CUE Organic Actions API Document

This API manual will introduce the CUE Organic Actions mapping with the related lib and program code API and give a simple example about how to use it.

```
# Version:      v_0.0.1
# Created:      2024/05/03
# Copyright:
# License:      MIT License
```

Network Activities Module

Action 01 : Ping targets sequence or parallel

Source code module: `pingActor.py`

API Usage :

Init the ping actor obj:

```
actor = pingActor({'127.0.0.1':10},parallel=True, Log=None, showConsole=False)
```

Pass in parameters:

- `config` (dict): Ping destination config dictionary or the dictionary json file path. json item format `'dest ip/url': int(pingtime)`
- `parallel` (bool, optional): Ping the dest in sequential if val==False or parallel threading (multi-thread) if val==True. Defaults to False.
- `Log` (*type*, optional): A logger object used to log the result to local if necessary. Defaults to None.
- `showConsole` (bool, optional): Flag to identify whether pop-up the OS console. Defaults to False.

Start ping action:

```
result = actor.runPing()
print(result)
```

Pass in parameters: `None`

Action 02: Capture webpage screen shot

Source code module: `WebScreenShoter.py`

API Usage :

Init the pyWebscreenshoter obj:

```
capturer = webScreenShoter()
```

Pass in parameters: `None`

Start capture:

```
capturer.getScreenShot(url, outputFolder, driverMode=driverMode)
```

Pass in parameters:

- `urlList` (list/tuple): url string list.
- `outDirPath` (str): output directory path.
- `driverMode` (int, optional): driver selection. Defaults to QT_DRIVER.

Action 03: Download all the contents in a web page

Source code module: `webDownloader.py`

Function: provide API to download the webpage components: html file, image file, javascript file, href link file, host SSL certificate and xml file based on the input url.

API Usage :

Init the webDownloader obj:

```
downloader = webDownloader(imgFlg=True, linkFlg=True, scriptFlg=True,
caFlg=True)
```

Pass in parameters:

- `imgFlg` (bool, optional): flag to identify whether download image. Defaults to True.
- `linkFlg` (bool, optional): flag to identify whether download all the hyper link contents. Defaults to True.
- `scriptFlg` (bool, optional): flag to identify whether download script. Defaults to True.
- `caFlg` (bool, optional): flag to identify whether download certificate. Defaults to True.

Download contents:

```
downloader.downloadWebContents(urlStr, downloadFolderPath)
```

Pass in parameters:

- `urlStr` (str): url string.
- `outputDirPath` (str): output folder path.

Action 04: Connect to FTP server

Source code module: `networkServiceProber.py`

API Usage:

Connect to FTP server:

```
checkFtpConn(self, target, loginConfig=None, timeout=3)
```

Pass in parameters:

- `target` (str): IP-address/domain-name
- `loginConfig` (dict, optional): {'user':str, 'password':str}. Defaults to None.
- `timeout` (int, optional): connection timeout.. Defaults to 3.

Action 05: SSH connect to target and run command

Source code: `SSHconnector.py`

API Usage:

Init the ssh connector obj:

```
mainInfo = ('gateway.ncl.sg', 'xxxxxx', '*******')
mainHost = sshConnector(None, mainInfo[0], mainInfo[1], mainInfo[2])
```

Pass in parameters:

- `parent` (sshConnector or paramiko.SSHClient: parent ssh client.
- `host` (str): host ip address or host domain name.
- `username` (str): username.
- `password` (str): user password.
- `port` (int, optional): ssh port. Defaults to 22.

Run command on target host

```
mainHost.addCmd('pwd', test1RplyHandleFun)
mainHost.InitTunnel()
mainHost.runCmd(interval=0.1)
```

Pass in parameters:

- `cmdline` (string): command line string.
- `handleFun`: a function use to handle the command response. default use None. Below reply dict will be passed in the handle function.
- `interval` (float, optional): Sleep time after time interval (unit second). Defaults to None.

Action 06: SCP file to target

Source code: `SCPconnector.py`

API Usage:

Init the scp connector obj:

```
scpClient = scpConnector(destInfo, showProgress=True)
```

Pass in parameters:

- `destInfo` (tuple): The destation host's ssh login information. example: (sshHost(ip/domain), userName, password)
- `jumpChain` (list, optional): The jump host chain ssh info:scpConnectorHost ---> jumphost1 ---> jumphost2---> ... ---> destinationHost[jumphost1Infor, jumphost2Info]. example: [(JumpHost1_ip, userName, password), (JumpHost2_ip, userName, password) ...] Defaults to None.
- `showProgress` (bool, optional): Flag to identify whether show the file transmission progress. Defaults to False, better to set True when transfer big file.

Upload file:

```
scpClient.uploadFile('scpTest.txt', '~/scpTest2.txt')
```

Pass in parameters:

- `srcPath` (str): source file path.
- `destPath` (str): destination file path.

Download file:

```
scpClient.downFile('~/scpTest2.txt')
```

Pass in parameters:

- `srcPath` (str): destination host file path.
- `localPath` (str, optional): local path. Defaults set same as the program folder.

Action 07: SSH port forward to local

Source code: `SSHforwarder.py`

API Usage:

Init the ssh forwarder obj:

```
forwarder = localForwarder(localport, remoteHost[0], remoteHost[1])
```

Pass in parameters:

```
def __init__(self, localPort, remoteHost, remotePort, remoteUser=None,
remotePwd=None) -> None:
```

- `localPort` (int): local port
- `remoteHost` (str): target remote host address.
- `remotePort` (str): target remote host's port need to be forwarded to local.
- `remoteUser` (str, optional): remote host username. Defaults to None.
- `remotePwd` (str, optional): remote host password. Defaults to None.

Start forward:

```
gw = {  'address': 'gateway.ncl.sg',
    'user': 'xxxxx',
    'password': '********'
    }
forwarder.addNextJH(gw['address'], gw['user'], gw['password'])
print(forwarder.getJsonInfo())
forwarder.startForward()
```

Pass in parameters: `None`

Action 08: UDP connect to the target

Source code: `udpCom.py`

API Usage:

Init the UDP client obj:

```
client = udpClient((ipAddr, udpPort))
```

Pass in parameters:

- `ipAddr` (str, int): IP address and UDP port

Send message:

```
resp = client.sendMsg(msg, resp=True)
```

Pass in parameters:

- `msg` (str): UDP message string
- `resp` (bool): server response flag, method will wait server's response and return the bytes format response if it is set to True.

Action 09: TCP connect to the target

Source code: `tcpCom.py`

API Usage:

Init the TCP client obj:

```
client = tcpClient(('127.0.0.1', 502))
```

Pass in parameters:

- `ipAddr` (str, int): IP address and UDP port

Send message:

```
resp = client.sendMsg(msg, resp=True)
```

Pass in parameters:

- `msg` (str): TCP message string
- `resp` (bool): server response flag, method will wait server's response and return the bytes format response if it is set to True.

Action 10: Connect to SQLite3 Database

Source code: `databaseHandler.py`

API Usage:

Init the SQLite3 connector obj:

```
dbhandler = Sqlite3Cli('database.db', databaseName ='testdb',threadSafe=False,
rowFac=sqlite3.Row )
```

Pass in parameters:

- `dbPath` (str): sqlite3 database local path or connection url.
- `databaseName` (str, optional): Name of the DB. Defaults to None.
- `threadSafe` (bool, optional): flag to check_same_thread, if you want the client be used in different thread, set the val to False. Defaults to True.
- `rowFac` (*type*, optional): select row factor. Defaults to None.

Run SQL query string:

```
executeQuery(self, queryStr, paramList=None)
```

Pass in parameters:

- `queryStr` (str): query string
- `paramList` (tuple, optional): parameter tuple. Defaults to None.

Run SQL query script;

```
executeScript(self, scriptPath)
```

Pass in parameters:

- `scriptPath` (str): query script file path.

Action 11: Connect to influxDB1.8X Database

Source code: `databaseHandler.py`

API Usage:

Init the influxDB connector obj:

```
client = InfluxCli(ipAddr=('127.0.0.1', 8086), dbinfo=('root', 'root',
'gatewayDB'))
```

Pass in parameters:

- `ipAddr` (str, int): IP address and UDP port
- `dbinfo` (str, str, str): user name, password, DB name

Insert Fields:

```
insertFields(self, measurement, fieldDict, tags=None, timeStr=None)
```

- `measurement`(str): measurement name
- `fieldDict`(dict): data field dictionary

Action 12: Send and Receive Email

Source code: `emailActor.py`

API Usage:

Init the email actor:

```
actor = emailActor.emailActor('xxx@gmail.com', '******')
```

Pass in parameters:

- `account` (str): full email address. For example: liu_yuan_cheng@hotmail.com
- `password` (str): password.

Get the email unique ID list:

```
getEmailIdList(self, emailBox='inbox', emailNum=10, sender=None)
```

Pass in parameters:

- `emailBox` (str, optional): mailbox name. Defaults to 'inbox'.
- `emailNum` (int, optional): number of email. Defaults to 10.
- `sender` (*type*, optional): if set not None, only return the ID list of the email from the sender. Defaults to None.

Read the email based on the config setting

```
readLastMail(self, configDict=DEFAULT_CFG, downloadDir=None)
```

Pass in parameters:

- `configDict` (dict(), optional): refer to the <DEFAULT_CFG>.
- `downloadDir` (str, optional): The folder to save the attachment.

Download the attachment from the mail

```
downloadAttachment(self, emailMsg, downloadDir)
```

Pass in parameters:

- `emailMsg` : refer to <email.message_from_string()> return value.
- `downloadDir` : download folder (absolute path).

Forward an exported email file *.eml as email to destinations.

```
forwardEml(self, dests, emlFilePath)
```

Pass in parameters:

- `dests` (str): destination email address.
- `emlFilePath` (*type*): *.eml file path.

Send a html format email.

```
sendEmailHtml(self, dests, subjectStr, htmlContent, attachmentPath=None)
```

Pass in parameters:

- `dests` (str) : receiver's email address.
- `subjectStr` (str) : email subject title.

- `htmlContent` (str) : refer to the example below.
- `attachmentPath` : attachment file absolute path.

Action 13: Use browser to open URL

Source code: `functionActor.py`

API Usage:

Init the browser actor:

```
actor = browserActor(driverPath="/home/user/download/chromedriver")
```

Pass in parameters:

- driverPath (str, optional): driver path. Defaults to None.

Open a url with browser:

```
openUrls(self, urlConfig)
```

Pass in parameters:

- urlConfig(dict): URL config dictionary example:

```
urlitem = {
    'cmdID': 'YouTube',
    'url': 'https://www.youtube.com/watch?v=VMebB6hhjW4',
    'interval' :  0 # time interval to wait for next operation.
}
```

Action 14: Connect to NTP(Network Time Protocol) service

Source code: `networkServiceProber.py`

API Usage:

Check whether a NTP(Network Time Protocol) service is available and get the time information

```
checkNtpConn(self, target, pingFlg=False, portFlg=False, ntpPort=123)
```

Pass in parameters:

- `target` (str): IP-address/domain-name
- `pingFlg` (bool, optional): whether ping the server. Defaults to False.
- `portFlg` (bool, optional): whether check ntp Port connectable. Defaults to False.
- `ntpPort` (int, optional): ntp port. Defaults to 123.

Action 15: Send HTTP GET/POST request

Source code: `networkServiceProber.py`

API Usage:

Send the http request:

```
checkHttpRquest(self, requestType, url, param)
```

Pass in parameters:

- `requestType` (str): the request type 'get'/'post'
- `url` (str): api/php url
- `param` (dict): get or post input dict

GET request input example:

```
{
    "url" : "https://jsonplaceholder.typicode.com/posts/",
    "type" : "get",
    "parm": {
        "id": [1, 2, 3],
        "userId":1
     }
}
```

POST request input example:

```
{
    "url" : "https://jsonplaceholder.typicode.com/posts",
    "type" : "post",
    "parm": {
        "userID": 1,
        "id": 1,
        "title": "Making a POST request",
        "body": "This is the data we created."
    }
}
```

Application Activities Module

Action 16: Start an application file

Source code: `funcActor.py`

API Usage:

```
startFile(filePath)
```

Pass in parameters:

- `filePath` (str) : application file path.

Action 17: Edit a PPT file

Source code: `funcActor.py`

API Usage:

```
msPPTedit(filePath, actionDict)
```

Pass in parameters:

- `filePath` (str) : ppt file path.
- actionDict(list): example:

```
[
    {   "fileName": "Bob Report",
        "title": "report for this morning",
        "body": "I need to pause the ping result here",
        "picName": "pptxInputPic2.png",
        "interval": 1
    }
]
```

Action 18: Start a Zoom meeting

Source code: `zoomActor.py`

API Usage:

Init a Zoom actor obj:

```
actor = zoomActor(userName='TestUser_Bob')
```

Pass in parameters:

- userName(str): Name shown in the Zoom meeting

Start a Zoom Meeting:

```
startMeeting(self, meetUrl, appFlg=True):
```

Pass in parameters:

- meetUrl(str): meeting url
- appFlg(bool): flag to identify whether use the Zoom app to start the meeting.

Action 19: Git download/pull the project

Source code: `gitDownloader.py`

API Usage:

Init the Git downloader

```
downloader = gitDownloader(dirpath)
```

Pass in parameters:

- dirpath(str): local proejct save folder.

Clone the repo:

```
cloneRepo(self, repoUrl, folderName):
```

Pass in parameters:

- `reportUrl` (str): Git repo url
- folderName (str): repo folder name.

Action 20: Use Wireshark to capture network traffic or open pcap file

Source code: `tsharkUtils.py`

API Usage:

Init the network sniffer obj:

```
sniffer = trafficSniffer()
```

Pass in parameters: `None`

Load the network packet capture file (*.cap, *.pcap, *.pcapng):

```
loadCapFile(self, filePath, decryptionkKey=None)
```

Pass in parameters:

- `filePath` (list[str]): pcap file path list .

Capture the packet to file. If applied the filter, only capture the packet which match the filter.

```
capture2File(self, filePath, displayFilter=None, timeoutInt=30)
```

Pass in parameters:

- `filePath` (str): pcap file path
- `displayFilter` (str, optional): display filter https://wiki.wireshark.org/DisplayFilters Defaults to None.
- `timeoutInt` (int, optional): Capture time. Defaults to 30.

Capture the packet in the memory <self.packetList>

```
capture2Mem(self, displayFilter=None, packetCount=20)
```

Pass in parameters:

- `displayFilter` (str, optional): display filter. Defaults to None.
- `packetCount` (int, optional): number of packet which can match to the filter. Defaults to 20.

Action 21: Use FTK Imager to capture the memory dump file

Source code: `ftkMemDumper.py`

API Usage:

```
ftkMemDumper(ftkImagerPath)
```

Pass in parameters:

- ftkImagerPath(str): FTK imager installed path.

Start create memory dump file

```
funcActor.startFile(memDumpPath)
```

Pass in parameters:

- memDumpPath(str): path to save the memory dump file.

Action 22: Connect to Scheduler PLC to send command

Source code: `modbusTcpCom.py`

API Usage:

Init the PLC connector:

```
client = modbusTcpCom.modbusTcpClient(hostIp)
```

Pass in parameters:

- `tgtIp` (str): target PLC ip Address.
- `tgtPort` (int, optional): modbus port. Defaults to 502.
- `defaultTO` (int, optional): default time out if modbus server doesn't response. Defaults to 30 sec.

Get PLC input holding register state:

```
getHoldingRegs(self, addressIdx, offset)
```

Get PLC output coil state:

```
getCoilsBits(self, addressIdx, offset)
```

Change PLC holding register state:

```
setHoldingRegs(self, addressIdx, bitVal)
```

Change PLC coil state:

```
setCoilsBit(self, addressIdx, bitVal)
```

Action 23: Connect to Siemens RTU to send command

Source code: `snap7Comm.py`

API Usage:

Init the RTU connector:

```
client = snap7Comm.s7CommClient('127.0.0.1', rtuPort=102, snapLibPath=libpath)
```

Read the data value/bytes from the RTU memory address:

```
readAddressVal(self, addressIdx, dataIdxList=None, dataTypeList=None)
```

Pass in parameters:

- `addressIdx` (int): memory address index.
- `dataIdxList` (list(int)): data index list
- `dataTypeList` (list(XX_TYPE), optional): data type list. Defaults to None

Set the data Idx value in the address

```
setAddressVal(self, addressIdx, dataIdx, data, dataType=REAL_TYPE)
```

Pass in parameters:

- `addressIdx` (int): memory address index.
- `dataIdx` (int): data index.
- `data` (int): data value
- `dataType` (*type*, optional): data type. Defaults to REAL_TYPE.

Action 25: Use Nmap to scan the network

Source code: `nmapUtils.py`

API Usage:

Init the Nmap scanner:

```
scanner = nmapScanner()
```

Pass in parameters: `None`

Check a list TCP ports' state and service type:

```
scanTcpPorts(self, target, portList, showFiltered=False)
```

Pass in parameters:

- `target` (str): target IP address/Url.

- `portList` (str): list of int ports.
- `showFiltered` (bool, optional): whether show the 'filtered' state port. Defaults to False.

Scan a port range and return the result

```
scanPortRange(self, target, portRange, showFiltered=False)
```

Pass in parameters:

- `target` (str): target IP address/Url.
- `portRange` (tuple): (start port, end port)
- `showFiltered` (bool, optional): whether show the 'filtered' state port. Defaults to False.

Check a list of service type:

```
scanServices(self, target, serviceList)
```

Pass in parameters:

- `target` (*type*): arget IP address/Url.
- `serviceList` (*type*): service list.

Scan the subnet and find the reachable IP addresses:

```
scanSubnetIps(self, subnetStr)
```

Pass in parameters:

- `subnetStr` (str): subnet string, 192.168.10.0/24

Action 26: Use speed test to check current network speed

Source code: `speedChecker.py`

API Usage:

get the download and upload result

```
rst = speedtest.Speedtest()
rst.startTest()
```

Pass in parameters: `None`

Human Activities Module

Action 27: Record User's mouse and keyboard action

Source code: `UserActionrecorder.py`

API Usage:

Init recorder:

```
recorder = ` UserActionrecorder()
```

Pass in parameters: `None`

Start to record:

```
recorder.start()
```

Pass in parameters: `None`

Action 28: Generate the keyboard type in event

Source code: `keyEventActor.py`

API Usage:

Init event generator:

```
actor = keyEventActor(winOS=True)
```

Pass in parameters:

- `winOS` (bool): Windows OS Flag.

Simulate user press key and release the key

```
pressAndrelease(self, keySet, interval=0.1)
```

Pass in parameters:

- `keySet` (str): *description*
- `interval` (float, optional): time interval (sec) between 2 key press. Defaults to 0.1 sec.

Simulate user type in a string:

```
simuUserType(self, typeinStr, interval=0.2)
```

Pass in parameters:

- `typeinStr` (str): string or key set user type in

Action 29: Generate the mouse event

Source code : `mouseInput.py`

API Usage:

Init the mouse even generator

```
actor = mouseEventActor()
```

Pass in parameters: `None`

Simulate user mouse event:

```
actor.play(mouse_events)
```

Pass in parameters:

- `mouse_events` (dict): mouse event dictionary

Action 30: Start chrome browser and play the google dino game

Source code : `dinoActor.py`

API Usage:

Init the game player

```
actor = dinoActor(playtime=40)
```

Pass in parameters:

- `driverPath` (*type*, optional): The google driver(exe) path. Defaults to None.
- `playtime` (int, optional): how many second you want to play. Defaults to 0.

Start to play the game:

```
actor.play()
```

Pass in parameters: `None`

Action 31: Send chat message via Telegram

Source code : `telebotActor.py`

API Usage:

Init the telegram robot:

```
bot = telebotActor(botToken)
```

Pass in parameters:

- `botToken` (str): telegram bot id token.

Send message to a chat:

```
bot.sendMsg(chatID, msgStr)
```

Pass in parameters:

- `chatID` (str): Chat ID string.
- msgStr(str): message send to the chat.

Action 32: Select local file (video, music, picture) and play/open

Source code : `telebotActor.py`

API Usage:

```
selectFile(filePath)
```

Pass in parameters:

- filePath(str): file path.

Action 33: Connect to web camera and capture video

Source code: `cameraClient.py`

API Usage:

Init the web camera connector

```
cam = camClient(videoSrc=None)
```

Pass in parameters:

- videoSrc(str): web camera URL source.

Start to capture video:

```
cam.run()
```

Pass in parameters: `None`

Action 33: Play simple sudoku game

Source code: `pyQt5_Sudoku_Calculator.py`

API Usage:

Init the sudoku solver

```
w = SudokuCalculator()
```

Pass in parameters: `None`

Solve a sudoku:

```
w.calculateSu(self, imagePath)
```

Pass in parameters:

- `imagePath`(str): Sudoku image screen shot path.

System Activities Module

Action 34: Run commands on different system

Source code: `c2MwUtils.py(CmdRunner)`

API Usage:

Init the command runner obj

```
cmdrunner = Command(maxQsz=10, rstDetailFlg=True)
```

Pass in parameters:

- `maxQsz` (int, optional): max number of cmd can be enqueued. Defaults to MAX_CMD_QUEUE_SIZE.
- `rstDetailFlg` (bool, optional): flag to identify whether to show the cmd execution detailed result. Defaults to False.

Run a command and collect the result

```
runCmd(self, cmdStr, detailFlg=False)
```

Pass in parameters:

- `cmdStr` (str): command string.
- `detailFlg` (bool, optional): flag to identify whether to show/return the execution detail. Defaults to False.

Open window PowerShell console to run the cmd:

```
runWinCmds(cmdConfig, rstRecord=False)
```

Pass in parameters:

- `cmdConfig` (dict): power shell command config dict, exmaple:

```
[
    {
        "cmdID": "cmd_1",
        "console": true,
        "cmdStr": "ping -n 5 www.google.com.sg",
        "repeat": 1,
        "interval": 0.8
    },
    {
        "cmdID": "cmd_2",
        "console": false,
        "cmdStr": "ipconfig",
        "repeat": 1,
        "interval": 0.8
    },
]
```

Action 35: Open local connected camera can capture video

Source code: `cameraServer.py`

API Usage:

Init the local camera capture server

```
cam = camServer(camIdx=0)
```

Pass in parameters:

- camIdx(int, optional): camera index.

Motion detection and target tracking function:

```
cam.detectTgt(self, frame)
```

Pass in parameters:

- `frame` (cv.frame): image frame

Start to capture the video:

```
cam.run()
```

Pass in parameters: `None`

Action 36: Open RS232/485 Comm port to read/set serial data

Source code: `serialCom.py`

API Usage:

Init the serialCom connector:

```
connector = serialCom(None,serialPort="COM_NOT_EXIST", baudRate=115200)
```

Pass in parameters:

- `serialPort` (str): Port ID
- `baudRate` (int): data transmission rate Bytes/sec

Call `read(int *)` or `write(bytes *)` to read number of bytes from the serial port or send bytes to the port. Call close() to close the port.

Action 37: Check OS state and record result

Source code: `localServiceProber.py`

API Usage:

Init the OS state prober obj:

```
prober = localServiceProber()
```

Get the cpu, ram, login user, disk and network connection state

```
getResUsage(self, configDict=None):
```

Pass in parameters:

- `configDict` (dict, optional): result config dictionary, example:

```
configDict = {
    'cpu': {'interval': 0.1, 'percpu': False},
    'ram': 0,
    'user': None,
    'disk': ['C:'],
    'network': {'connCount': 0}
} .Defaults to None.
```

Get the total process number and spec process detail (name, id , users)

```
getProcessState(self, configDict=None)
```

Pass in parameters:

- `configDict` (dict, optional): result config dictionary, example:

```
configDict = {
    'process': {
        'count': 0,
        'filter': ['python.exe']
    }
} . Defaults to None.
```

Get the folder contents

```
getDirFiles(self, configDict=None)
```

Pass in parameters:

- `configDict` (dict, optional): The folder path and deep of check, example: `{'dir':{}}`

Action 38: User Ettercap to mirror target network traffic

Source code: `ettercapWrapper.py`

API Usage:

Init the Ettercap wrapper obj:

```
client = ettercapWrapper(malwareID, ownIP, c2Ipaddr, c2port=c2Port,
reportInt=c2RptInv, tasksList=taskList, c2HttpsFlg=c2HttpsFlg)
```

Pass in parameters:

- `malwareID` (str): malware id
- `ownIp` (str): malware ip address
- `c2Ipaddr` (str): c2 server IP address
- `reportInt` (int, optional): time interval between 2 report to c2. Defaults to 10 sec.
- `tasksList` (list of dict, optional): refer to taskList. Defaults to None.
- `c2HttpsFlg` (bool, optional): flag to identify whether connect to c2 via https. Defaults to False.
- `cmdTDFlg` (bool, optional): flag to identify whether run the command execution task in the command runner's sub-thread. Defaults to False.

Use the command running to execute the ettercap in sub thread and apply the related filleter on it.

```
runPktFilter(self, fileterCfgStr)
```

Pass in parameters:

- `fileterCfgStr`(str): fileter config string. Example:

```
formate 1: <filetername> : use the default local filter json config.
formate 2: <filetername>;<target ip>: use the user assigned config.
```

Action 39: Use Google map API to get the direction to destination

Source code: `geoLRun.py`

API Usage:

Init the geolocation director obj:

```
mainFrame = GeoLFrame(None, -1, gv.APP_NAME)
```

Get the diction result:

```
fineRoute(self, pos)
```

Pass in parameters:

- `pos`(str): destination GPS location or Name.

Action 40: Use Google map API to get the direction to destination

Action 40: Open Cytoscape to view graph and convert to Json file

Source code: `Cytoscape_2_Json.py`

API Usage:

Open cytoscape and load 'case_*' file, then convert the to the json file. Each case file should only have one cytoscape graph build in.

```
caseCvt(filePath, outPutDir)
```

Pass in parameters:

- `filePath` (Str): Case* source data file path.
- `outPutDir` (Str): Output directory path. Exmaple: The result Json file will be saved as 'outPutDir/case/filename.json'

Open cytoscape and load the 'linked*', *'subgraphs*' file, then convert to the json file. The linked/subgrapshs files can have multiple cytoscape graphs build in.

```
graphCvt(filePath, outPutDir, graphType='linked',graphName='snort_forti')
```

Pass in parameters:

- `filePath` (Str): Case* source data file path.
- `outPutDir` (Str): Output directory path. The result Json file will besaved as outPutDir/filename.json
- `graphType` (str, optional): Identify what kind of data we want to convert. Defaults to 'linked'.
- `graphName` (str, optional): Identify the graph name we want to find in the cytoscape file. Defaults to 'snort_forti'.

Action 41: Do python program obfuscation encode and decode

Source code: `pyObfuscator.py`

API Usage:

Obfuscate encode the input python source code.

```
obfEecode(data, removeCmt=True)
```

Pass in parameters:

- `data` (str): python source code string.
- `removeCmt` (bool, optional): flag to remove the code original comments and empty line. Defaults to True.

Decode the obfuscate bytes back to source code.

```
obfDecode(data, removeCmt=True)
```

Pass in parameters:

- `data` (bytes): obfuscated bytes data.
- `removeCmt` (bool, optional): flag to remove the random comments and empty line of the result. Defaults to True.

> Last edit by LiuYuancheng (liu_yuan_cheng@hotmail.com) at 12/05/2024, if you have any problem or find anu bug, please send me a message .