_observing_task,
                                         args=(tcp_connection, ping_ip))
        client_thread.start()
        connections.append(client_thread)
    # await observers job to be done
    for client_thread in connections:
        client_thread.join()


# ============================================================================
if __name__ == '__main__':
    from threaded_ping_server import start_ping_servers, stop_ping_servers

    logging.basicConfig(
        level=logging.DEBUG,
        format='%(asctime)s |%(name)-40s |%(message)s',
        datefmt='%H:%M:%S',
        stream=sys.stderr,
    )
    connections2observe4ip = [(('localhost', 5671), '10.0.2.15'),
                              (('localhost', 5672), '10.0.2.16')]
    servers = start_ping_servers(connections2observe4ip)
    main(connections2observe4ip)
    stop_ping_servers(servers)

'''
LOG OUTPUT

14:10:44 |threaded.ping.tcp-server(5671)           |Ping Sim started at tcp://
↪localhost:5671
14:10:44 |threaded.ping.tcp-server(5672)           |Ping Sim started at tcp://
↪localhost:5672
14:10:44 |moler.user.app-code                      |observe 10.0.2.15 on tcp://
↪localhost:5671 using NetworkDownDetector(id:47689008)
14:10:44 |moler.runner.thread-pool                 |starting␣
↪NetworkDownDetector(id:47689008)
14:10:44 |moler.user.app-code                      |observe 10.0.2.16 on tcp://
↪localhost:5672 using NetworkDownDetector(id:47752080)
14:10:44 |moler.runner.thread-pool                 |starting␣
↪NetworkDownDetector(id:47752080)
```

(continues on next page)

(continued from previous page)

```
14:10:44 |threaded.tcp-connection(localhost:5671)  |connecting to tcp://localhost:5671
14:10:44 |threaded.tcp-connection(localhost:5672)  |connecting to tcp://localhost:5672
14:10:44 |threaded.tcp-connection(localhost:5671)  |connection tcp://localhost:5671␣
↪is open
14:10:44 |threaded.ping.tcp-server(5671 -> 58889)  |connection accepted - client at␣
↪tcp://127.0.0.1:58889
14:10:44 |threaded.tcp-connection(localhost:5671)  |< b'\n'
14:10:44 |moler.runner.thread-pool                 |awaiting␣
↪NetworkDownDetector(id:47689008)
14:10:44 |threaded.tcp-connection(localhost:5672)  |connection tcp://localhost:5672␣
↪is open
14:10:44 |moler.runner.thread-pool                 |awaiting␣
↪NetworkDownDetector(id:47752080)
14:10:44 |threaded.ping.tcp-server(5672 -> 58890)  |connection accepted - client at␣
↪tcp://127.0.0.1:58890
14:10:44 |threaded.tcp-connection(localhost:5672)  |< b'\n'
14:10:45 |threaded.tcp-connection(localhost:5671)  |< b'greg@debian:~$ ping 10.0.2.
↪15\n'
14:10:45 |threaded.tcp-connection(localhost:5672)  |< b'greg@debian:~$ ping 10.0.2.
↪16\n'
14:10:46 |threaded.tcp-connection(localhost:5671)  |< b'PING 10.0.2.15 (10.0.2.15)␣
↪56(84) bytes of data.\n'
14:10:46 |threaded.tcp-connection(localhost:5672)  |< b'PING 10.0.2.16 (10.0.2.16)␣
↪56(84) bytes of data.\n'
14:10:47 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=1 ttl=64 time=0.080 ms\n'
14:10:47 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=1 ttl=64 time=0.080 ms\n'
14:10:48 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=2 ttl=64 time=0.037 ms\n'
14:10:48 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=2 ttl=64 time=0.037 ms\n'
14:10:49 |threaded.tcp-connection(localhost:5671)  |< b'64 bytes from 10.0.2.15: icmp_
↪req=3 ttl=64 time=0.045 ms\n'
14:10:49 |threaded.tcp-connection(localhost:5672)  |< b'64 bytes from 10.0.2.16: icmp_
↪req=3 ttl=64 time=0.045 ms\n'
14:10:50 |threaded.tcp-connection(localhost:5671)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
14:10:50 |moler.NetworkDownDetector(id:47689008)   |Network 10.0.2.15 is down!
14:10:50 |threaded.tcp-connection(localhost:5672)  |< b'ping: sendmsg: Network is␣
↪unreachable\n'
14:10:50 |moler.NetworkDownDetector(id:47752080)   |Network 10.0.2.16 is down!
14:10:50 |moler.runner.thread-pool                 |shutting down
14:10:50 |moler.runner.thread-pool                 |NetworkDownDetector(id:47689008)␣
↪returned 1519823450.4585001
14:10:50 |moler.user.app-code                      |Network 10.0.2.15 is down from␣
↪14:10:50
14:10:50 |moler.user.app-code                      |observe 10.0.2.15 on tcp://
↪localhost:5671 using NetworkUpDetector(id:47689568)
14:10:50 |moler.runner.thread-pool                 |starting␣
↪NetworkUpDetector(id:47689568)
14:10:50 |moler.runner.thread-pool                 |awaiting␣
↪NetworkUpDetector(id:47689568)
14:10:50 |moler.runner.thread-pool                 |shutting down
14:10:50 |moler.runner.thread-pool                 |NetworkDownDetector(id:47752080)␣
↪returned 1519823450.4620001
14:10:50 |moler.user.app-code                      |Network 10.0.2.16 is down from␣
↪14:10:50
```

(continues on next page)

(continued from previous page)

```
14:10:50 |moler.user.app-code                        |observe 10.0.2.16 on tcp://localhost:5672 using NetworkUpDetector(id:47752472)
14:10:50 |moler.runner.thread-pool                   |starting NetworkUpDetector(id:47752472)
14:10:50 |moler.runner.thread-pool                   |awaiting NetworkUpDetector(id:47752472)
14:10:51 |threaded.tcp-connection(localhost:5671)    |< b'ping: sendmsg: Network is unreachable\n'
14:10:51 |threaded.tcp-connection(localhost:5672)    |< b'ping: sendmsg: Network is unreachable\n'
14:10:52 |threaded.tcp-connection(localhost:5671)    |< b'ping: sendmsg: Network is unreachable\n'
14:10:52 |threaded.tcp-connection(localhost:5672)    |< b'ping: sendmsg: Network is unreachable\n'
14:10:53 |threaded.tcp-connection(localhost:5671)    |< b'64 bytes from 10.0.2.15: icmp_req=7 ttl=64 time=0.123 ms\n'
14:10:53 |moler.NetworkUpDetector(id:47689568)       |Network 10.0.2.15 is up!
14:10:53 |threaded.tcp-connection(localhost:5672)    |< b'64 bytes from 10.0.2.16: icmp_req=7 ttl=64 time=0.123 ms\n'
14:10:53 |moler.NetworkUpDetector(id:47752472)       |Network 10.0.2.16 is up!
14:10:53 |moler.runner.thread-pool                   |shutting down
14:10:53 |moler.runner.thread-pool                   |NetworkUpDetector(id:47689568) returned 1519823453.459
14:10:53 |moler.user.app-code                        |Network 10.0.2.15 is back "up" from 14:10:53
14:10:53 |moler.runner.thread-pool                   |shutting down
14:10:53 |moler.runner.thread-pool                   |NetworkUpDetector(id:47752472) returned 1519823453.4625
14:10:53 |moler.user.app-code                        |Network 10.0.2.16 is back "up" from 14:10:53
14:10:53 |threaded.tcp-connection(localhost:5671)    |connection tcp://localhost:5671 is closed
14:10:53 |threaded.tcp-connection(localhost:5672)    |connection tcp://localhost:5672 is closed
14:10:53 |threaded.ping.tcp-server(5671)             |Ping Sim: ... bye
14:10:53 |threaded.ping.tcp-server(5672)             |Ping Sim: ... bye
14:10:55 |threaded.ping.tcp-server(5671 -> 58889)    |Connection closed
14:10:55 |threaded.ping.tcp-server(5672 -> 58890)    |Connection closed
'''
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

W

Z