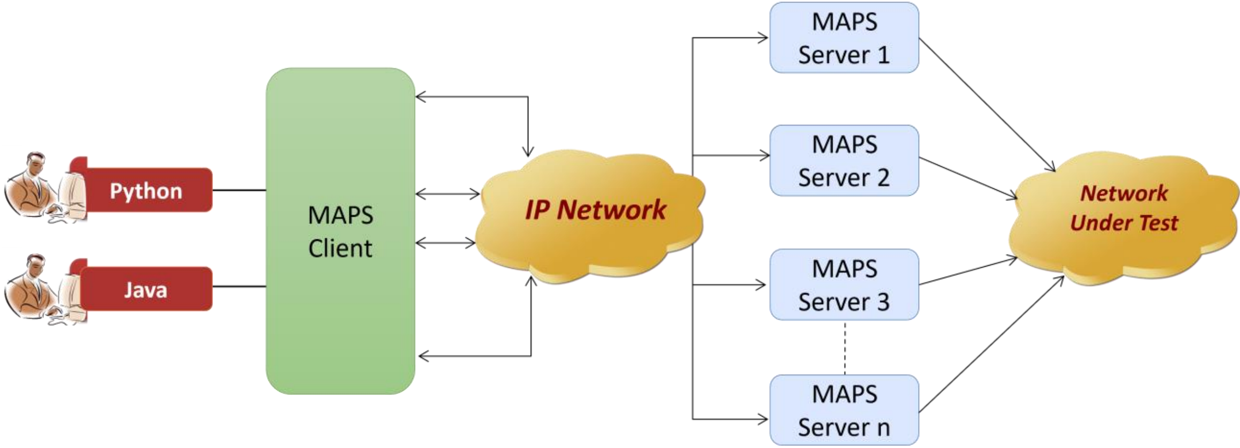
MAPS™ APIs for Complete Automation

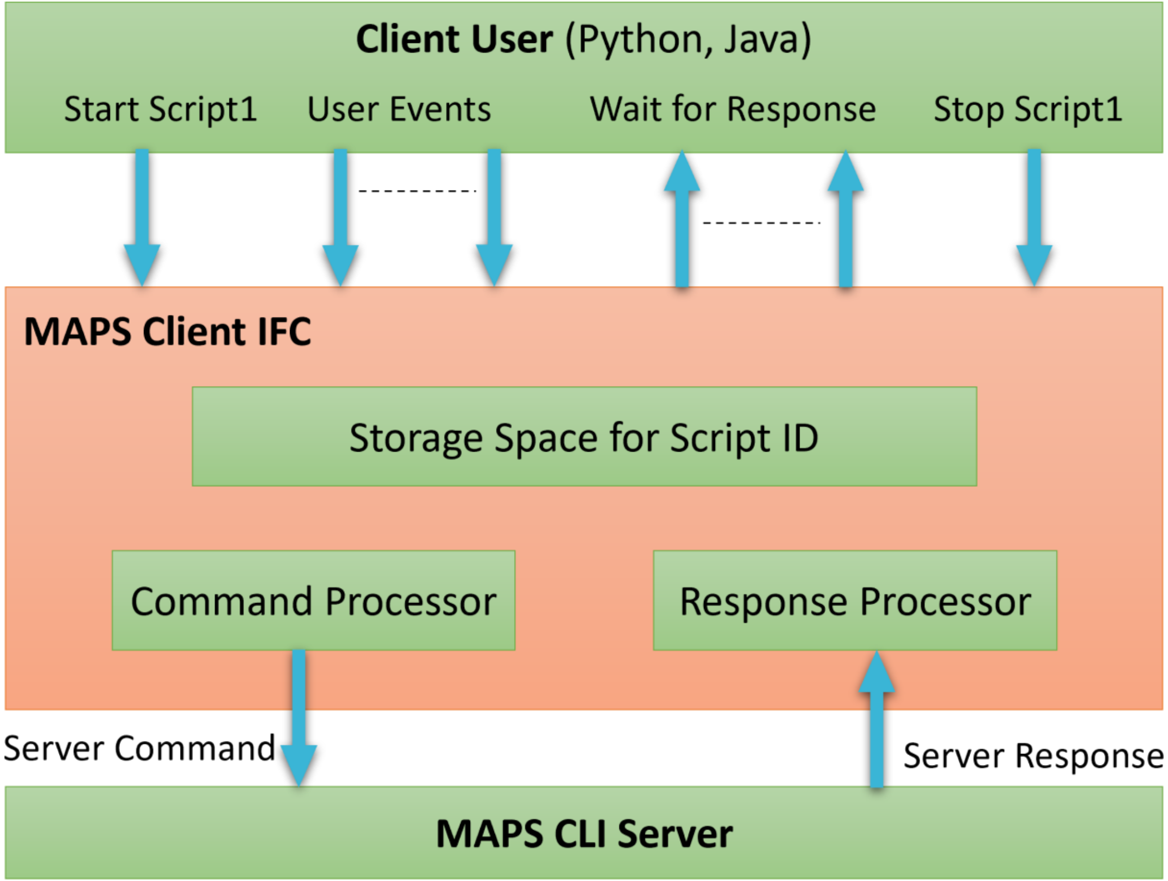
GL Communications Inc.
818 West Diamond Avenue - Third Floor, Gaithersburg, MD 20878
Phone: (301) 670-4784 Fax: (301) 670-9187 Email: info@gl.com
Website: https://www.gl.com

API Overview

- API wraps our proprietary scripting language in standard languages familiar to the user:
- Python
- Java
- Clients and Servers support a “Many-to-Many” relationship, making it very easy for users to develop complex test cases involving multiple signaling protocols



Working Principle of MAPS™ CLI



Client User (Python, Java)

Start Script1

User Events

Wait for Response

Stop Script1

MAPS Client IFC

Storage Space for Script ID

Command Processor

Response Processor

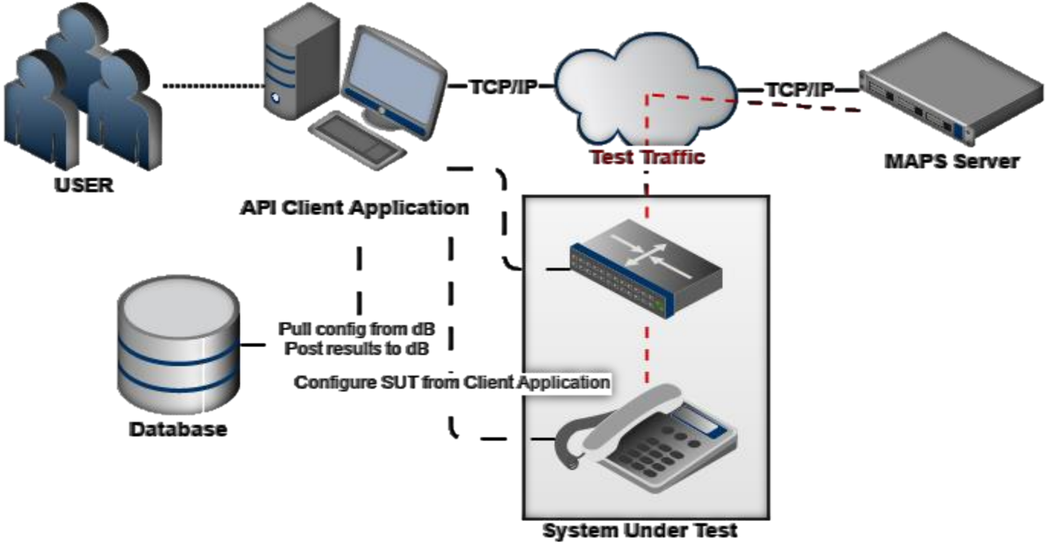
Server Command

Server Response

MAPS CLI Server

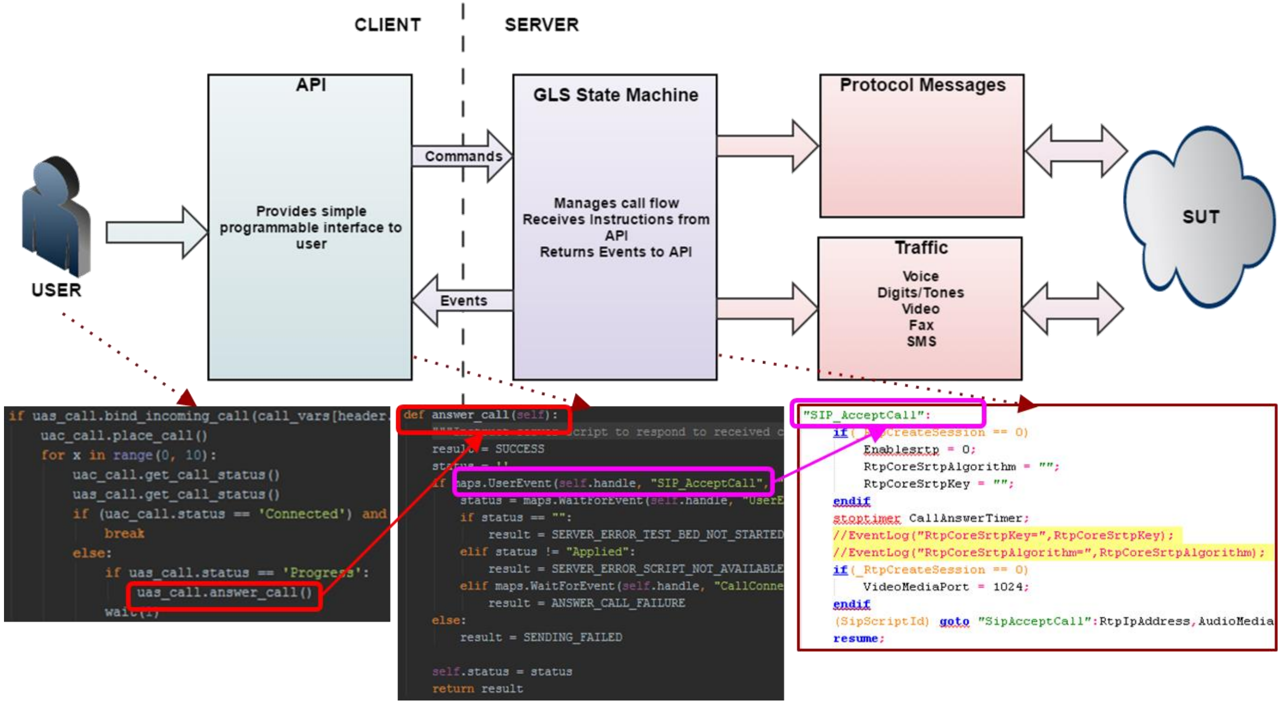
System Integration

- The same Client Application used to control MAPS™ can be, and very often is, used to control other elements of the System Under Test



System Integration (Contd.)

- Client Application can be as simple as executing a script from an IDE or it can be integrated into a full-fledged automation



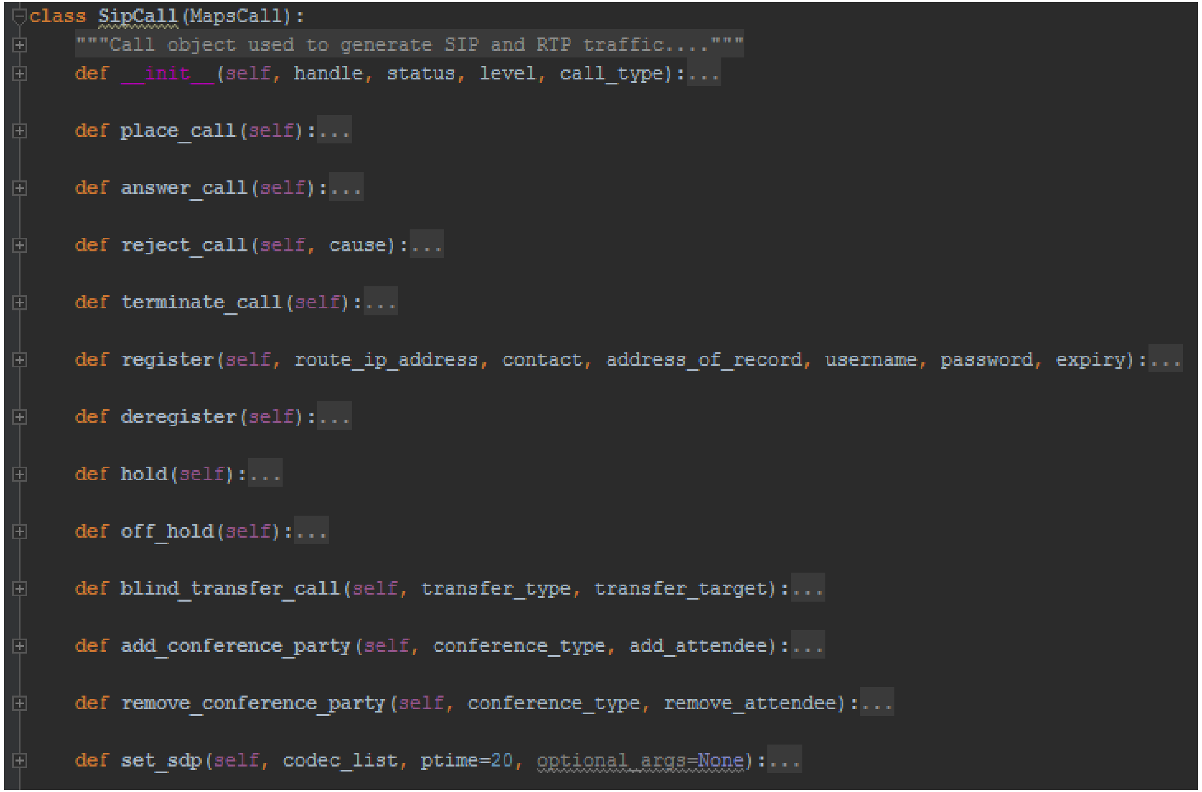
```
if uas_call.bind_incoming_call(call_vars[header.
    uac_call.place_call()
    for x in range(0, 10):
        uac_call.get_call_status()
        uas_call.get_call_status()
        if (uac_call.status == 'Connected') and
            break
        else:
            if uas_call.status == 'Progress':
                uas_call.answer_call()
            wait(1)
```

```
def answer_call(self):
    """Instruct server script to respond to received c
    result = SUCCESS
    status = ''
    if maps.UserEvent(self.handle, "SIP_AcceptCall", "
        status = maps.WaitForEvent(self.handle, "UserE
        if status == "":
            result = SERVER_ERROR_TEST_BED_NOT_STARTED
        elif status != "Applied":
            result = SERVER_ERROR_SCRIPT_NOT_AVAILABLE
        elif maps.WaitForEvent(self.handle, "CallConne
            result = ANSWER_CALL_FAILURE
    else:
        result = SENDING_FAILED

    self.status = status
    return result
```

```
"SIP_AcceptCall":
    if(_RtpCreateSession == 0)
        Enablesrtp = 0;
        RtpCoreSrtpAlgorithm = "";
        RtpCoreSrtpKey = "";
    endif
    stoptimer CallAnswerTimer;
    //EventLog("RtpCoreSrtpKey=",RtpCoreSrtpKey);
    //EventLog("RtpCoreSrtpAlgorithm=",RtpCoreSrtpAlgorithm);
    if(_RtpCreateSession == 0)
        VideoMediaPort = 1024;
    endif
    (SipScriptId) goto "SipAcceptCall":RtpIpAddress,AudioMedia
    resume;
```

Signaling Functions

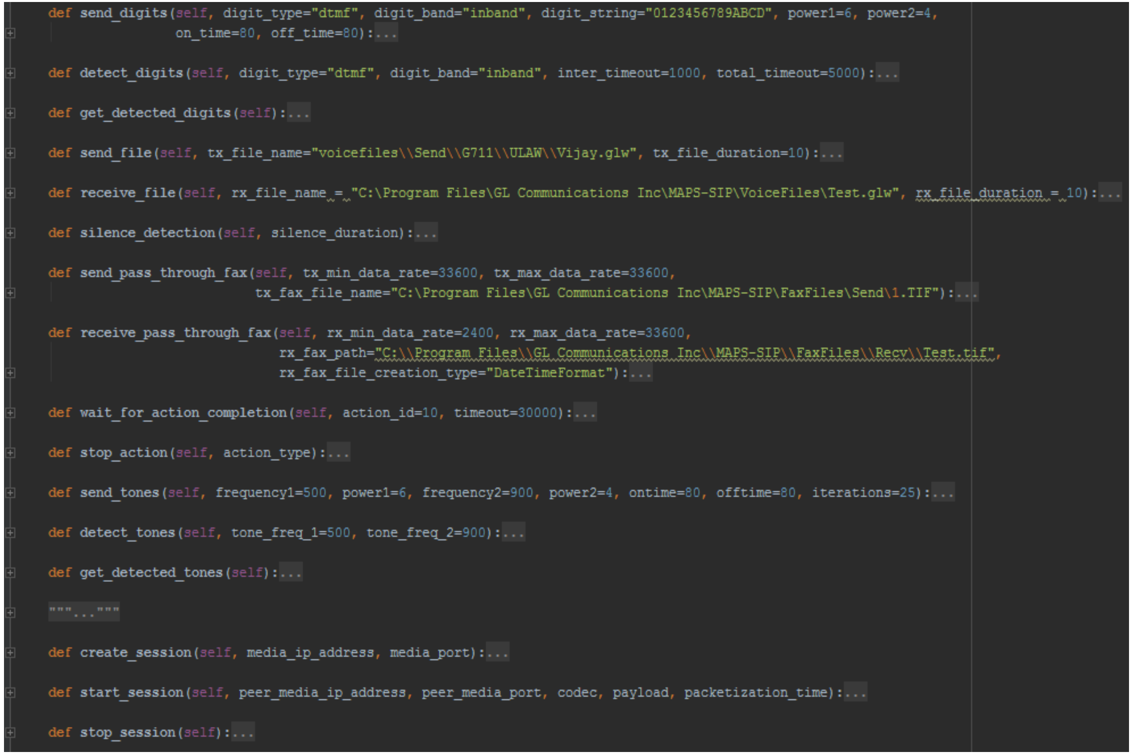
- Each protocol comes with a pre-built set of functions for basic calling features, ie **place_call(), answer_call(), reject_call(), terminate_call()** etc can be found in (almost) all protocols
- Many protocols also have specialized functions unique to them ie **register()** and **deregister()** for SIP



```
class SipCall(MapsCall):
    """Call object used to generate SIP and RTP traffic...."""
    def __init__(self, handle, status, level, call_type):...

    def place_call(self):...

    def answer_call(self):...

    def reject_call(self, cause):...

    def terminate_call(self):...

    def register(self, route_ip_address, contact, address_of_record, username, password, expiry):...

    def deregister(self):...

    def hold(self):...

    def off_hold(self):...

    def blind_transfer_call(self, transfer_type, transfer_target):...

    def add_conference_party(self, conference_type, add_attendee):...

    def remove_conference_party(self, conference_type, remove_attendee):...

    def set_sdp(self, codec_list, ptime=20, optional_args=None):...
```

Traffic Functions

- The API also delivers a library of Traffic functions of the generation of RTP or TDM traffic
- Digit, Tone, Voice File, Video File and Fax File transmission and reception are all supported
- Same default values supplied for all functions to make it easy for users who don’t require fine grained control

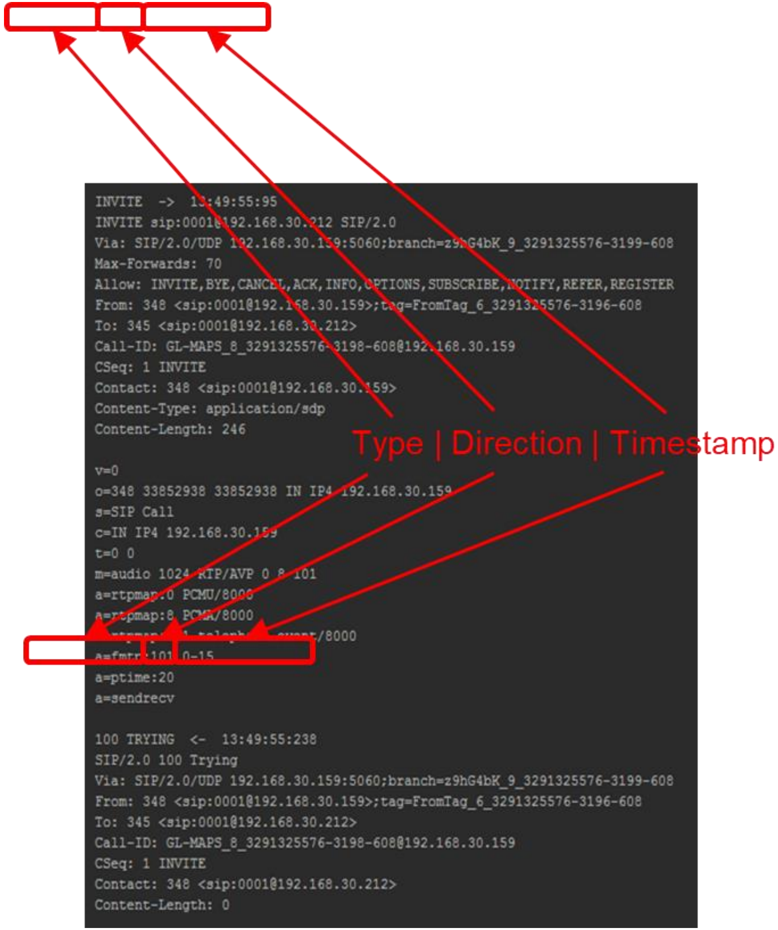


```
def send_digits(self, digit_type="dtmf", digit_band="inband", digit_string="0123456789ABCD", power1=6, power2=4,
                on_time=80, off_time=80):...

def detect_digits(self, digit_type="dtmf", digit_band="inband", inter_timeout=1000, total_timeout=5000):...

def get_detected_digits(self):...

def send_file(self, tx_file_name="voicefiles\\Send\\G711\\ULAW\\Vijay.glw", tx_file_duration=10):...

def receive_file(self, rx_file_name = "C:\Program Files\GL Communications Inc\MAPS-SIP\VoiceFiles\Test.glw", rx_file_duration = 10):...

def silence_detection(self, silence_duration):...

def send_pass_through_fax(self, tx_min_data_rate=33600, tx_max_data_rate=33600,
                          tx_fax_file_name="C:\Program Files\GL Communications Inc\MAPS-SIP\FaxFiles\Send\1.TIF"):...

def receive_pass_through_fax(self, rx_min_data_rate=2400, rx_max_data_rate=33600,
                             rx_fax_path="C:\\Program Files\\GL Communications Inc\\MAPS-SIP\\FaxFiles\\Recv\\Test.tif",
                             rx_fax_file_creation_type="DateTimeFormat"):...

def wait_for_action_completion(self, action_id=10, timeout=30000):...

def stop_action(self, action_type):...

def send_tones(self, frequency1=500, power1=6, frequency2=900, power2=4, ontime=80, offtime=80, iterations=25):...

def detect_tones(self, tone_freq_1=500, tone_freq_2=900):...

def get_detected_tones(self):...

"""..."""

def create_session(self, media_ip_address, media_port):...

def start_session(self, peer_media_ip_address, peer_media_port, codec, payload, packetization_time):...

def stop_session(self):...
```

]

Message Decode

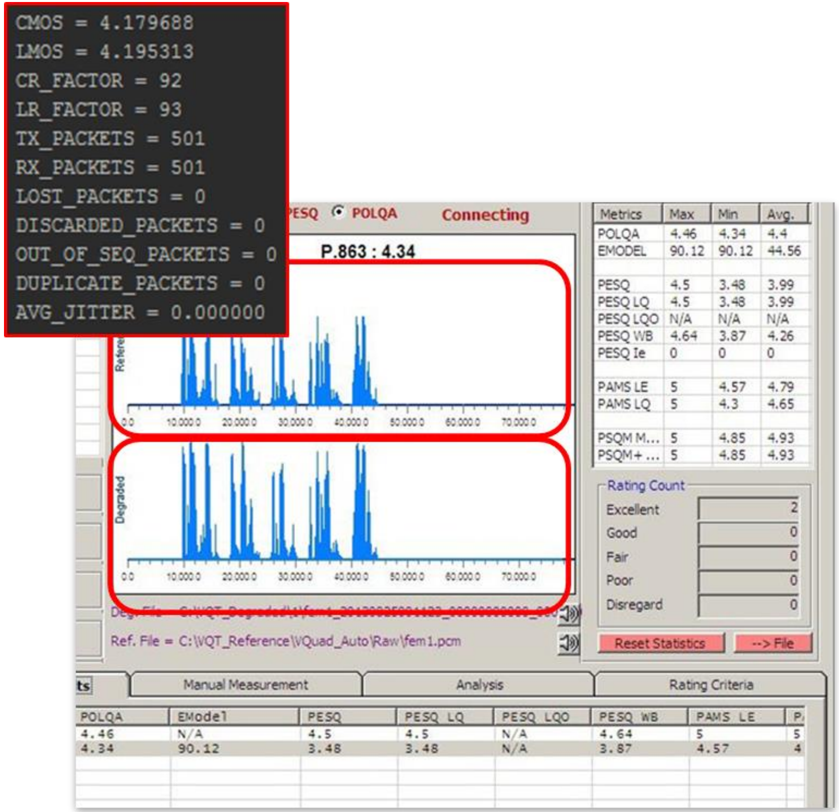
- We can extract complete message sequences from calls into objects in our API languages. These objects will hold:
  - Message type (ie INVITE)
  - Message direction (Tx / Rx)
  - Message timestamp (w/ ms accuracy)
  - Full message decode
  - Use this for custom pass/fail verification, message/response delay calculation, etc.
  - Messages can even be extracted in real time for custom parsing in the API language



```
INVITE  ->  13:49:55:95
INVITE sip:0001@192.168.30.212 SIP/2.0
Via: SIP/2.0/UDP 192.168.30.159:5060;branch=z9hG4bK_9_3291325576-3199-608
Max-Forwards: 70
Allow: INVITE,BYE,CANCEL,ACK,INFO,OPTIONS,SUBSCRIBE,NOTIFY,REFER,REGISTER
From: 348 <sip:0001@192.168.30.159>;tag=FromTag_6_3291325576-3196-608
To: 345 <sip:0001@192.168.30.212>
Call-ID: GL-MAPS_8_3291325576-3198-608@192.168.30.159
CSeq: 1 INVITE
Contact: 348 <sip:0001@192.168.30.159>
Content-Type: application/sdp
Content-Length: 246

v=0
o=348 33852938 33852938 IN IP4 192.168.30.159
s=SIP Call
c=IN IP4 192.168.30.159
t=0 0
m=audio 1024 RTP/AVP 0 8 101
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
a=sendrecv

100 TRYING  <-  13:49:55:238
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 192.168.30.159:5060;branch=z9hG4bK_9_3291325576-3199-608
From: 348 <sip:0001@192.168.30.159>;tag=FromTag_6_3291325576-3196-608
To: 345 <sip:0001@192.168.30.212>
Call-ID: GL-MAPS_8_3291325576-3198-608@192.168.30.159
CSeq: 1 INVITE
Contact: 348 <sip:0001@192.168.30.212>
Content-Length: 0
```

Voice Quality Analysis

- MAPS™ offers two integrated forms of Voice Quality Measurement: Packet Analysis and Waveform analysis
- Packet-based protocols which carry RTP traffic can be analyzed for MOS, loss, discard, sequence errors, duplication errors and jitter
- TDM and packet protocols can employ our VQT product to deliver PESQ and/or POLQA (essentially this involves the transmission of a known voice file through an SUT, and then a post-processed comparison of the degraded file to the pre-transmission reference file)



```
CMOS = 4.179688
LMOS = 4.195313
CR_FACTOR = 92
LR_FACTOR = 93
TX_PACKETS = 501
RX_PACKETS = 501
LOST_PACKETS = 0
DISCARDED_PACKETS = 0
OUT_OF_SEQ_PACKETS = 0
DUPLICATE_PACKETS = 0
AVG_JITTER = 0.000000
```

P.863 : 4.34

Ref. File = C:\VQT_Reference\VQuad_Auto\Raw\fem1.pcm

Metrics	Max	Min	Avg.
POLQA	4.46	4.34	4.4
EMODEL	90.12	90.12	44.56
PESQ	4.5	3.48	3.99
PESQ LQ	4.5	3.48	3.99
PESQ LQO	N/A	N/A	N/A
PESQ WB	4.64	3.87	4.26
PESQ Ie	0	0	0
PAMS LE	5	4.57	4.79
PAMS LQ	5	4.3	4.65
PSQM M...	5	4.85	4.93
PSQM+ ...	5	4.85	4.93

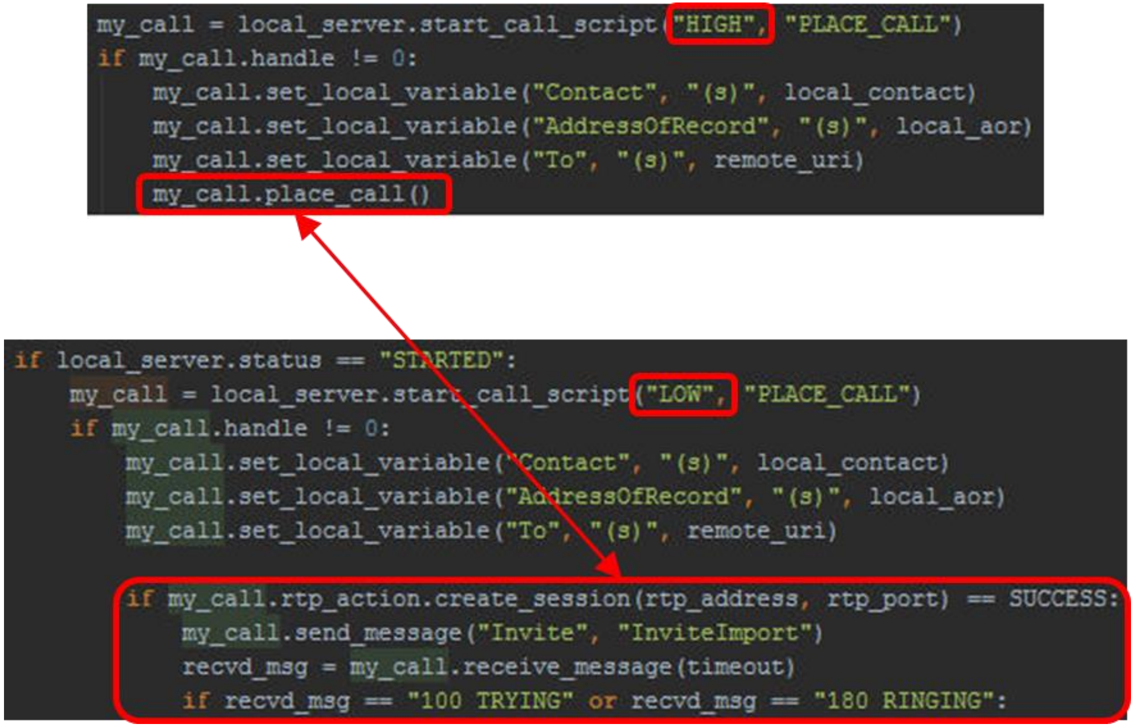
Rating Count

Excellent	2
Good	0
Fair	0
Poor	0
Disregard	0

POLQA	EModel	PESQ	PESQ LQ	PESQ LQO	PESQ WB	PAMS LE	P
4.46	N/A	4.5	4.5	N/A	4.64	5	5
4.34	90.12	3.48	3.48	N/A	3.87	4.57	4

High and Low Level Scripts

- The API is broken into High and Low level function calls / scripts
- For High Level scripts, all the fine-grained protocol control happen in the script running on the MAPS™ server, hidden from the API user
- Low Level scripts put the API user in complete control of the protocol stack. This makes Low Level scripts more flexible and powerful, but also correspondingly more complex



```
my_call = local_server.start_call_script("HIGH", "PLACE_CALL")
if my_call.handle != 0:
    my_call.set_local_variable("Contact", "(s)", local_contact)
    my_call.set_local_variable("AddressOfRecord", "(s)", local_aor)
    my_call.set_local_variable("To", "(s)", remote_uri)
    my_call.place_call()
```

```
if local_server.status == "STARTED":
    my_call = local_server.start_call_script("LOW", "PLACE_CALL")
    if my_call.handle != 0:
        my_call.set_local_variable("Contact", "(s)", local_contact)
        my_call.set_local_variable("AddressOfRecord", "(s)", local_aor)
        my_call.set_local_variable("To", "(s)", remote_uri)

        if my_call.rtp_action.create_session(rtp_address, rtp_port) == SUCCESS:
            my_call.send_message("Invite", "InviteImport")
            recvd_msg = my_call.receive_message(timeout)
            if recvd_msg == "100 TRYING" or recvd_msg == "180 RINGING":
```

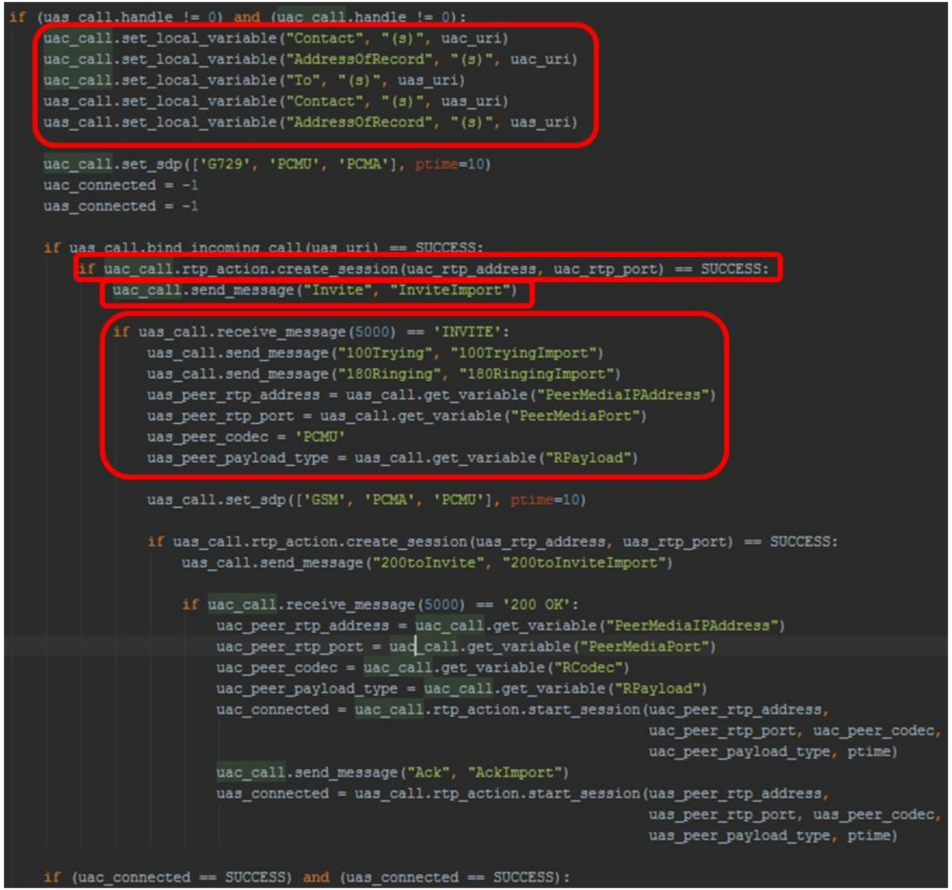
High Level Example: SipBasicCall.py

- Initialize variables, this is the only part of the script the user needs to modify to place a basic call
- Those variables get passed to script before call is started
- Call is generated with a single line, all fine-grained details of the protocol are hidden from the user. The same **place_call()** function works in *all* protocols supported by MAPS™
- RTP is transmitted with a single line, arguments permitted but not required.
- Call is terminated with a single line, all fine-grained details of the protocol are hidden from the user. The same **terminate_call()** function works in all protocols supported by MAPS™

```python
from MapsSipApi import *


def main():
    local_ip = "192.168.30.159"
    local_port = 10024
    # protocol = "SIP"

    local_contact = "0001@192.168.30.159"
    local_aor = local_contact
    remote_uri = "0001@192.168.30.212"

    local_server = SipClient(local_ip, local_port)
    if local_server.connect() == 0:
        if local_server.start_testbed() == 0:
            if local_server.load_profile_group() == 0:
                if local_server.set_global_variable("_EnableCLI", "(i)", "1") == 0:
                    print "SERVER INITIALIZED"

    if local_server.status == "STARTED":
        my_call = local_server.start_call_script("HIGH", "PLACE_CALL")
        if my_call.handle != 0:
            my_call.set_local_variable("Contact", "(s)", local_contact)
            my_call.set_local_variable("AddressOfRecord", "(s)", local_aor)
            my_call.set_local_variable("To", "(s)", remote_uri)
            my_call.place_call()
            for x in range(0, 10):
                my_call.get_call_status()
                wait(1)
                if my_call.status == "Connected":
                    break

        if my_call.status == "Connected":
            print "CONNECTED"
            if my_call.rtp_action.send_digits() == 0:
                print "RTP Action pass"
            else:
                print "RTP Action Fail"

            wait(10)
            print my_call.rtp_stats.get_rtp_statistics()
            wait(5)
            my_call.terminate_call()
            for index in range(0, my_call.get_message_count()):
                my_call.message_list.append(my_call.get_message(index))
                print my_call.message_list[index]
        local_server.stop_call_script(my_call)

if __name__ == "__main__":
    main()
```

Low Level Example: SipLowBasicCall.py

- The same set of variables exist in the Low Level Scripts and are passed with the same function
- Where the High Level user just issues place_call(), the low level user must:
    - ➢ create_session() to open an RTP socket
    - ➢ send_message(“Invite”) to start the call
    - ➢ Manually process responses

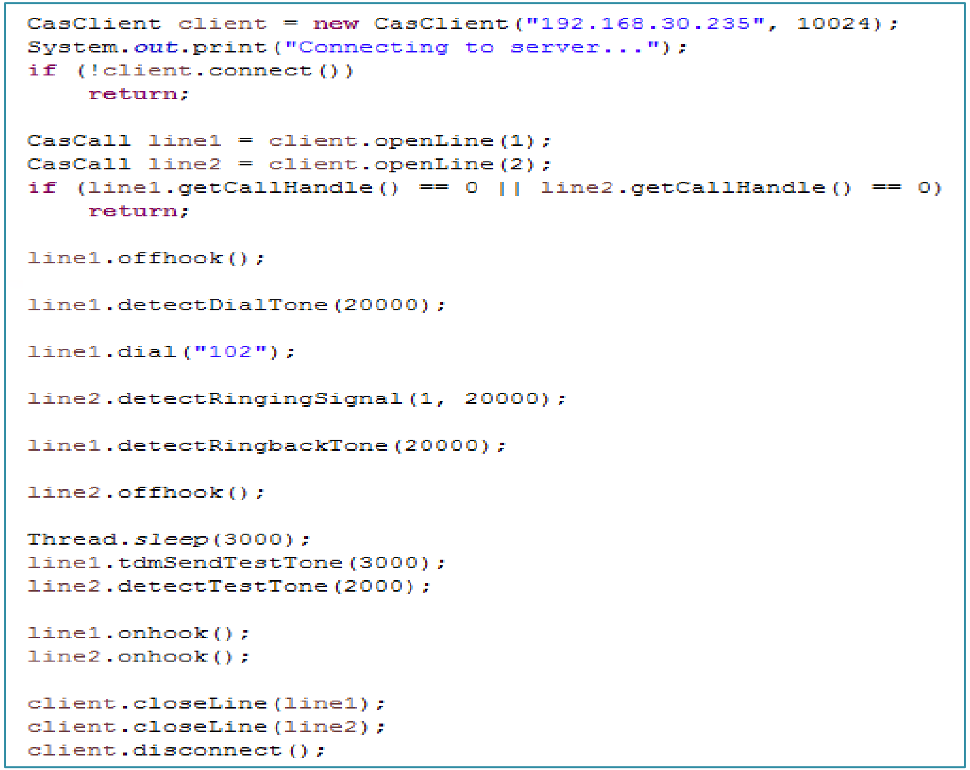


```
if (uas_call.handle != 0) and (uac_call.handle != 0):
    uac_call.set_local_variable("Contact", "(s)", uac_uri)
    uac_call.set_local_variable("AddressOfRecord", "(s)", uac_uri)
    uac_call.set_local_variable("To", "(s)", uas_uri)
    uas_call.set_local_variable("Contact", "(s)", uas_uri)
    uas_call.set_local_variable("AddressOfRecord", "(s)", uas_uri)

    uac_call.set_sdp(['G729', 'PCMU', 'PCMA'], ptime=10)
    uac_connected = -1
    uas_connected = -1

    if uas_call.bind_incoming_call(uas_uri) == SUCCESS:
        if uac_call.rtp_action.create_session(uac_rtp_address, uac_rtp_port) == SUCCESS:
            uac_call.send_message("Invite", "InviteImport")

            if uas_call.receive_message(5000) == 'INVITE':
                uas_call.send_message("100Trying", "100TryingImport")
                uas_call.send_message("180Ringing", "180RingingImport")
                uas_peer_rtp_address = uas_call.get_variable("PeerMediaIPAddress")
                uas_peer_rtp_port = uas_call.get_variable("PeerMediaPort")
                uas_peer_codec = 'PCMU'
                uas_peer_payload_type = uas_call.get_variable("RPayload")

                uas_call.set_sdp(['GSM', 'PCMA', 'PCMU'], ptime=10)

                if uas_call.rtp_action.create_session(uas_rtp_address, uas_rtp_port) == SUCCESS:
                    uas_call.send_message("200toInvite", "200toInviteImport")

                    if uac_call.receive_message(5000) == '200 OK':
                        uac_peer_rtp_address = uac_call.get_variable("PeerMediaIPAddress")
                        uac_peer_rtp_port = uac_call.get_variable("PeerMediaPort")
                        uac_peer_codec = uac_call.get_variable("RCodec")
                        uac_peer_payload_type = uac_call.get_variable("RPayload")
                        uac_connected = uac_call.rtp_action.start_session(uac_peer_rtp_address,
                                                                          uac_peer_rtp_port, uac_peer_codec,
                                                                          uac_peer_payload_type, ptime)
                        uac_call.send_message("Ack", "AckImport")
                        uas_connected = uas_call.rtp_action.start_session(uas_peer_rtp_address,
                                                                          uas_peer_rtp_port, uas_peer_codec,
                                                                          uas_peer_payload_type, ptime)

    if (uac_connected == SUCCESS) and (uas_connected == SUCCESS):
```

GL Communications

CAS/FXO/FXS API

- Channel Associated Signaling
- Method of signaling where a channel carrying speech also carries the signaling and addressing to set up and tear down calls
- Supervisory signaling carried as “onhook” and “offhook”, addressing signaling carried as DTMF or MF tones
- All functions are “low level”
- Signaling bits manipulation, call progress tone/signal detection, TDM traffic transmission/reception



```
CasClient client = new CasClient("192.168.30.235", 10024);
System.out.print("Connecting to server...");
if (!client.connect())
    return;

CasCall line1 = client.openLine(1);
CasCall line2 = client.openLine(2);
if (line1.getCallHandle() == 0 || line2.getCallHandle() == 0)
    return;

line1.offhook();

line1.detectDialTone(20000);

line1.dial("102");

line2.detectRingingSignal(1, 20000);

line1.detectRingbackTone(20000);

line2.offhook();

Thread.sleep(3000);
line1.tdmSendTestTone(3000);
line2.detectTestTone(2000);

line1.onhook();
line2.onhook();

client.closeLine(line1);
client.closeLine(line2);
client.disconnect();
```

Regression from .csv

- Use the API language to easily access and read large regression configurations from local .csv files
- Similarly, the API language can pull regression configurations from a database instead

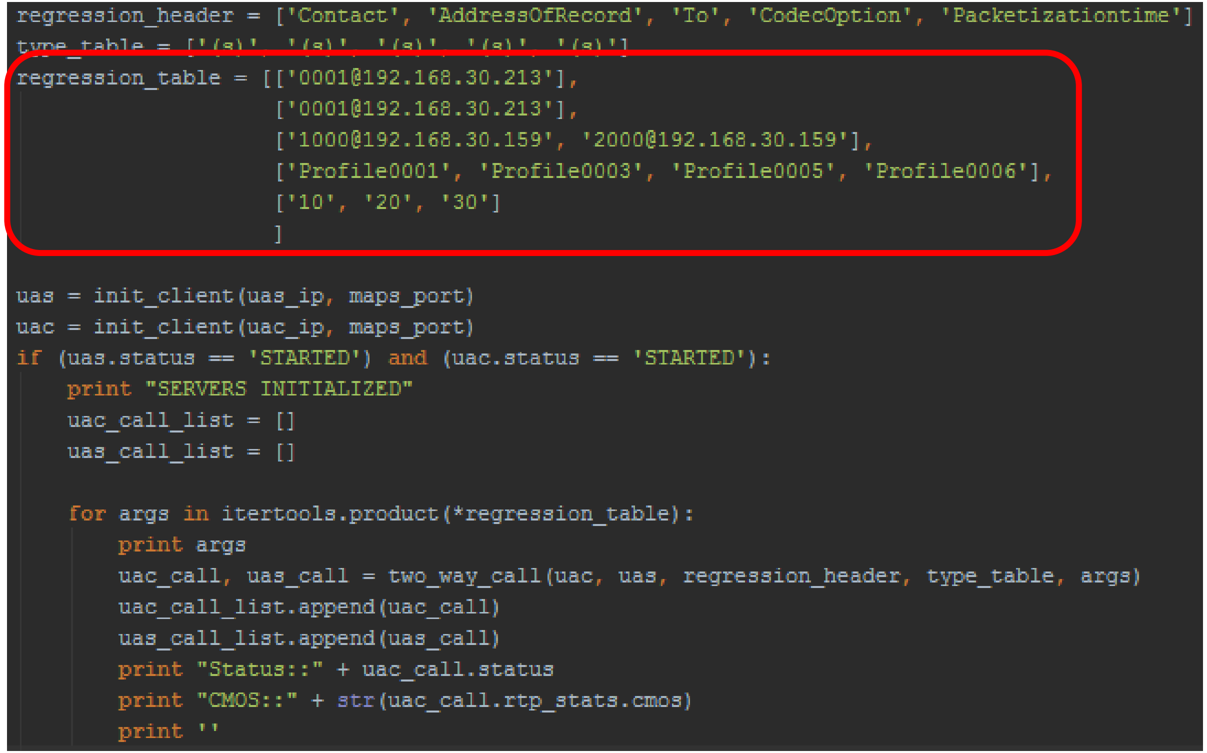
```
Contact,AddressOfRecord,To,CodecOption
(s),(s),(s),(s)
0001@192.168.30.159,0001@192.168.30.159,0001@192.168.30.212
0002@192.168.30.159,0002@192.168.30.159,0002@192.168.30.212
0003@192.168.30.159,0003@192.168.30.159,0003@192.168.30.212
0004@192.168.30.159,0004@192.168.30.159,0004@192.168.30.212
```

```
import s
import c
from Map

def main(src_file='C:\\Users\\GL\Desktop\\example_regression.csv'):
    local_ip = "192.168.30.159"
    local_port = 10024
    local_server = SipClient(local_ip, local_port)

    f = open(src_file, "rb")
    reader = csv.reader(f, delimiter=',')

    regression_header = reader.next()
    type_table = reader.next()
    regression_table = list(reader)
    f.close()

    call_list = []
    init_server(local_server)
    if local_server.status == "STARTED":
        for call_params in regression_table:
            print "CALL " + str(len(call_list) + 1) + "..."
            call_list.append(basic_call(local_server, regression_header, type_table, call_params))

    print "REGRESSION COMPLETE"
    for call in call_list:
        print "Status::" + call.status
        print "CMOS::" + str(call.rtp_stats.cmos)
        print ""
```

Multiplex Regression

- Use the advanced features of API languages to quickly and simply build complex regressions

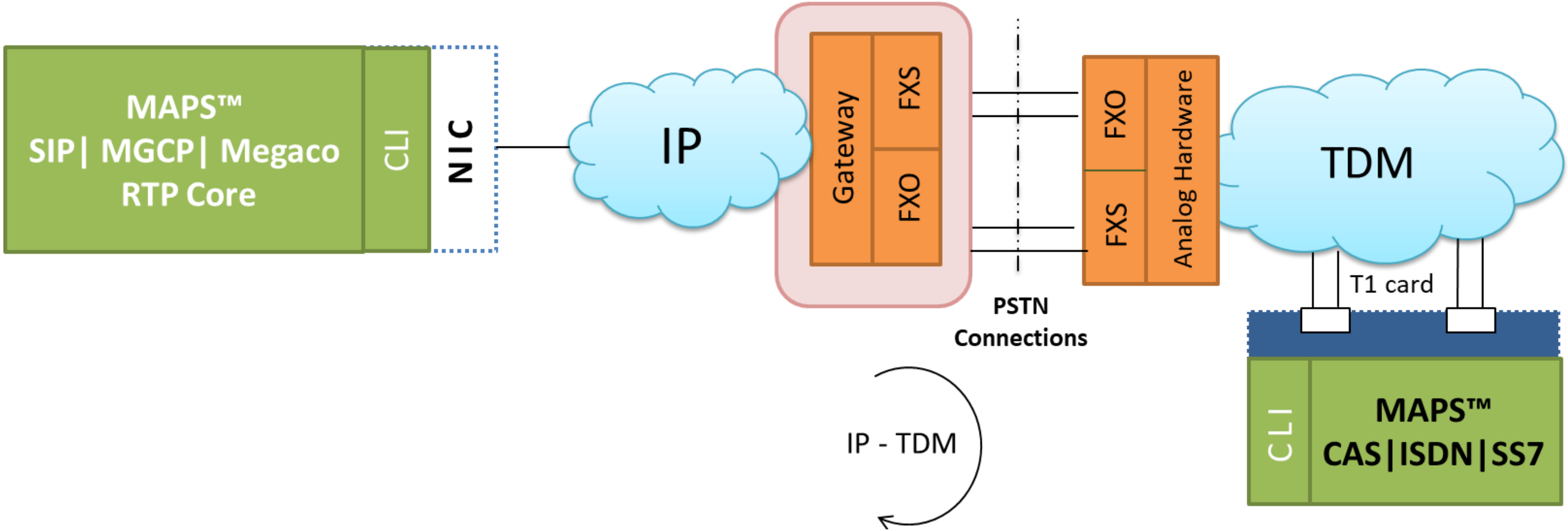
- This example shows a Python script that will iterate over every possible combination of values in the variable regression_table



```
regression_header = ['Contact', 'AddressOfRecord', 'To', 'CodecOption', 'Packetizationtime']
type_table = ['(s)', '(s)', '(s)', '(s)', '(s)']
regression_table = [['0001@192.168.30.213'],
                    ['0001@192.168.30.213'],
                    ['1000@192.168.30.159', '2000@192.168.30.159'],
                    ['Profile0001', 'Profile0003', 'Profile0005', 'Profile0006'],
                    ['10', '20', '30']
                    ]

uas = init_client(uas_ip, maps_port)
uac = init_client(uac_ip, maps_port)
if (uas.status == 'STARTED') and (uac.status == 'STARTED'):
    print "SERVERS INITIALIZED"
    uac_call_list = []
    uas_call_list = []

    for args in itertools.product(*regression_table):
        print args
        uac_call, uas_call = two_way_call(uac, uas, regression_header, type_table, args)
        uac_call_list.append(uac_call)
        uas_call_list.append(uas_call)
        print "Status::" + uac_call.status
        print "CMOS::" + str(uac_call.rtp_stats.cmos)
        print ''
```

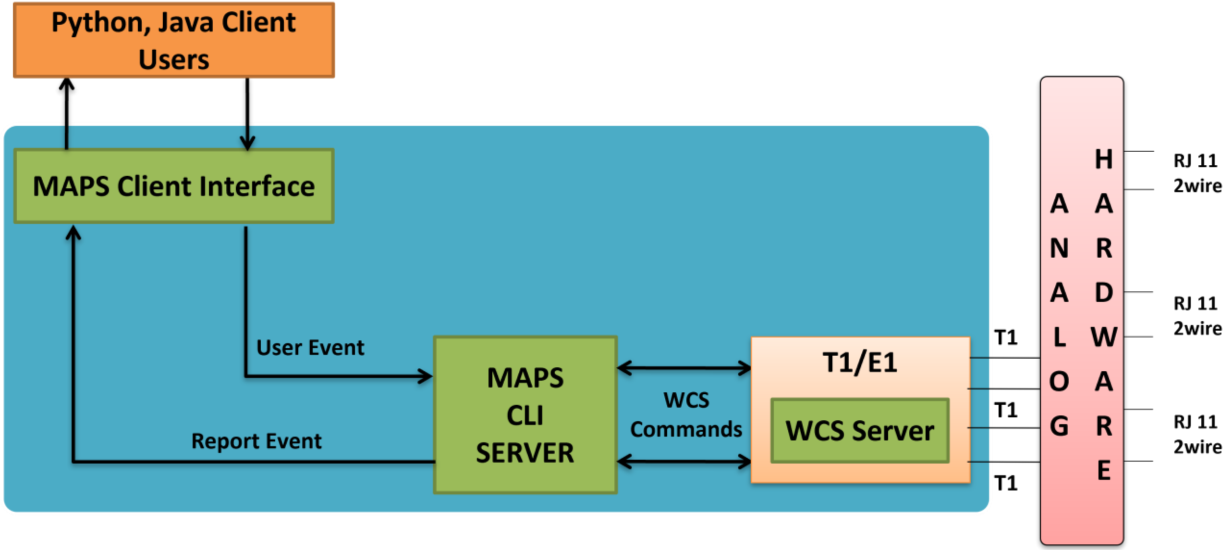
Typical Test Systems

Test Setup for Gateway Testing



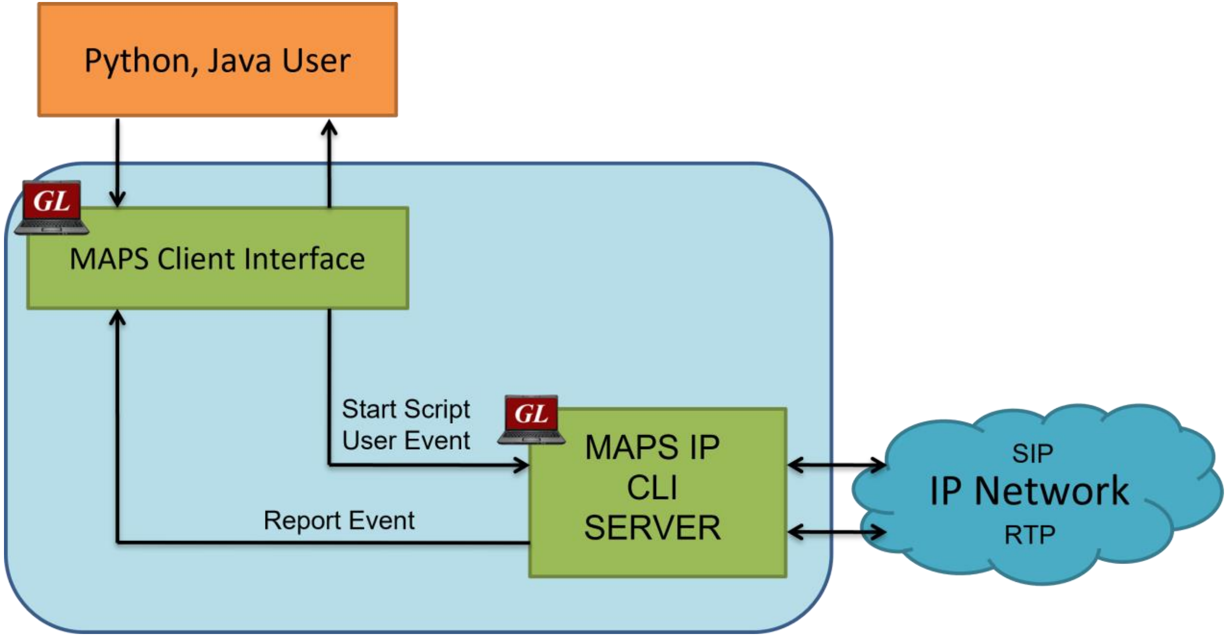
Typical MAPS™ CAS Test System

- Python/Java user communicating over TCP/IP
- MAPS™ Client IFC, MAPS™ CAS CLI Server, T1 Software (including Windows® Client Server software) and a Dual T1 Card
- Analog Hardware with FXO Cards
- A patch panel for RJ-11 connections to the outside world



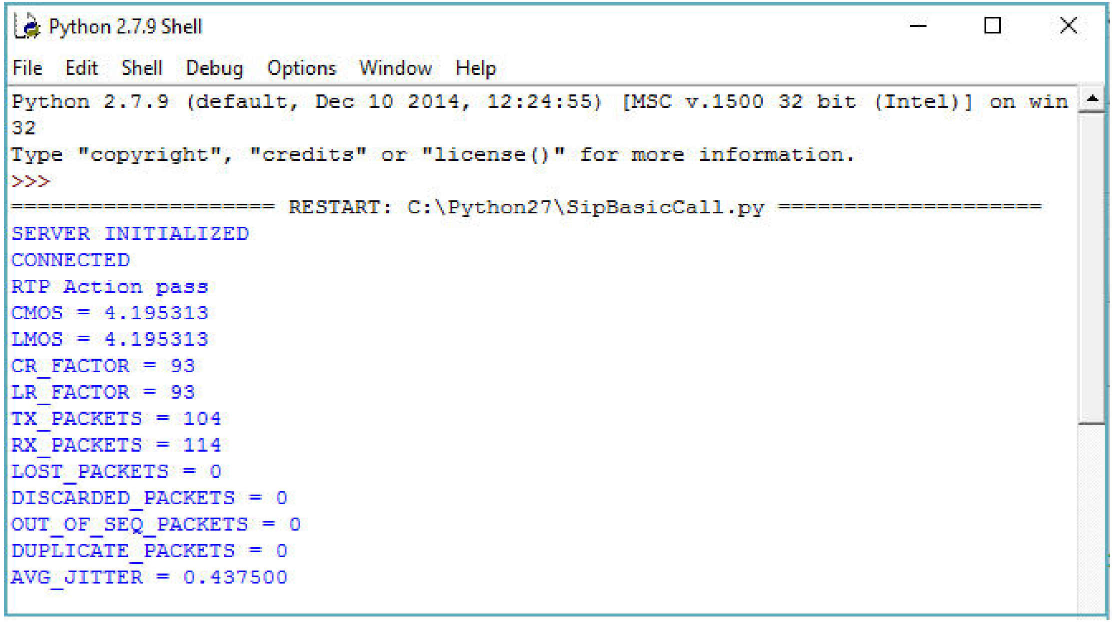
Typical MAPS™ SIP Test System

- Python/Java user communicating over TCP/IP
- MAPS™ Client IFC, and MAPS™ SIP CLI Server



Voice Quality Testing

- The MAPS™ API is also now fully integrated with GL’s VQT software which delivers PESQ/POLQA scores (i.e. waveform analysis, rather than packet analysis)



```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 
==================== RESTART: C:\Python27\SipBasicCall.py ====================
SERVER INITIALIZED
CONNECTED
RTP Action pass
CMOS = 4.195313
LMOS = 4.195313
CR_FACTOR = 93
LR_FACTOR = 93
TX_PACKETS = 104
RX_PACKETS = 114
LOST_PACKETS = 0
DISCARDED_PACKETS = 0
OUT_OF_SEQ_PACKETS = 0
DUPLICATE_PACKETS = 0
AVG_JITTER = 0.437500
```

Thank You