RENAT

Library version:	RENAT 0.1.10
Library scope:	global
Named arguments:	supported

Introduction

Document for RENAT framework

All in one pdf renat.pdf

Libraries

RENAT includes following libraries:

Common:
    Common library of RENAT
VChannel:
    Library controls connection to targets (servers, routers, ...)
Logger:
    Library provides enhanced loggging keywords
Optical:
    Library provides keywords to control L1 switches, includes mod_calient mod, mod_ntm mod
Router:
    Library provides keywords to control routers, includes mod_juniper mod , mod_cisco mod and mod_gr mod
Tester:
    Library provides keywords to control testers, includes mod_ixnet , mod_ixload and mod_ixbps
WebApp:
    Common library for web application, includes 2 child libraries: Samurai and Arbor
Hypervisor:
    Library provides keywords to control Hypervisor, included mod_vmware
LabKeyword:
    Common lab keywords

Others

Changes:
    Changes information

Choose each libraries for detail infomration and samples about keywords.

Shortcuts

Keywords

Keyword	Arguments	Documentation

Altogether 0 keywords.
Generated by Libdoc on 2018-09-23 09:18:15.

Common

Library version:	RENAT 0.1.10
Library scope:	global
Named arguments:	supported

Introduction

Common library for RENAT

It loads config files and create necessary varibles. The file should be the 1st library included from any test case.

Table of Contents

Configuration file

Global configuration

There are 2 important configuration files. The global configuration files (aka master files) include device information, authentication etc that are used for all the test cases in the suite. The local configuration file `local.yaml` includes information about nodes, tester ports etc. that are used in a specific test case.

At the beginning, the module makes a local copy the master files and initialize necessary variables.

The RENAT framework utilized the YAML format for its configurations file.

The master files folder is defined by `renat-master-folder` in `$RENAT_PATH/config/config.yaml`. Usually, users do not need to modify the master files. The most common case is when new device is deployed, the `device.yaml` need to be update so that device could be used in the test cases.

1. device.yaml: contains global device information

Each device information is store under `device` block and has the following format:

```
<node_name>
  type:         <device type>
  description:    <any useful description>
  ip:           <the IPv4 address of the device
```

Where <node_name> is the name of the device. It could be the name of a switch, router or a web appliance box and should be uniq between the devices. <description> is any useful information and <ip> is the IP that RENAT uses to access the device.

<type> is important because it will be used as the ky of the `access_template` in template file. Usually users do not need to invent a new type but should use the existed type. When a new platform need to be supported, a new type will be introduced with the correspon template and authentication information.

Samples:

```
device:
  apollo:
    type: ssh-host
    description: main server
    ip: 10.128.3.101
  artermis:
    type: ssh-host
    description: second server
    ip: 10.128.3.91
  vmx11:
    type: juniper
    description: r1
    ip: 10.128.64.11
  vmx12:
    type: juniper
    description: r2
    ip: 10.128.64.12
```

2. template.yaml: contains device template information

The template file contains information about how to access to the device and how it should polling information (SNMP only for now). Each template has the following format:

<type>: access: <ssh or telnet> auth: <plaint-text or public-key> profile: <authentication profile name> prompt: <a regular expression for the PROMPT of the CLI device> (optional) login_prompt: <a login PROMPT for CLI device> (optional) password_prompt:<a PROMPT for asking password of CLI device> (optional) append: <a pharase to append automatically for every CLI command that executes> on this device (optional> init: <an array of command that will be executed automatically after a sucessful login of CLI device> (optional)

Note: Becareful about the prompt field. Usually RENAT will wait until it could see the prompt in its output. A wrong prompt will halt the system until it is timed out.

Samples:

```
access-template:
  ssh-host:
    access: ssh
    auth: public-key
    profile: default
    prompt: \$
    append:
    init: unalias -a
  juniper:
    access: telnet
    auth: plain-text
```

```
      profile: default
      prompt: "(#|>) "
      append: ' | no-more'
      init:
    cisco:
      access: ssh
      auth: plain-text
      profile: default
      prompt: "\@.*(#|>) "
      append:
      init:
snmp-template:
    juniper:
        mib: ./mib-Juniper.json
        community: public
        poller: renat
    cisco:
        mib: ./mib-Cisco.json
        community: public
```

3. auth.yaml: contains authentication information

The file contains authentication information that system uses when access to a device. Each authencation type has follwing format:

```
plain-text
  <profile>
    user:     <user name>
    password:  <password>
```

or

```
public-key:
  <profile>:
    user:     <user name>
    key:      <public key path>
```

Where <profile> is the name of the authentication profile specificed in the `access template` of the device

Sample:

```
auth:
  plain-text:
    default:
      user: user
      pass: nttXXX
    flets:
      user: user
      pass: lpcoXXXX
    arbor:
      user: admin
      pass: nttXXX

  public-key: # for Public Key authentication
    default:
      user: robot
      key: /home/user/.ssh/robot_id_rsa
    test:
      user: jenkins
      key:  /var/lib/jenkins/.ssh/id_rsa
```

Local Configuration

Local configuration (aka `local.yaml`) was used by a test case of its sub test cases. Test cases could includes several test cases (the sub level is not limited). The local configuration is defined by `local.yaml` in the `config` folder of each test case. If a test case does not has the `local.yaml` in its `config` folder, it will use the `local.yaml` file in its parent test case and so on. This will help users to share the test information for related test case without having the same `local.yaml` for each test case (**Note:** this feature is enabled from RENAT 0.1.4). The `local.yaml` that is really used for the test is called `active local.yaml`.

When user used the wizard `item.sh` to create a new test case, they have the ability to crete new `local.yaml` or not. `local.yaml` could be edited and inserted new information later to hold more informations for the test case.

When a test is run, it will display its current active `local.yaml`

The local configuration file of each test item is stored in the `config` folder of the item as *`local.yaml*

Usually the `local.yaml` has following parts:

- CLI node information: started by `node` keyword
- WEB node information: started by `webapp` keyword
- Tester device information: started by `tester` keyword
- Default information: automatically created and started by `default` keyword
- And other neccessary information for the test by yaml format

Sample:

```
# CLI node
node:
  vmx11:
    device: vmx11
    snmp_polling: yes
  vmx12:
    device: vmx11
    snmp_polling: yes
  apollo:
    device: vmx11
    snmp_polling: yes
```

```
# web application information
webapp:
  arbor-sp-a:
    device: arbor-sp-a
    proxy:
      http:  10.128.8.210:8080
      ssl:   10.128.8.210:8080
      socks: 10.128.8.210:8080

# Tester information
tester:
  tester01:
    type: ixnet
    ip: 10.128.32.70
    config: vmx_20161129.ixncfg

# Other user information|
port-mapping:
  uplink01:
    device: vmx11
    port: ge-0/0/0
  downlink01:
    device: vmx12
    port: ge-0/0/2

# Default information
default:
  ignore_dead_node: yes
  terminal:
    width: 80
    height: 32
  result_folder: result
```

Variables

The module automatically create `GLOBAL` & `LOCAL` variable for other libraries. It also creates global list variables *GLOBAL,LOCAL and* `NODE` *that could be accessed from* Robot Framework` test cases.

The GLOBAL variable holds all information defined by the master files and LOCAL variable holds all variables defined by active `local.yaml`. And `NODE` is a list that hold all active nodes defined in the `local.yaml`.

Users could access to the information of a key in `local.yaml` by `${LOCAL['key']}`, information of a node by `${LOCAL['node']['vmx11']}` or simply `$NODE['vmx']`. When a keyword need a list of current node, `@{NODE}` could be used.

Notes: By default, RENAT will stop and raise an exception if connection to a node is failed. But if `ignore_dead_node` is defined as `yes` (default) is the current active `local.yaml`, RENAT will omit an warning but keep running the test and remove the node from its active node list.

Shortcuts

Change Mod · **C**leanup Result · **C**onvert Html To Pdf · **C**ount Keyword · **C**ount Keyword Line · **C**ount Match Regexp · **C**reate Sequence · **C**sv Concat · **C**sv Merge · **C**sv Select · **D**iff File · **E**rr · **E**rror Line Should Not Be Bigger Than · **E**rror Should Not Be Bigger Than · **E**xplicit Run · **F**ile Md5 · **F**old Str · **F**ollow Syslog And Trap · **G**et Config Path · **G**et Config Value · **G**et File Without Error · **G**et Item Config Path · **G**et Item Name · **G**et Myid · **G**et Renat Path · **G**et Result Folder · **G**et Result Path · **G**et Test Device · **I**s Stable · **K**eyword Line Should Not Be Bigger Than · **K**eyword Should Not Be Bigger Than · **L**oad Plugin · **L**og · **L**og Csv · **L**og To Console · **L**oop For Node Tag · **M**d 5 · **M**erge Files · **M**ib For Node · **N**ode With Attr · **N**ode With Tag · **N**ode Without Tag · **P**ause · **P**ing Until Ok · **R**andom Name · **R**andom Number · **R**enat Version · **S**et Multi Item Variable · **S**et Result Folder · **S**lack · **S**tr 2 Seq · **V**ersion · **W**ait

Keywords

Keyword	Arguments	Documentation

Change Mod	*name*, *mod*, *relative=False*	Changes file mod, likes Unix chmod
		`mod` is a string specifying the privilege mode `relative` is `False` or `True`
		Examples:
		`Common.Change Mod` | `tmp` | `0775`

Cleanup Result	*ignore=^(log.html|output.xml|report.html)$*	Cleans up the result folder Deletes all files in current active folder that does not match the `ignore` expression and are older than the time the test has started. **Note**: The keyword only removes files but not folders
Convert Html To Pdf	*html_file*, *pdf_file*	Converts html file to pdf file
Count Keyword	*keyword*, **pattern_list*	Count the keyword in files. Keyword is not case-sensitive
Count Keyword Line	*keyword*, **pattern_list*	Count the number of lines contains the `keyword` **Notes:** Keyword is matched partially. For example, `error` or `errorXXX` will be matched by `error` keyword.
Count Match Regexp	*regexp*, **pattern_list*	Count the number of `regex` found in `pattern_list` Examples: `${err_num}=` | `Count Match RegExp` | `.*error.*` | `result/*.csv` | `result/*.txt`
Create Sequence	*start*, *end*, *interval*, *option=float*	Creates a list with number from `start` to `end` with `interval` Example:

		@{list}= *Create Sequence* 10 15 0.5 will create a list of [11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5]
Csv Concat	*src_pattern*, *dst_name*, *input_header=None*, *result_header=True*	Concatinates CSV files vertically If the CSV files has header, set has_header to ${TRUE} Examples: Commmon.*CSV Concat* config/data0[3,4].csv result/result2.csv Commmon.*CSV Concat* config/data0[3,4].csv result/result2.csv has_header=${TRUE}
Csv Merge	*src_pattern*, *dst_name*, *input_header=None*, *key=0*, *select_column=:*, *result_header=True*	Merges all CSV files horizontally by key key from src_pattern input_header defines whether the input files has header row or not. If input_header is ${NULL}, the keyword assume that input files have no header and automatically define columns name. When input_header is not null (default is zero), the row define by input_header will be used as header and data is counted from the next row. select_column is a string that define the output columns and key is the column name that used to merge. When input_header is ${NULL}, select_column and *key* is the index of columns. Otherwise, they are *column name*. The result header (column names) is decided by result_header (*True* or *False*) The keyword returns False if no file is found by the pattern Examples: Common.*CSV Merge* config/data0[3,4].csv result/result2.csv Common.*CSV Merge* config/data0[3,4].csv result/result2.csv input_header=0 Common.*CSV Merge* src_pattern=${RESULT_FOLDER}/balance*.csv input_header=0 ... dst_name=${RESULT_FOLDER}/result.csv result_header=${FALSE} ... key=Stat Name select_column=Valid Frames Rx. Common.*CSV Merge* src_pattern=${RESULT_FOLDER}/balance*.csv input_header=${NULL} ... dst_name=${RESULT_FOLDER}/result.csv result_header=${FALSE} ... key=0 select_column=5
Csv Select	*src_file*, *dst_file*, *str_row=:*, *str_col=:*, *has_header=None*	Select part of the CSV file and write it to other file str_row and str_col are used to specify necessary rows and columns. They are using the same format with slice for Python list. ▪ : and : means all rows and columns ▪ :2 and : means first 2 rows and all columns ▪ : and 1,2 means all rows and 2nd and 3rd columns ▪ 0:3 and 1 means 3 rows from the 1st one(0,1,2) and second column ▪ 0:5:2 and 1 means 3 rows(0,3,5) and second column **Notes:** ▪ Rows and columns are indexed from zero ▪ When ':' is used, the string has format: <start>:<stop> or <start>:<stop>:<step> For convenience, ':' means all the data, ':x' means first 'x' data Examples: *CSV Select* result/data05.csv result/result3.csv 0,1,2 0,1 *CSV Select* result/data05.csv result/result4.csv : 0,1 *CSV Select* result/data05.csv result/result5.csv :2 : *CSV Select* result/data05.csv result/result6.csv 0:3 : *CSV Select* result/data05.csv result/result7.csv 0:5:2 :
Diff File	*path1*, *path2*, *newline=True*	Shows difference between files Returns the diff result (multi lines) path1, path2 are absolute paths.
Err	*msg*	Prints error msg to console
Error Line Should Not Be Bigger Than	*num*, **pattern_list*	Checks whether the number of lines that contains error be less than a number
Error Should Not Be Bigger Than	*num*, **pattern_list*	Checks whether the number of error be less than a number
Explicit Run		skip the test case if global_variable RUN_ME is not defined Sample scenario: 00. Cabling Common.*Explicit Run* Log To Console cabling... run.sh will bypass 00. Cabling by default. In other to run this test case *${FORCE}* needs declared globally run.sh -X -v FORCE
File Md5	*path*	Returns MD5 hash of a file path is an absolute path
Fold Str	*str*	Folds a string by adding Non-Width-Space char (0x200b) at 6th char
Follow Syslog And Trap	*pattern*, *log_file_name=syslog-trap.log*, *delay_str=1s*	Pauses the execution and wait for the pattern is matched if the file *log_file_name* located in the current result folder.

		By default the *log_file_name* is *./result/syslog-trap.log* which is created by *Follow Syslog and Trap* keyword. The keyword should be in tests between *Follow Syslog adn Trap Start* and *Follow Syslog and Trap Stop* keywords.
Get Config Path		Returns absolute path of RENAT config folder path
Get Config Value	*key*, *base=default*	Returns value of a key for renat configuration with this other LOCAL[base][key] > GLOBAL[base][key] > None
Get File Without Error	*file_path*	Get content of the file and return null string if the file does not exist
Get Item Config Path		Returns absolute path of current item config folder
Get Item Name		Returns the name of the running item
Get Myid		
Get Renat Path		Returns the absolute path of RENAT folder
Get Result Folder		Returns current result folder name. Default is `result` in current test case. **Note**: the keyword only returns the `name` of the result folder not its absolue path.
Get Result Path		Returns absolute path of the current result folder
Get Test Device		Return a list of all test device that is used in this test **Notes:** Device number could less than node number
Is Stable	*seq*, *threshold*, *percentile=90*	Checks if the value sequence is stable or not
Keyword Line Should Not Be Bigger Than	*num*, *keyword*, **pattern_list*	Checks whether the number of line containing the keyword be less than a number
Keyword Should Not Be Bigger Than	*num*, *keyword*, **pattern_list*	Checks whether the number of keyword be less than a number
Load Plugin		Load plugin in renat/plugin folder
Log	*msg*, *level=1*	Logs `msg` to the current log file (not console) The `msg` will logged only if the level is bigger than the global level `${DEBUG}` which could be defined at runtime. If `${DEBUG}` is not defined, it will be considered as the default `level` as 1. Examples: Common.*Log* XXX # this always be logged Common.*Log* AAA level=2 # this will not be logged with common run.sh Common.*Log* BBB level=2 # ./run.sh -v `DEBUG:2` will log the message **Notes**: For common use ■ level 1: is default ■ level 2: is debug mode ■ level 3: is very informative mode
Log Csv	*csv_file*, *index=False*, *border=0*	Logs a content of `csv_file` into default log.html *index*, *border* are table attributes
Log To Console	*msg*, *level=1*	Logs a message to console See Common.*Print* for more details about debug level
Loop For Node Tag	*var*, *tags*, **keywords*	Repeatly executes RF `keyword` for nodes that has tag `tags` multi tags are separated by *:* keywords has same meaning with `keywords` used by *Run Keywords* of RobotFramework (keyword and its arguments are separated by `AND` with the others. Example: *Loop For Node Tag* \${node} tag1 ... *Switch* \${node} AND ... *Cmd* show system user AND ... *Cmd* show system uptime **Note:** `$` in variable name must be escaped
Md 5	*str*	Returns MD5 hash of a string
Merge Files	*path_name*, *file_name*	Merges all the text files defined by `path_name` to `file_name` Example: *Merge Files* ./result/*.csv ./result/test.csv
Mib For Node	*node*	Returns the mib file name for this `node` mib file is define by `mib` keyword under the `node` in `local.yaml` ... node: vmx11: device: vmx11 snmp_polling: yes mib: mib11.txt ... Default value is defined by `mib` keyword from global `config/snmp-template.yaml` for the `type` of the node Example:

		${mib}= Common.*MIB For Node* vmx11
Node With Attr	*attr_name*, *value*	Returns a list of nodes which have attribute `attr_name` with value `value`

Node With Tag	**tag_list*	Returns list of `node` or `webapp` from `local.yaml` that has **ALL** tags defined by `tag_list`
		Tag was defined like this in local.yaml
		`vmx11:`
		`  device: vmx11`
		`  snmp_polling: yes`
		`  tag:`
		`  - tag1`
		`  - tag2`
		Examples:
		${test3}= Common.*Node With Tag* tag1 tag3

Node Without Tag	**tag_list*	Returns list of `node` from `local.yaml` that **does not has ANY** tags defined by `tag_list`
		Tag was defined like this in local.yaml
		`vmx11:`
		`  device: vmx11`
		`  snmp_polling: yes`
		`  tag:`
		`  - tag1`
		`  - tag2`
		Examples:
		${test3}= Common.*Node Without Tag* tag1 tag3

Pause	*msg=*, *time_out=3h*, *error_on_timeout=True*, *default_input=*	Displays the message `msg` and pauses the test execution and wait for user input In case of `error_on_timeout` is True(default), the keyword will raise an error when timeout occurs. Otherwise, it will continue the test. **Notes:** If the variable `${RENAT_BATCH}` was defined, the keyword will print out the message and keeps running without pausing. Examples: Common.*Pause* Waiting... 10s error_on_timeout=${TRUE} default input Common.*Pause* Waiting... 10s
Ping Until Ok	*node*, *wait_str=5s*, *extra=-c 3*	Ping a `node` until it gets response. Then wait for more `wait_str` Default `extra` option is `-c 3`
Random Name	*base*, *a=0*, *b=99*	Returns a random name by a *base* and a random number between [a,b] Example: ${FOLDER}= *Random Name* capture_%05d 0 99
Random Number	*a=0*, *b=99*	Returns a random number between [a,b]
Renat Version		Returns RENAT version string
Set Multi Item Variable	**vars*	Set multiple varibles to be *suite variable* at the same time Suite variables (or item variable) could be access anywhere in all the item scenario.
Set Result Folder	*folder*	Sets the result folder to `folder` and return the old result folder. The result folder contains all output files from the test likes tester ouput, config file ... `folder` is a folder name that under current test case folder The system will create a new folder if it does not exist and set its mode to *0775* **Note:** Result folder should be set at the begining of the test. Changing result folder only has effect on up comming connection
Slack	*msg*, *channel=#automation_dev*, *user=renat*, *host=10.128.3.103:4713*	Post a message to Slack
Str 2 Seq	*str_index*, *size*	Returns a sequence from string format Samples: *Str2Seq* :: 5 # (0,1,2,3,4) *Str2Seq* :2 5 # (0,1) *Str2Seq* 1:3 5 # (1,2) *Str2Seq* 0:5:2 5 # (0,2,4)
Version		Returns the current version of RENAT
Wait	*wait_time*, *size=10*	Waits for *wait-time* and display the proress bar *wait_time* used RF *DateTime* format. Examples: Common.*Wait* wait_time=30s size=10

VChannel

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A basic library that provides Terminal connection to routers/hosts

`VChannel` is a core RENAT library that maintains input/output to nodes with an attached virtual terminal. It encapsulates the SSH/Telnet connections behind and provides common usage of access and execute commands to the nodes. Each channel instance has its own log file and a virtual terminal.

Table of Contents

Device, Node and Channel

RENAT has 3 types of connection target. `Device`, `Node` and `Channel`.

Device

Each device stands for a real physical box that has its own IP address and is defined in the master file `device.yaml`. Users do not directly use `device` in keywords.

Node

Node is a logical instance of a `device`. It could stand for a logical instance of a router or just a virtual terminal to the router. Nodes were defined in `local.yaml` of the test case. Several nodes could point to a same device.

Channel

Each channel holds a session to a node. Each channel has its own log file and a virtual terminal. Any command used by *Cmd*,*Write* or *Read* will be logged to the log file. Each channel is identified by a name when it is created with *Connect* keyword and is released with *Close* keyword.

Notes: multi sessions to a same device could be done with predefined multi nodes to same device in the `local.yaml` file or by using multi *Connect* with different *name*.

Connections

The library provides a channel to a target node. Each channel is attached with a virtual terminal. Input and output to the node are made through this virtual terminal. This will help to provide the output looks like the output when operator is using the real terminal.

When keywords *Read*, *Write*, *Cmd* are used, if the connection is not available anymore, the system will try to reconnect to the host with the information provided in the 1st connect. It will try `max_retry_for_connect` times and wait for `interval_between_retry` seconds between retries. The values of `max_retry_for_connect` and `interval_between_retry` are defined in `./config/config.yaml`

Usually when RENAT could not make the connections to the target, the system will raise an exception. But if the `ignore_dead_node` is defined as `yes` in the current active `local.yaml`, the system will ignore the dead node, remove it from the global variable `LOCAL['node']` and `NODE` and keep running the test.

Shortcuts

Change Log · **C**hange Prompt · **C**lose · **C**lose All · **C**md · **C**md And Wait For · **C**md Yesno · **C**onnect · **C**onnect All · **C**urrent Prompt · **E**xec File · **F**lush All · **G**et Channel · **G**et Channels · **G**et Current Channel · **G**et Ip · **L**og · **R**ead · **R**econnect · **S**et Log Separator · **S**nap · **S**nap Diff · **S**tart Screen Mode · **S**top Screen Mode · **S**witch · **W**rite

Keywords

Keyword	Arguments	Documentation
Change Log	*log_file*, *mode=w*	Stops current log file and create a new log file. Every log from that point will be saved to the new log file Return old log filename
Change Prompt	*str_prompt*	Changes the current prompt of the channel Returns previous prompt. User should change the prompt `before` execute the new command that expects to see new prompt. Example:

Router.*Switch*	vmx11	
${prompt}=	VChannel.*Change Prompt*	%
VChannel.*Cmd*	start shell	
VChannel.*Cmd*	ls	

		VChannel.*Change Prompt* ${prompt} Vchannel.*Cmd* exit
Close		Closes current connection and returns the active channel name
Close All		Closes all current sessions and flush out all log files. Current node name was reset to `None`
Cmd	*command=, prompt=, match_err=(unknown command.|syntax error, expecting <command>.)*	Executes a `command` and wait until for the prompt. This is a blocking keyword. Execution of the test case will be postponed until the prompt appears. If `prompt` is a null string (default), its value is defined in the `./config/template.yaml` Output will be automatically logged to the channel current log file. See Common for details about the config files.
Cmd And Wait For	*command, keyword, interval=30s, max_num=10, error_with_max_num=True*	Execute a command and expect `keyword` occurs in the output. If not wait for `interval` and repeat the process again After `max_num`, if `error_with_max_num` is `True` then the keyword will fail. Ortherwise the test continues.
Cmd Yesno	*cmd, ans=yes, question=? [yes,no]*	Executes a `cmd`, waits for `question` and answers that by `ans`
Connect	*node, name, log_file, timeout=20m, w=80, h=32, mode=w*	Connects to the node and create a VChannel instance Login information is automatically extracted from yaml configuration. By defaullt a virtual terminal (vty100) with size 80x64 is attachted to this channel. If a login was successful, VChannel will create a log file name `log_file` for the connection in the current result folder of the test case. This log file will contain any command input/output executed on this channel. Multi sessions to the same node could be open with different names. Use *Switch* to change the current active session by its name Examples: *Connect* vmx11 vmx11 vmx11.log *Connect* vmx11 vmx11 vmx11.log 80 64 See `Common` for more detail about the yaml config files.
Connect All	*prefix=*	Connects to **all** nodes that are defined in active `local.yaml`. A prefix `prefix` was appended to the alias name of the connection. A new log file by `<alias>.log` was automatiocally created. See *Common* for more detail about active `local.yaml`
Current Prompt		Return current prompt

Exec File	*file_name, vars=, comment=# , step=False, str_error=syntax,rror*	Executes commands listed in `file_name` Lines started with `comment` character is considered as comments
		`file_name` is a file located inside the `config` folder of the test case.
		This command file could be written in Jinja2 format. Default usable variables are `LOCAL` and `GLOBAL` which are identical to `Common.LOCAL` and `Common.GLOBAL`. More variables could be supplied to the template by `vars`.
		`vars` has the format: `var1=value1,var2=value2`
		If `step` is `True`, after very command the output is check agains an error list. And if a match is found, execution will be stopped. Error list is define by `str_err`, that contains multi regular expression separated by a comma. Default value of `str_err` is *error*
		A sample for command list with Jinja2 template:
		`show interface {{ LOCAL['extra']['line1'] }}`
		`show interface {{ LOCAL['extra']['line2'] }}`
		`{% for i in range(2) %}`
		`show interface et-0/0/{{ i }}`
		`{% endfor %}`
		Examples:
		Router.*Exec File* cmd.lst
		Router.*Exec File* step=${TRUE} str_error=syntax,error
		Note: Comment in the middle of the line is not supported For example if `comment` is "# "
		`# this is comment line <-- this line will be ignored`
		`## this is not an comment line, and will be enterd to the router cli,`
		but the router might ignore this

Flush All		
Get Channel	*name*	Returns a channel by its `name`

Get Channels		Returns all current vchannel instances
Get Current Channel		Returns the current active channel
Get Ip		Returns the IP address of current node Examples: | ${router_ip}= | Router.*Get IP* |
Log	*msg*	Writes the log message `msg` to current log file of the channel
Read	*silence=False*	Returns the current output of the virtual terminal and automatically logs to file. In `normal mode` this will return the **unread** output only, not all the content of the screen.
Reconnect	*name*	Reconnects to the `name` node using existed information The only difference is that the mode of the log file is set to `a+` by default
Set Log Separator	*sep=*	Set a separator between the log of `read`, `write` or `cmd` keywords
Snap	*name*, **cmd_list*	Remembers the result of a list of command defined by `cmd_list` Use this keyword with *Snap Diff* to get the difference between the command's result. The a new snapshot will overrride the previous result. Each snap is identified by its `name`
Snap Diff	*name*	Executes the comman that have been executed before by `name` snapshot and return the difference. Difference is in `context diff` format
Start Screen Mode		Starts the `screen mode`. In the `screen mode`, the output is just the same with the real terminal. It means that any real-time application likes `top` will be captured as-is. Consecutive *read* from this VChannel instance may produce redundancy ouput.
Stop Screen Mode		Stops the `screen mode` and returns to `normal mode` In `screen mode`, *Write* does not return any thing and no output is logged. In `normal mode`, escape sequences are not processed by the virtual terminal.
Switch	*name*	Switches the current active channel to `name`. There only one active channel at any time Examples: | VChannel.*Switch* | vmx12 |
Write	*str_cmd=*, *str_wait=0s*, *start_screen_mode=False*	Sends `str_cmd` to the target node and return after `str_wait` time. If `start_screen_mode` is `True`, the channel will be shifted to `Screen Mode`. Default value of `screen_mode` is False. In `normal mode`, a `new line` char will be added automatically to the `str_cmd` and the command return the output it could get at that time from the terminal and also logs that to the log file. In `screen Mode`, if it is necessary you need to add the `new line` char by your own and the ouput is not be logged or returned from the keyword. Parameters: ▪ `str_cmd`: the command ▪ `str_wait`: time to wait after apply the command ▪ `start_screen_mode`: whether start the `screen mode` right after writes the command Special input likes Ctrl-C etc. could be used with global variable ${CTRL-<char>} Returns the output after writing the command the the channel. When *str_wait* is not *0s*, the keyword `read` and `return the output` after waiting *str_wait*. Otherwise, the keyword return without any output. **Notes:** This is a non-blocking command. Examples: \| VChannel.*Write* \| monitor interface traffic \| start_screen_mode=${TRUE} | | VChannel.*Write* | ${CTRL_C} | # simulates Ctrl-C |

Altogether 26 keywords.
Generated by Libdoc on 2018-09-23 09:18:05.

Logger

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

Provides advanced logging functions. Every Logger instance has one VChannel object and the is synchronized with the current active VChannel.

Notes: log file is not updated pararelly. Anytime a terminal is switched to, it will update its log file.

Shortcuts

Log · **L**og All · **S**witch

Keywords

Keyword	Arguments	Documentation

Log	*msg*, *with_time=False*, *mark=****	Inserts a message `msg` to the current *VChannel* log file. A default mark of `***` will be added at the beginning ant the end of this message.
		Example:
		`Logger.Log | START TRAFFIC FROM HERE | ${TRUE} |`
		`Logger.Log | START TRAFFIC FROM HERE | ${False} | ===`

Log All	*msg*, *with_time=False*, *mark=****	inserts a message `msg` to current **all** VChannel log files.
		A default `mark` of `***` and newline will be added at the beggining and the end of this message.
		Example:
		`Logger.Log All | START TRAFFIC FROM HERE | ${TRUE} |`
		`Logger.Log All | START TRAFFIC FROM HERE | ${TRUE} | ===`
		The log file will look likes this:
		`user@vmx12>`
		`*** 06:01PM on August 13, 2017: START TRAFFIC FROM HERE ***`
		`=== 06:01PM on August 13, 2017: START TRAFFIC FROM HERE ===`
		`configure`

Switch	*name*	Switches the current VChannel instance to `name`. `name` is the name of the VChannel (usually is the node name defined in the current active `local.yaml`). Example: `Logger.Switch | vmx11`

Altogether 3 keywords.
Generated by Libdoc on 2018-09-23 09:18:01.

OpticalSwitch

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A library provides control for L1 Optical Switch

Unlike other device, there is no *Switch* keywork with optical switch. Usually user only need to care about the interfaces not the ports of the switches.

Shortcuts

Add · **C**lear By File · **C**lose All · **C**onnect All · **D**elete · **G**et Connection Info · **L**oad From File · **S**ave To File

Keywords

Keyword	Arguments	Documentation
Add	*dev1*, *intf1*, *dev2*, *intf2*, *direction=bi*, *force=False*	Adds a connection. See details in each module help
Clear By File	*file_name=*, *comment=#*	Clears all x-connections defined in the *connection file* Default *connection file* is defined in `optic/connection` of `config/local.yaml`
Close All		Close all connections
Connect All		Connect to all L1 switch and read all neccesary information
Delete	*dev1*, *intf1*, *dev2*, *intf2*, *direction=bi*, *force=False*	Deletes a connection. See details in each module help
Get Connection Info	*dev*, *intf*	Returns connection information. See details in each module help.
Load From File	*file_name=*, *force=True*, *comment=#*	Loads the connection file and set the connections `filename` is the name of the connection file under the current config folder. If `filename` is empty, the value of `optic/connection` from `config/local.yaml` will be used. The `connection file` supports `jinja2` template language. Besides, `#` is the default comment char which could be changed The format of `connection file` follows: ▪ each connection is described by 1 line ▪ `source` and `destination` are separated by `` `- *or* > *, which mean* `bidirection` or `unidirection` (unidirection connects `source tx` to `dest rx` Connection file sample: `device1:port1 - device2:port2` `device1:port3 > device2:port` Examples: OpticalSwitch.*Load From File* OpticalSwitch.*Load From File* save1.conn
Save To File	*file_name*	Saves the current connection of all devices in this test. By default, all interfaces of the devices are save. If a connection file is given, only interfaces specified in the connection file are saved Examples: OpticalSwitch.*Save To File* save1.conn

Altogether 8 keywords.
Generated by Libdoc on 2018-09-23 09:18:04.

calient

Library scope:	global
Named arguments:	supported

Introduction

A library provides control for Calient Optical Switch

Table of Contents

Master file

The L1 switch provides a mechanism to remotely connect device interface. Each device interface has been wired to L1 switch already. The connection was described in the master file located specific by *calient-master-path* in the configuration file *renat/config/config.yaml*.

The master file includes several Calients in each tab. The column meaning and order is trivial.

Connection file Format

Keywords *Load From File*, *Clear By File* and *Save To File* use the x-connection file. X-connection files are text files and have the following format:

```
# this is the comment
device1,interface1,-,device2,interface2
device1,interface1,>,device2,interface2
```

The separator `-` means a bidirection connection and `>` means a unidirection connection. For a unidirection connection, `device1/interface1` TX will be connected to `device2/interface2` RX.

Note: The separator character must be surrounded by spaces or commas.

The connection file also support jinja2 template format. After the template is evaluated, comment could be used by `comment char`

There is no need to specify which L1 switch for the x-connection. The system will automatically find the appropriate switch.

Shortcuts

Add · **D**elete · **G**et Connection Info

Keywords

Keyword	Arguments	Documentation
Add	*self*, *dev1*, *intf1*, *dev2*, *intf2*, *direction=bi*, *force=False*	Adds x-connection between `dev1:intf1` and `dev2:intf2` `direction` is `bi` for bi-direction or `uni` for uni-direction. If `direction` is `uni`, the tx of `dev 1:port 1` will be connected to `dev 2:port 2`. With `force` mode, existed connection that use those ports will be deleted. Without `force` mode, an existed connection will make the keyword fails Examples: `OpticalSwitch.Add` `mx2008-31-33` `xe-3/0/0` `mx2008-31-33` `xe-3/0/1` `bi` `${TRUE}` **Note**: when `force` is `False` but the current ports is owned by the same connection endpoints, keyword will succeed. For a bidirection connection, 2 single uni-direction connection will be made instead of 1 bi-direction connection. This will make the link could be simulated tx/rx failure later.
Delete	*self*, *dev1*, *intf1*, *dev2*, *intf2*, *direction=bi*, *force=False*	Deletes the connection between `dev1:intf1 - dev2:intf2` Examples: `OpticalSwitch.Delete` `mx2008-31-33` `xe-3/0/1` `mx2008-31-33` `xe-3/0/1` `uni`
Get Connection Info	*self*, *dev*, *intf*	Returns information of the optic switch port that connected to `dev:intf`. The information is in jason format. Examples: `OpticalSwitch.Get Connection Info` `mx2008-31-33` `xe-3/0/1` return information looks like below: result = {u'outoc': u'NOHW', u'outopwdh': u'-20.0', u'inos': u'OOS', u'outalias': u'', u'inowner': u'TRANSIT', u'outopwct': u'-23.0', u'inpower': u'-3.4', u'inas': u'IS', u'outpower': u'-4.8', u'outas': u'OOS-NP', u'inopt': u'-17.0', u'inopth': u'13.0', u'incircuit': u'3.3.1>3.3.2', u'inalias': u'',

		u'inoc': u'NOHW', u'inoptc': u'-20.0', u'outos': u'OOS', u'port': u'3.3.1', u'outowner': u'NONE', u'outcircuit': u''}

Altogether 3 keywords.
Generated by Libdoc on 2018-09-23 09:18:09.

g4ntm

Library scope:	global
Named arguments:	supported

Introduction

A library provides control for Telescent Network Topology Management (NTM) robot patch.

Shortcuts

Add · **D**elete · **G**et Connection Info

Keywords

Keyword	Arguments	Documentation
Add	*self*, *dev1*, *intf1*, *dev2*, *intf2*, *direction=bi*, *force=False*	
Delete	*self*, *dev1*, *intf1*, *dev2*, *intf2*, *direction=bi*, *force=False*	Deletes the connection between `dev1:intf1 - dev2:intf2`
Get Connection Info	*self*, *dev*, *intf*	Returns information about the connection by router/interface

Altogether 3 keywords.
Generated by Libdoc on 2018-09-23 09:18:10.

Router

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A class provides keywords for router control. An instance of Router class automatically assigned methods of a VChannel class (**Note**: this is not an inheritance but rather 1-to-1 relation)

See VChannel for more details about *VChannel*.

Device's `type` is defined in master `device.yaml`. The system will load appropriate modules for each device.

Details about keywords provided by modules could be found in document of each module likes:

- Juniper module
- Cisco module
- GR module

Keywords provides by above module could be executed through *Xrun* keyword or directly called from `Router`. Examples:

Router.*Switch*	vmx12
Router.*Xrun*	Load Config
Router.*Load Config*	

Shortcuts

Follow Mib · **X**run

Keywords

Keyword	Arguments	Documentation
Follow Mib	*node_list*, *wait_time=10s*, *interval_time=5s*, *len=12*, *percentile=80*, *threshold=75*, *max_len=300*, *factor=1*	Waits until all the nodes defined in `node_list` become `stable`. Stableness is checked by SNMP polling result. The MIB list is define by `mib` in `node` section Parameter: ▪ `wait_time(1)`: the time before the evaluation starting ▪ `interval_time(2)`: interval between SNMP polling time ▪ `threshold`: below this value is evaluated as `stable` ▪ `len(3)`: the size of the evaluation window (number of values that are used in each valuation) ▪ `percentile`: real useful percentage of data (ignore top `100-percentile` percent) ▪ `max_len(4)`: maximum waiting `lend` for this checking time sequence: --(1)--|-(2)-|-----|-----|----|-----|-----| <-------(3)----------> poll poll <-------(3)----------> <----------(4)---------->
Xrun	*cmd*, **args*, ***kwargs*	Runs the vendor independent keywords. Parametes: ▪ `cmd`: a keyword ▪ `args`: other argumemts Examples: Router.*Xrun* | Flap Interface | ge-0/0/0 This keyword will then actually calling the correspond keyword for the device type.

Altogether 2 keywords.
Generated by Libdoc on 2018-09-23 09:18:02.

cisco

Library scope:	global
Named arguments:	supported

Introduction

Documentation for test library `cisco`.

Shortcuts

Get User · **G**et Version

Keywords

Keyword	Arguments	Documentation
Get User	*self*	Return the current login user
Get Version	*self*	return router version information

Altogether 2 keywords.
Generated by Libdoc on 2018-09-23 09:18:06.

gr

Library scope:	global
Named arguments:	supported

Introduction

Provides keywords for Hitachi GR platform

Shortcuts

Get Chassis Serial · **G**et Version

Keywords

Keyword	Arguments	Documentation
Get Chassis Serial	*self*	Returns the serial number of the chassis
Get Version	*self*	return router version information

Altogether 2 keywords.
Generated by Libdoc on 2018-09-23 09:18:06.

juniper

Library scope: global
Named arguments: supported

Introduction

Provides keywords for Juniper platform

Notes: Ignore the *self* parameters when using those keywords.

Shortcuts

Create Best Path Select Data · **D**isable Interface · **E**nable Interface · **F**lap Interface · **G**et Chassis Serial · **G**et Cli Mode · **G**et Config · **G**et Current Datetime · **G**et File · **G**et Intf Addr · **G**et Route Number · **G**et Version · **L**ink Status · **L**oad Config · **N**umber Of Bgp Neighbor · **N**umber Of Ospf Neighbor · **N**umber Of Ospf3 Neighbor

Keywords

Keyword	Arguments	Documentation
Create Best Path Select Data	*self*, *route_content*, *output_excel=best.xlsx*	Creates the matrix of best path selection Provides the test described in *smb://10.128.3.91/SharePoint01/31_VerificationRoom/31_13_検証環境セット/BGP-Best-Path-SelectionのAll-in-One設定_20161118改良/* The test uses predefined Ixia config and follows predefined steps
Disable Interface	*self*, *intf*	Disables an interface `intf`
Enable Interface	*self*, *intf*	Enables an interface `intf`
Flap Interface	*self*, *intf*, *time_str=10s*	Simulates an interface flap for interface `intf` Disables the interface and wait for a while before turning it up again
Get Chassis Serial	*self*	Returns the serial number of the chassis
Get Cli Mode	*self*	Returns current mode of the CLI. Return value is `config` for configuration mode or `command` for command mode
Get Config	*self*, *dst_name=*	Gets the current configuration file of the router to current `result` folder. Default `dst_name` is `juniper.conf.gz`

Get Current Datetime	*self*, *time_format=%H:%M:%S*, *delta_time=0s*, *dir=+*, ***kwargs*	Returns the current date time with vendor format `delta_time` will be added or subtracted to current time, default is `0s`
		`time_format` decides the time part of the output. Example result are :
		`May 24 04:14:25`
		`May  4 04:14:25`
		Note: The date part is padded by space, and the result is allways 15 characters

Get File	*self*, *src_file*, *dst_file=*	Gets a file from router ▪ `src_file` is a absolute path insides the router ▪ `dst_file` is a file name under `result` folder
Get Intf Addr	*self*, *intf_name*, *family=inet*	Returns the tuple of address and netmask of an interface `family` should be `inet` or `inet6` If the address is not set, `(",")` will be returned.
Get Route Number	*self*, *table=inet.0*	Returns number of active route in the `table` `table` could be `inet.0` or `inet.6`
Get Version	*self*	Returns router version information
Link Status	*self*, *if_name*	Returns link physical status as string (aka: "up down", "up up")
Load Config	*self*, *mode=set*, *config_file=*, *confirm=0s*, *vars=*, *err_match=(error:)*	Loads configuration to a router. Usable `mode` is `set`, `override`, `merge` and `replace` `set` mode uses configuration that contains `set` command. Mode `override`, `merge` and `replace` use ordinary JunOS configuration file with appropriate mode. `config_file` is a configuration file inside the `config` folder of the current test case. Config file could includes jinja2 template. The template will be evalued with *LOCAL*, *GLOBAL* and varibles defined by *vars*. The *vars* has the format: var1=value1,var2=value2 ... If the loading has no error that match the `error_match`, the configuration will be commited. The keywordl waits for `confirm` seconds before rollback the commited configuration. A zero value indicates an immediatly commit
Number Of Bgp Neighbor	*self*, *state=Established*	Returns number of BGP neighbor in `state` state

Number Of Ospf Neighbor	*self*, *state=Full*	Returns number of OPSF neighbors with status `state`
Number Of Ospf3 Neighbor	*self*, *state=Full*	Returns number of OPSFv3 neighbors with status `state`

Altogether 17 keywords.
Generated by Libdoc on 2018-09-23 09:18:05.

WebApp

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A library provides common keywords for web applications (aka Samurai, Arbor TMS)

The library utilize *Selenium2Library* and adds more functions to control Samurai application easily.

The *WebApp* uses the configuration in `local.yaml` in `webapp` section. The webapp device has following format:

```
<test node name>:
  device:    <device name>
  proxy:
    http:     <proxy for http>
    https:    <proxy for http>
    ssl:      <proxy for http>
```

Where `<device name>` is defined in master `device.yaml`, `proxy` section could be optional.

Samples:

```
...
webapp:
  samurai-1:
    device: samurai-b
    proxy:
      http:  10.128.8.210:8080
      ssl:   10.128.8.210:8080
      socks: 10.128.8.210:8080
  arbor-1:
    device: arbor-sp-a
    proxy:
      http:  10.128.8.210:8080
      ssl:   10.128.8.210:8080
      socks: 10.128.8.210:8080
...
```

Selenium2Library keywords still could be used along with this library like this:

Click Link	//a[contains(.,'ユーザ設定')]
Sleep	2s
Click Link	Home設定
Sleep	2s
Samurai.Capture Screenshot	

See Selenium2Library for more details.

The module *Samurai* and *Arbor* based on this module. See Arbor, Samurai for details about keywords of each application.

Shortcuts

Capture Screenshot · **R**eset Capture Counter · **S**et Ajax Wait · **S**et Capture Counter · **S**et Capture Format

Keywords

Keyword	Arguments	Documentation
Capture Screenshot	*filename=None*, *extra=*	Captures the current screen to file Using the internal counter for filename if `filename` is not specified. In this case, the filename is defined by a pre-set format. *Set Capture Format* could be used to change the current format. An extra information will be add to the filename if `extra` is defined Examples: Samurai.*Capture Screenshot* | | # samurai_0000000001.png Samurai.*Capture Screenshot* | extra=_list | # samurai_0000000002_*list*.png Arbor.*Capture Screenshot* | | # arbor_0000000001.png Arbor.*Capture Screenshot* | extra=_xxx | # arbor_0000000001_*xxx*.png Samurai.*Capture Screenshot* | filename=1111.png | # 1111.png
Reset Capture Counter		Resets the counter of the screen capture
Set Ajax Wait	*wait_time=2s*	Set the ajax wait time
Set Capture	*value=0*	Sets the counter of the screen capture to `value`

Counter. **Set Capture Format**	*format*	Sets the format for the screen capture file The format does not include the default prefix `.png` The default format is `<mod>_%010d`. `mod` could be `samurai` or `arbor` See https://docs.python.org/2/library/string.html#format-specification-mini-language for more details about the format string. Examples: Samurai.*Set Capture Format* ${case}_%010d # ${case} is a predefined variable

Altogether 5 keywords.
Generated by Libdoc on 2018-09-23 09:18:10.

Samurai

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A library provides functions to control Samurai application

The library utilize *Selenium2Library* and adds more functions to control Samurai application easily. Without other furthur mentions, all of the concepts of `user`, `user group` are Samurai concepts. By default, RENAT will try to connec to all Samurai nodes defined in active `local.yaml` at the beginning of the test and disconnect from them at the end of the test automatically. Usually user does not need to use `Connect All` and `Close` explicitly.

Currently, this module supposed that Samurai is used in Japanese locale. When Samurai module has error, it tried to make the last snapshot in `result/selenium-screenshot-x.png`. Checking this capture will help to understand the reason of the error.

Currently the module support Samurai 09/14/16

Some keywords of Samurai is using `xpath` to identify elements. See *Selenium2Library* for more details about xpath.

See WebApp for common keywords of web applications and how to configure the `local.yaml` file.

Selenium2Library keywords still could be used together within this library. See Selenium2Library for more details.

Shortcuts

Add Policy · **A**dd Policy Group · **A**dd User · **C**apture Screenshot · **C**hange Policy View Group · **C**lick All Elements · **C**lose · **C**lose All · **C**lose Window · **C**onnect · **C**onnect All · **D**elete Policy · **D**elete Policy Group · **D**elete User · **E**dit Mitigation Controller · **E**dit Policy · **G**et Mitigation List · **L**eft Menu · **L**ogin · **L**ogout · **M**ake Item Map · **R**econnect · **R**eset Capture Counter · **S**elect Items In Table · **S**elect Window · **S**et Ajax Wait · **S**et Capture Counter · **S**et Capture Format · **S**how Detail Mitigation · **S**how Policy Basic · **S**how Policy Mitigation · **S**how Policy Mo · **S**how Policy Monitor · **S**tart Mitigation · **S**top Mitigation · **S**witch

Keywords

<table>
<tr><th>Keyword</th><th>Arguments</th><th>Documentation</th></tr>
<tr><td>Add Policy</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Add Policy Group</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Add User</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Capture Screenshot</td><td>filename=None, extra=</td><td>Captures the current screen to file
Using the internal counter for filename if <code>filename</code> is not specified. In this case, the filename is defined by a pre-set format. Set Capture Format could be used to change the current format.
An extra information will be add to the filename if <code>extra</code> is defined
Examples:<table><tr><td>Samurai.Capture Screenshot</td><td></td><td># samurai_0000000001.png</td></tr><tr><td>Samurai.Capture Screenshot</td><td>extra=_list</td><td># samurai_0000000002_list.png</td></tr><tr><td>Arbor.Capture Screenshot</td><td></td><td># arbor_0000000001.png</td></tr><tr><td>Arbor.Capture Screenshot</td><td>extra=_xxx</td><td># arbor_0000000001_xxx.png</td></tr><tr><td>Samurai.Capture Screenshot</td><td>filename=1111.png</td><td># 1111.png</td></tr></table></td></tr>
<tr><td>Change Policy View Group</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Click All Elements</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Close</td><td></td><td>Closes the current active browser</td></tr>
<tr><td>Close All</td><td></td><td>Closes all current opened applications</td></tr>
<tr><td>Close Window</td><td></td><td>Closes the current window</td></tr>
<tr><td>Connect</td><td>app, name</td><td>Opens a web browser and connects to application and assigns a <code>name</code>.
If not defined in <code>local.yaml</code> those following key will have defaut values:<table><tr><td>browser</td><td>firefox</td><td>optional</td></tr><tr><td>login_url</td><td>/</td><td>optiona</td></tr><tr><td>proxy:</td><td></td><td>optional</td></tr><tr><td>http: 10.128.8.210:8080</td><td>optional</td><td></td></tr><tr><td>ssl: 10.128.8.210:8080</td><td>optional</td><td></td></tr><tr><td>socks: 10.128.8.210:8080</td><td>optional</td><td></td></tr><tr><td>profile_dir</td><td>./config/samurai.profile</td><td>optional</td></tr></table></td></tr>
<tr><td>Connect All</td><td></td><td>Connects to all applications defined in <code>local.yaml</code>
The name of the connection will be the same of the webapp name</td></tr>
</table>

Delete Policy	**args*, ***kwargs*	
Delete Policy Group	**args*, ***kwargs*	
Delete User	**args*, ***kwargs*	
Edit Mitigation Controller	**args*, ***kwargs*	
Edit Policy	**args*, ***kwargs*	
Get Mitigation List	**args*, ***kwargs*	
Left Menu	**args*, ***kwargs*	
Login		Logs-in into the application User and password is set by the template and authentication methods in the master files
Logout	**args*, ***kwargs*	
Make Item Map	**args*, ***kwargs*	
Reconnect		Reconnects to the server
Reset Capture Counter		Resets the counter of the screen capture
Select Items In Table	**args*, ***kwargs*	
Select Window	**args*, ***kwargs*	
Set Ajax Wait	*wait_time=2s*	Set the ajax wait time
Set Capture Counter	*value=0*	Sets the counter of the screen capture to `value`
Set Capture Format	*format*	Sets the format for the screen capture file The format does not include the default prefix `.png` The default format is `<mod>_%010d`. `mod` could be `samurai` or `arbor` See https://docs.python.org/2/library/string.html#format-specification-mini-language for more details about the format string. Examples: Samurai.*Set Capture Format* ${case}_%010d # ${case} is a predefined variable
Show Detail Mitigation	**args*, ***kwargs*	
Show Policy Basic	**args*, ***kwargs*	
Show Policy Mitigation	**args*, ***kwargs*	
Show Policy Mo	**args*, ***kwargs*	
Show Policy Monitor	**args*, ***kwargs*	
Start Mitigation	**args*, ***kwargs*	
Stop Mitigation	**args*, ***kwargs*	
Switch	*name*	Switches the current browser to `name`

Altogether 36 keywords.
Generated by Libdoc on 2018-09-23 09:18:11.

Arbor

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A library provides functions to control Arbor application

The library utilize *Selenium2Library* and adds more functions to control Arbor application easily.

See WebApp for common keywords of web applications.

Selenium2Library keywords still could be used along with this library. See Selenium2Library for more details.

Shortcuts

Capture Screenshot · **C**lose · **C**lose All · **C**onnect · **C**onnect All · **D**etail First Mitigation · **L**ogin · **L**ogout · **M**enu · **R**econnect · **R**eset Capture Counter · **S**et Ajax Wait · **S**et Capture Counter · **S**et Capture Format · **S**how All Mitigations · **S**how Detail Countermeasure · **S**how Detail First Mitigation · **S**how Detail Mitigation · **S**how Detail Mitigation With Order · **S**witch

Keywords

<table>
<tr><th>Keyword</th><th>Arguments</th><th>Documentation</th></tr>
<tr><td>Capture Screenshot</td><td>filename=None, extra=</td><td>Captures the current screen to file

Using the internal counter for filename if <code>filename</code> is not specified. In this case, the filename is defined by a pre-set format. Set Capture Format could be used to change the current format.

An extra information will be add to the filename if <code>extra</code> is defined

Examples:<table><tr><td>Samurai.Capture Screenshot</td><td></td><td># samurai_0000000001.png</td></tr><tr><td>Samurai.Capture Screenshot</td><td>extra=_list</td><td># samurai_0000000002_list.png</td></tr><tr><td>Arbor.Capture Screenshot</td><td></td><td># arbor_0000000001.png</td></tr><tr><td>Arbor.Capture Screenshot</td><td>extra=_xxx</td><td># arbor_0000000001_xxx.png</td></tr><tr><td>Samurai.Capture Screenshot</td><td>filename=1111.png</td><td># 1111.png</td></tr></table></td></tr>
<tr><td>Close</td><td></td><td>Closes the current active browser</td></tr>
<tr><td>Close All</td><td></td><td>Closes all current opened applications</td></tr>
<tr><td>Connect</td><td>app, name</td><td>Opens a web browser and connects to application and assigns a <code>name</code>.

Extra information could be added to the <code>webapp</code> sections likes <code>login_url</code>, <code>browser</code> or <code>profile_dir</code>. Default values are:<table><tr><td>browser</td><td>firefox</td></tr><tr><td>login_url</td><td>/</td></tr><tr><td>profile_dir</td><td>./config/samurai.profile</td></tr></table></td></tr>
<tr><td>Connect All</td><td></td><td>Connects to all applications defined in <code>local.yaml</code>

The name of the connection will be the same of the webapp name</td></tr>
<tr><td>Detail First Mitigation</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Login</td><td></td><td>Logged-into the Arbor application</td></tr>
<tr><td>Logout</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Menu</td><td>*args, **kwargs</td><td></td></tr>
<tr><td>Reconnect</td><td></td><td>Reconnect to server if necessary</td></tr>
<tr><td>Reset Capture Counter</td><td></td><td>Resets the counter of the screen capture</td></tr>
<tr><td>Set Ajax Wait</td><td>wait_time=2s</td><td>Set the ajax wait time</td></tr>
<tr><td>Set Capture Counter</td><td>value=0</td><td>Sets the counter of the screen capture to <code>value</code></td></tr>
<tr><td>Set Capture Format</td><td>format</td><td>Sets the format for the screen capture file

The format does not include the default prefix <code>.png</code> The default format is <code><mod>_%010d</code>. <code>mod</code> could be <code>samurai</code> or <code>arbor</code>

See https://docs.python.org/2/library/string.html#format-specification-mini-language for more details about the format string.

Examples:<table><tr><td>Samurai.Set Capture Format</td><td>${case}_%010d</td><td># ${case} is a predefined variable</td></tr></table></td></tr>
</table>

Show All Mitigations	**args*, ***kwargs*	
Show Detail Countermeasure	**args*, ***kwargs*	
Show Detail First Mitigation	**args*, ***kwargs*	
Show Detail Mitigation	**args*, ***kwargs*	
Show Detail Mitigation With Order	**args*, ***kwargs*	
Switch	*name*	Switches the current browser to `name`

Altogether 20 keywords.
Generated by Libdoc on 2018-09-23 09:18:12.

Tester

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A class provides keywords for controlling testers and traffic generators.

It could load predefined traffic file, manipulate traffic items, start and stop traffic flows. It also could generate traffic reports and support QuickTest for IxNetwork.

Tester information is stored in the active `local.yaml` likes this:

```
tester:
  tester01:
    device: ixnet03_8009
    config: vmx_20161129.ixncfg
    real_port:
      - description: to egde router
```

chassis: 10.128.32.71

```
        card: 6
        port: 11
      - description: to backbone router
```

chassis: 10.128.32.71

```
        card: 6
        port: 9
```

where `device` is the tester defined in the master `device.yaml` file. If `real_port` does not exist, port remapping will not take place. Otherwise, port remapping will use the `real_port` information to reassign all existed ports and map to Ixia ports.

In this case, the order will be the order when user created the port in Ixia GUI.

Note: User can always confirm the created order by `clear sorting` in Ixia GUI.

Examples:

Tester.*Connect All*	
Tester.*Switch*	tester01
Tester.*Load And Start Traffic*	
Sleep	30s
Tester.*Stop Traffic*	

Time format used in this module is same with `time string` format of Robot Framework. For more details about this, see DateTime library of Robot Framework.

Note: See IxNet module, IxLoad module for details about keyword of each module.

Shortcuts

Close All · **C**onnect · **C**onnect All · **S**witch

Keywords

Keyword	Arguments	Documentation
Close All		Closes all connections
Connect	*name*	Connect to the tester `name`
Connect All		Connects to all testers
Switch	*name*	Switchs the current tester to `name`

Altogether 4 keywords.
Generated by Libdoc on 2018-09-23 09:18:03.

ixload

Library scope:	global
Named arguments:	supported

Introduction

provides functions for IxLoad

To use IxLoad module, a IxLoad TCL server should be started properly.

RENAT runs a virtual IxLoad client locally in the background that connects to a Windows App server. Keywords from test case will send control messages to the client, which in turn will control the test ports.

Different to IxNetwork, an IxLoad test case usually stops within predefined time before `Stop Traffic` was called.

Notes: Ignore the *self* parameters when using those keywords.

Shortcuts

Close · **C**ollect Data · **G**et Test Report · **L**oad Config · **L**oad Traffic · **S**tart Traffic · **S**top Traffic

Keywords

Keyword	Arguments	Documentation
Close	*self*	Disconnects the current tester client
Collect Data	*self*, *prefix=*, *more_file=*, *ignore_not_found=True*	Collects all result data and save them to the current active `result` folder A `prefix` will be automatically added to the file names. Currently the follow data will be downloaded to the local machine ▪ HTTP_Server.csv ▪ HTTP Client.csv ▪ HTTP Client - Per URL.csv ▪ HTTP Server - Per URL.csv ▪ L2-3 Stats for Client Ports.csv ▪ L2-3 Stats for Server Ports.csv ▪ L2-3 Throughput Stats.csv ▪ Port CPU Statistics.csv Extra files could be add by `more_file` which is a comma separated filename string When `ignore_not_found` is True, the keyword will not terminate even when the expected file is not found.
Get Test Report	*self*, *prefix=*	Get the test report(PDF) and put it into the active result folder
Load Config	*self*, *config_name=*	Loads the test traffic defined by `config_name` `file_path` is the path of the test file on the **remote** App server A path to a remote network drive could be use to load a config file on Renat server.
Load Traffic	*self*, *file_path*	
Start Traffic	*self*	Starts the test traffic
Stop Traffic	*self*	Stops the current running test Returns the elapsed time in seconds

Altogether 7 keywords.
Generated by Libdoc on 2018-09-23 09:18:07.

ixnet

Library scope:	global
Named arguments:	supported

Introduction

provides functions for IxNetwork

To use IxNetwork module, a IxNetwork TCL server should be started properly.

RENAT will connect to the App server and control the test ports. Test files and result will be insde the RENAT server.

In order to run RENAT test case with *IxLoad*, the *TCLServer* must be activated with *Administrator* privileges on the Ixia App server.

Notes: Ignore the *self* parameters when using those keywords.

Shortcuts

Add Port · **A**dd Quicktest · **A**pply Traffic · **C**hange Frame Rate · **C**hange Frame Rate Dynamic · **C**hange Frame Size · **C**lose · **C**ollect All Data · **C**ollect Data · **G**et All Test Result · **G**et Quicktest List · **G**et Quicktest Result · **G**et Quicktest Result Path · **G**et Test Composer Result · **G**et Test Report · **G**et Test Result · **L**oad And Start Traffic · **L**oad Config · **L**oad Traffic · **L**oss From File · **P**ing · **R**egenerate · **R**eset Config · **R**un Quicktest · **S**et All Traffic Item · **S**et Bgp Items · **S**et Bgp Neighbor · **S**et Capture Port · **S**et Traffic Item · **S**hould Be Pingable · **S**tart Capture · **S**tart Protocol · **S**tart Test Composer · **S**tart Traffic · **S**top All Protocols · **S**top And Save Capture · **S**top Quicktest · **S**top Test Composer · **S**top Traffic · **W**ait Until Connected

Keywords

Keyword	Arguments	Documentation

Add Port	*self*, *force=True*, *time_out=2m*, *learn_time=2m*	Add ports using the information from active local config
		▪ `time_out` is the wait time until port is connected (default is 2m)
		▪ `learn_time` is the time waiting for arp to be learned (default is 2m)
		Sample of local config |tester:
		`tester:`
		`  device: ixnet03_8009`
		`  config: quicktest.ixncfg`
		`  real_port:`
		`  - chassis: 10.128.4.41`
		`  card: 4`
		`  port: 3`
		`  ip: 10.100.11.2`
		`  mask: 24`
		`  gw: 10.100.11.1`
		`  - chassis: 10.128.4.41`
		`  card: 4`
		port: 4
		`  ip: 10.100.14.2`
		`  mask: 24`
		`  gw: 10.100.14.1`

Add Quicktest	*self*, *name*, *test_type=rfc2544throughput*, *tx_mode=interleaved*, *clear_all=True*	Create a new Quicktest with default value Type could be one of following: `rfc2544throughput`, `rfc2544frameLoss`, `rfc2544back2back`. Use Tester.*Load Config* to load a customized quicktest When `clear_all` is True, any existed quicktests will be cleared. Transmit mode `tx_mode` takes following values: `interleaved` (default) or `sequential`. The mode should be identical with the transmit mod of the ports. **Notes**: The keyword **does not** create necessary ports. It should be used with a existed configuration by Tester.*Load Config* or Tester.*Add Port* keyword.
Apply Traffic	*self*, *refresh=True*	Applies the current traffic configuration `refresh`: Refreshed the learned information before apply the traffic or not **Note:** This is a blocking command
Change Frame Rate	*self*, *value*, *pattern=.**	Changes the frame rate Parameter: ▪ `value`: value to set. Depends on the current configuration, this could be `percent line rate` or `bit per second` etc. ▪ traffic_pattern*: a regular expression to identify traffic item name, default is everything `.**
Change Frame Rate Dynamic	*self*, *value*, *pattern=.**	Changes the traffic flow rate on-fly No need to stop the running traffic to change the rate Parameter: ▪ `value`: value to set. Depend on the current configuration, this could be `percent line rate` or `bit per second` etc. ▪ `pattern`: a regular expression to identify traffic item name, default is everything `.*`
Change Frame	*self*, *type*, *value*, *pattern=.**	Changes the frame size

Size		Parameter: ▪ `type` : could be `fixed size` , `increment_from` `increment_step` or `increment_to` ▪ `value` : value to set ▪ `traffic_pattern` : a regular expression to identify traffic item name, default is everything `.*`
Close	*self*	Disconnects the current tester client
Collect All Data	*self*, *prefix=stat_*	Deprecated. Use
Collect Data	*self*, *view*, *prefix=stat_*	Depricated. Use *Get Test Result*
Get All Test Result	*self*, *prefix=stat_*	Collects all Ixia traffic data after traffic is stopped. Results are CSV files that are stored in `result` folder. The prefix `prefix` is appended to the original view name
Get Quicktest List	*self*	Returns current loaded Quicktest list
Get Quicktest Result	*self*, *test_index=-1*, *prefix=*, *enable_all=True*	Get the result.csv file from the latest Quicktests `test_index` is a index of the current Quicktest. `-1` means that last one.
Get Quicktest Result Path	*self*, *test_index=-1*	Returns the path of the newest run of a Quicktest `test_index` is a index of the current Quicktest. `-1` means that last one.
Get Test Composer Result	*self*, *result_file=composer.log*	Get the result of test composer script
Get Test Report	*self*, *local_name=ixnet_report.pdf*, *enable_all=True*	Generates and get report of the current active test in PDF format `local_name` : name of the report on local machine. Default is `ixnet_report.pdf`
Get Test Result	*self*, *view*, *prefix=stat_*	Collects traffic data of a `view` and export to a CSV file in `result` folder Currently, supported views are: `Port Statistics` , `Global Protocol Statistics` , `BGP Aggregated Statistics` , `BGP Aggregated State Counts` , `OSPF Aggregated Statistics` , `OSPF Aggregated State Counts` , `OSPFv3 Aggregated Statistics` , `OSPFv3 Aggregated State Counts` , `L2-L3 Test Summary Statistics` , `Flow Statistics` , `Flow Detective` , `Data Plane Port Statistics` , `User Defined Statistics` , `Traffic Item Statistics` Result were store as CSV files in `result` folder. If there is no valid data, view will be silently ignored The prefix `prefix` is appended to the view name for the CSV file.
Load And Start Traffic	*self*, *wait_time1=10s*, *wait_time2=10s*	Combines *Load Traffic* and *Start Traffic* to one keyword.
Load Config	*self*, *config_name=*, *wait_time=2m*, *wait_time2=2m*, *apply=True*, *protocol=True*, *force=True*, *wait_time3=30s*	loads traffic configuration, applies and start protocol if necessary. The config file name was defined in the *`local.yaml* which is a Ixia Network configuration file and located in the *config* folder of the test. The keyword remap the vports to real port when data is specified in the local configuration file. For some reasons, the txMode is cleared when remapping happens. Use `tx_mode` to set the TxMode of the remapped ports. Parameters: ▪ `apply` : applies traffic when `True` otherwise ▪ `protocol` : starts all protocols when `True` otherwise ▪ `force` : force to reclaim the ports when `True` otherwise ▪ `wait_time` : wait time after applying protocols ▪ `wait_time2` : maximum wait time befor all ports become available. In common case, this is calculated automatically so user does not need to change this value. ▪ `wait_time3` : default waiting time after config file is loaded (30s) More information about ports could be define in `real_port` section like this: `# tester information` `tester:` `tester:` `device: ixnet03_8009` `config: bgp.ixncfg` `real-port:` `- chassis: 10.128.4.41` `card: 4` `port: 7` `media: fiber` `tx_mode: interleaved` Configurable port parameters ares: ▪ `tx_mode` : *sequential* or *interleaved*(default) ▪ `media` : *copper* or *fiber* (**Note**: no default value) See Common for more details about the yaml configuration files.
Load Traffic	*self*, *wait_time=2m*, *wait_time2=2m*, *apply=True*, *protocol=True*, *force=True*, *tx_mode=interleaved*	
Loss From File	*self*,	Returns `packet loss` by miliseconds and delta frame.

	file_name=Flow_Statistics.csv, *tx_frame_i=3*, *frame_delta_i=5*, *time1_i=23*, *time2_i=24*	The calculation should be performed when traffic is stopped. The calculation supposed traffic is configured by frame per second

Ping	*self*, *dst_ip*, *src_port_index=0*, *src_intf_index=0*	Ping from Ixia to `dst_ip`
		The keyword return the output string as it is. The return could be
		`- Port <portName>: ping failed: port not assigned`
		`- Response received from <sourceIp>/unknown . Sequence Number <sequenceNumber>`
		`- Ping request to <destinationIp>/unknown ip failed: <GenericPingError>/<error>: <genericError>unknown reason`
		`- Error: Couldn't find any source interface for Send Ping to <destinationIp> on <portName> Id <id>`
		`- Error: Couldn't find any source IP for Send Ping to <destinationIp> on <portName> Id <id>`
		Parameters:
		▪ src_port_index: index of Ixia port (starts from 0)
		▪ src_intf_index: index of interface insides the port (starts from 0)
		Examples:
		Tester.*Ping* 1.1.1.1 0 0
		Tester.*Ping* 1.1.1.1

Regenerate	*self*	Regenerates **all** flow of current traffic items
Reset Config	*self*	Clears current config and creates new blank config
Run Quicktest	*self*, *test_index=0*, *wait_until_finish=True*	Runs the Quicktest and wait until it finishes **Warning**: it could take a long time to finish a quicktest
Set All Traffic Item	*self*, *enabled=True*	Enables/Disables **all** traffic items at once
Set Bgp Items	*self*, *port_index*, *neighbor_index*, *route_range_index*, *is_enable*	Enables/Disables BGP entry by a set of port,neighbor,route_range index Parameters: ▪ `port_index`: index of the port ▪ `neighbor_index`: index of the neighbor or `*` ▪ `route_range_index`: index of the route range or `*` ▪ `is_enable`: ${TRUE} or ${FALSE} Note Examples: Tester.*Set BGP Items* 0 * * ${FALSE} Tester.*Set BGP Items* 0 * * ${TRUE}
Set Bgp Neighbor	*self*, **indexes*, ***kwargs*	Enables/Disables BGP entry by neighbor index `kwargs` contains following parameters: ▪ indexes: is a list of index of BGP neighbor (index is started from zero) ▪ vport_index: is the target vport index ▪ enabled: TRUE or FALSE Examples: Tester.*Set BGP Item* 0 1 vport_index=0 enabled=${FALSE} Tester.*Set BGP Item* 0 1 vport_index=1 enabled=${TRUE}
Set Capture Port	*self*, *data_mode=True*, *control_mode=True*, *port_index=0*	Capture packets for follow `port` `port_index`: is a index of current test port (start from 0) `data_mode`: capture data packets and save in <intf>_HW.cap file `control_mode`: capture controls packets and save in <intf>_SW.cap file **Note**: `control_mode` saves all control packets and `data_mode` only saves data packets. **Note**: `control_mode` saves all control packets and `data_mode` only saves data packet Examples: Tester.*Set Capture Port* 0 Tester.*Set Capture Port* control_mode=${TRUE} 0 1
Set Traffic Item	*self*, **items*, ***kwargs*	Enables/Disables some traffic items `items` Parameters: ▪ `items`: a list of Ixia traffic item name ▪ `enabled`: False or True ,the mode to set traffic item to, default is `True` (`enabled`) **Note**: traffic item could be specified by ::<num> format. In this case the `num` is the order of traffic item count from zero. Returns `True` if all items are set coordinately or otherwise Examples: Set Traffic Item Traffic Item 1 Traffic Item 2 Set Traffic Item @{item_list} Set Traffic Item Traffic Item 1 enabled=${FALSE}
Should Be Pingable	*self*, *dst_ip*, *src_port_index=0*, *src_intf_index=0*	Ping from Ixia and raise an error if ping fails The keyword return *True* if succeeds

Start Capture	*self*, *wait_time=30s*	Start packet capture Target ports are set by the configuration file or by [Set Capture] keyword
Start Protocol	*self*, *wait_time=1m*	Starts all protocols and wait for `wait_time` Default `wait_time` is 1 minute. Make sure `wait_time` is big engouh to start all protocols.
Start Test Composer	*self*, *script_name=Main_Procedure*, *run_num=1*, *wait_for_test=True*, *parameter=*, *wait=10s*	Run a test composer script. The test composer script should be included in an Ixia Network configuration file and loaded properly with *Load Config* Parameters: ▪ `script_name` is the name of the script to run. Default value is `Main_Procedure`. ▪ `wait_for_test`: if `${TRUE}` then wait until the script finishes. ▪ `parameter`: parameter that is passed to the script. Parameter could be in 2 formats: `{{VAR1 VALUE1} {VAR2 VALUE2}}` or simply as `VALUE1 VALUE2`. The script must prepare *VAR1* and *VAR2* properly by *Test parameter*. See Ixia Network anout composer script for more details. ▪ `wait`: wait time before go to next keyword Examples: Tester.*Start Test Composer* | parameter=XXX YYY Tester.*Get Test Composer Result* | result_file=script1.log Tester.*Start Test Composer* | parameter={{VAR1 AAA} {VAR2 BBB}} Tester.*Get Test Composer Result* | result_file=script1.log
Start Traffic	*self*, *wait_time=30s*	Starts the current traffic settiing and wait for `wait_time`. **Note:** This is a asynchronus action. After called, the keyword finishes immediatly but it will take a while before traffic starts By default the keyword will wait for 30 seconds.
Stop All Protocols	*self*, *wait_time=30s*	Stop all running protocols
Stop And Save Capture	*self*, *prefix=*, *wait_until_finish=True*, *monitor_interval=5s*	Stop current capture and save the resuls to folder specified by `path` Captured files will be saved in current `result` folder with `prefix` appended in their names. Examples: Tester.*Start Capture* | Sleep | 10s Tester.*Stop And Save Capture* | ${RESULT_FOLDER}/capture.zip
Stop Quicktest	*self*, *test_index=0*	Stops a running test
Stop Test Composer	*self*, *wait=10s*	Stop a running composer Do nothing when a test composer has already stopped or no composer has been prepared.
Stop Traffic	*self*, *stop_protocol=False*, *wait_time=10s*	Stops the current traffic and wait for `wait_time` Parameters: ▪ stop_protocol: if `True` also stops all running protocols ▪ wait_time: time to wait after apply the command
Wait Until Connected	*self*, *timeout_str=5m*	Waits until ports become enabled and connected

Altogether 40 keywords.
Generated by Libdoc on 2018-09-23 09:18:06.

ixbps

Library scope:	global
Named arguments:	supported

Introduction

provides functions for Ixia Breaking Point

Breaking Point testers setting is set by `tester` section in `local.yaml`. Users need to specify the physical ports used by the test by its card and port number.

Setting example:

```
ixbps01:
  device: ixbps01
  config: test.bpt
  real-port:
    - card:  1
      port:  0
    - card:  1
      port:  1
```

Notes: Ignore the *self* parameters when using those keywords. The module requires Breaking Point Python library installedd properly to work.

Shortcuts

Cleanup Tests · **C**lose · **G**et Card Config · **G**et Card Mode · **G**et Test Report · **L**oad Config · **S**et Card Config · **S**tart Test · **S**top Test · **W**ait Until Finish

Keywords

Keyword	Arguments	Documentation
Cleanup Tests	*self*	Cleans up running test and release their ports
Close	*self*	Closes the connection to the BP box
Get Card Config	*self*, *slot_num*	Get card configuration for *slot_num* Parameter: ▪ *slot_num* is *all* or an integer start from 1 Result is a json formatted string contains the informaition for specific slot or *all* slots.
Get Card Mode	*self*, *slot_num*	Gets the *mode* of a specific slot
Get Test Report	*self*, *report_name=result*, *format=csv*	Gets and saves the test report to local disk The report will be in PDF format. If `export_csv` is `True` then test results are also exported by CSV format.
Load Config	*self*, *config_name=*, *force=True*	Loads test configuration config_name is defined in `local.yaml` or specific by user in the main scenario.
Set Card Config	*self*, *slot*, *action=mode*, *param=ixload*	Changes the configuration of BPS card Parameters: ▪ slot: slot number ▪ action: `mode` or `perfacc` ▪ param: depending on *action* Values of *param*: ▪ if *action* is `mode` then *param* should be `ixload`, `bp` or `bpl23` (BreakingPoint L2/3) ▪ if *action* is `perfacc` then *param* should be ${TRUE} or ${FALSE}
Start Test	*self*, *test_name=None*, *force=True*	Starts a test by its name The system automatically reserve the ports defined in `local.yaml`. The reserved ports are released when the test is stopped `test_name` is the name of the testmodel saved in the configuration. If `test_name` is *None*, the 1st testmodel will be used. If `force` is *True* then all running tests and their reserved ports will be released.
Stop Test	*self*, *wait=5s*	Stops a running test
Wait Until Finish	*self*, *interval=30s*, *timeout=30m*, *verbose=False*	Waits until the test finished or timeout **Notes**: This is a blocking keyword

Altogether 10 keywords.
Generated by Libdoc on 2018-09-23 09:18:08.

Hypervisor

Library version:	RENAT 0.1.10
Library scope:	test suite
Named arguments:	supported

Introduction

A module controls Hypervisors

A hypervisor is declared in `local.yaml` like this:

```
# esxi information
hypervisor:
  esxi-server:
    device: esxi-3-15
```

Notes: Currently support VMWare(Esxi) only

Shortcuts

Capture Mks Screenshot · **C**lose · **C**lose All · **C**onnect · **C**onnect All · **G**et Mks Ticket · **G**et Vm Id · **G**et Vm List · **G**et Vm Power State · **O**pen Console · **P**ower Off · **P**ower On · **R**eset Capture Counter · **S**end Mks Cmd · **S**end Mks Key · **S**et Capture Format · **S**witch · **X**run

Keywords

Keyword	Arguments	Documentation
Capture Mks Screenshot	**args*, ***kwargs*	
Close		Closes and disconnects from a hypervisor
Close All		Closes all current opend hypervisor connection
Connect	*hyper*, *name*	Connects to a Hypervisor
Connect All	*prefix=*	Connect to **all** hypervisor listed in local config yaml
Get Mks Ticket	**args*, ***kwargs*	
Get Vm Id	**args*, ***kwargs*	
Get Vm List	**args*, ***kwargs*	
Get Vm Power State	**args*, ***kwargs*	
Open Console	**args*, ***kwargs*	
Power Off	**args*, ***kwargs*	
Power On	**args*, ***kwargs*	
Reset Capture Counter	**args*, ***kwargs*	
Send Mks Cmd	**args*, ***kwargs*	
Send Mks Key	**args*, ***kwargs*	
Set Capture Format	**args*, ***kwargs*	
Switch	*name*	Switch the current hypervisor to a new one
Xrun	*cmd*, **args*, ***kwargs*	

Altogether 18 keywords.
Generated by Libdoc on 2018-09-23 09:18:13.

vmware

Library scope: global
Named arguments: supported

Introduction

provides function for VMware ESXI

Athough this module provides some keywords for interact with the console (WebConsole), they only suppport very primitive actions. Do not use this to accomplish complex tasks. Instead, using VChannel through a SSH/Telnet channel.

Shortcuts

Capture Mks Screenshot · **G**et Mks Ticket · **G**et Vm Id · **G**et Vm List · **G**et Vm Power State · **O**pen Console · **P**ower Off · **P**ower On · **R**eset Capture Counter · **S**end Mks Cmd · **S**end Mks Key · **S**et Capture Format

Keywords

Keyword	Arguments	Documentation
Capture Mks Screenshot	*self*, *filename=None*, *extra=*	Captures the current web console to *filename* If *filename* is `None`, the captured filename will be decided by the current `format` with an auto-increment counter and a `extra` at the end. Example: *Hypervisor*.Capture MKS Screenshot | # will create a file console_0000000001.png | *Hypervisor*.Capture MKS Screenshot | xxx.png | # will create a file xxx.png *Hypervisor*.Capture MKS Screenshot | # will create a file console_0000000002.png |
Get Mks Ticket	*self*, *vm_name*	Returns a MSK ticket for WebConsole
Get Vm Id	*self*, *vm_name*	Returns a VMID of a VM
Get Vm List	*self*	Returns current VM name list of the hypervisor
Get Vm Power State	*self*, *vm_name*	Get vm power status Return `on` of `off`
Open Console	*self*, *vm_name*	Opens a web console for a VM *vm_name* Examples: Hypervisor.*Set Capture Format* | console_%010d | *Open Console* | ${VM_NAME} | Hypervisor.*Capture MKS Screenshot* | *Send MKS Cmd* | root | *Send MKS Cmd* | password | wait=10s *Send MKS Cmd* | ls | Hypervisor.*Capture MKS Screenshot* | *Send MKS Key* | ${CTRL_L} | *Send MKS Cmd* | whoami | Hypervisor.Capture MKS Screenshot` |
Power Off	*self*, *vm_name*, *graceful=True*	Shutdowns a VM If *graceful* is True, a graceful shutdown is tried before a power off. **Note**: if VMware tools is not install on the VM, graceful shutdown is not available
Power On	*self*, *vm_name*	Power on a VM
Reset Capture Counter	*self*	
Send Mks Cmd	*self*, *cmd*, *wait=2s*	Sends command to current web console and wait for a while By default, *wait* time is `2s` and the keyword will automaticall add a `Newline` char after sending the *cmd*
Send Mks Key	*self*, *key*, *wait=1s*	Sends key strokes to current web console Special Ctrl char could be use by `${CTRL_A}` to `${CTRL_Z}` Examples: *Send MKS Key* | ${CTRL_L} |
Set Capture Format	*self*, *format*	Set console capture format Initialized format is `'vmware_%010d'`

Altogether 12 keywords.
Generated by Libdoc on 2018-09-23 09:18:14.