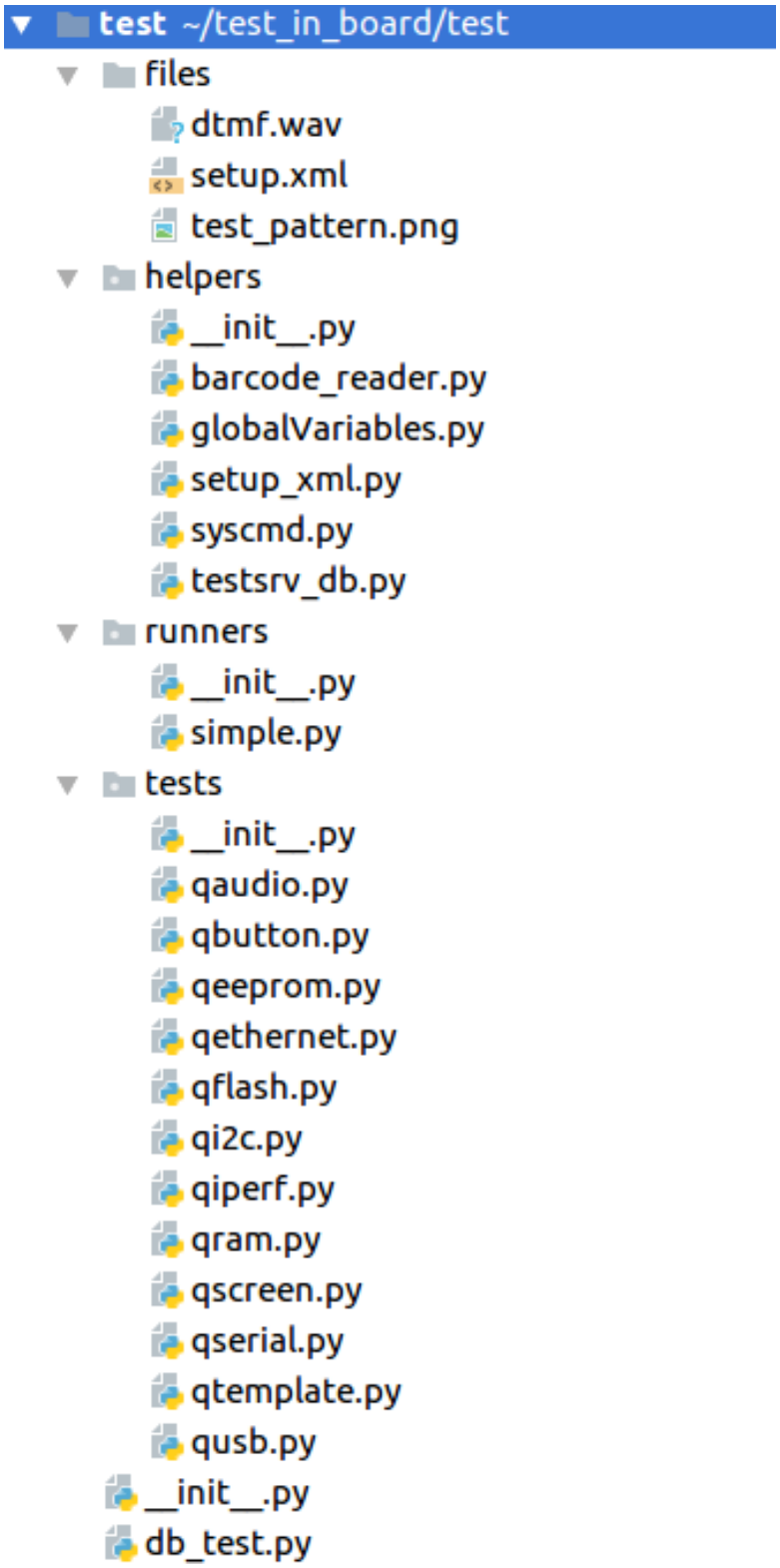
Annex 1: Python Code



File - /home/calonso/test_in_board/test/db_test.py

```
import os
import unittest

from test.helpers.globalVariables import globalVar
from test.runners.simple import SimpleTestRunner

from helpers.testsrv_db import TestSrv_Database


def create_board():
    psdb = TestSrv_Database()
    psdb.open("setup.xml")
    processor_id='123123123'
    model_id='IGEP0046'
    variant='DWE2-8SXX'
    globalVar.g_mid=model_id + "-" + variant
    print(globalVar.g_mid)
    globalVar.g_uuid = psdb.create_board(processor_id, model_id, variant, bmac = None)

def testsuite():
    psdb=TestSrv_Database()
    psdb.open("setup.xml")
    suite = unittest.TestSuite()
    tests=psdb.getboard_comp_test_list(globalVar.g_uuid)
    for i in range(len(tests)):
        #newstr = oldstr.replace("M", "")
        variables=str(tests[i][0]).split(",")
        testname=variables[0].replace('(', '')
        testdes=variables[1]
        testfunc=variables[2]
        if len(tests)>2:
            testparam=variables[3].replace(')', '')
            testparam = testparam.replace('"', '')
            testparam = testparam.replace(';', "','")
            command="suite.addTest({}('{}','execute','{}'))".format(testfunc, testname,testparam)
        else:
            command = "suite.addTest({}('{}','execute','{}'))".format(testfunc, testname)
        exec(command)
    globalVar.testid_ctl=psdb.open_testbatch(globalVar.g_uuid)
    return suite

def addtestdef():
    mid = "IGEP0046RA01"
    psdb = TestSrv_Database()
    psdb.open("setup.xml")
    #psdb.create_test_definition("TESTNAME", "DES", "FUNC")
    psdb.create_test_definition("ETHERNET", "Iperf eth test", "Qethernet")
    # psdb.add_testdef_to_group("MID", "TESTNAME", "PARAMETERS")
    psdb.add_testdef_to_group("IGEP0046RA01", "ETHERNET", "192.168.2.171;1000")
    #--------------------
    psdb.create_test_definition("RAMSIZE", "Ram total size check", "Qram")
    psdb.add_testdef_to_group("IGEP0046RA01", "RAMSIZE", "15000")
    # --------------------
    psdb.create_test_definition("AUDIO", "Audio loopback test", "Qaudio")
    psdb.add_testdef_to_group("IGEP0046RA01", "AUDIO", "/home/carlosa/Music/dtmf-13579.wav")
    # --------------------
    psdb.create_test_definition("USBHOST", "USB HOST test", "Qusb")
    psdb.add_testdef_to_group("IGEP0046RA01", "USBHOST", "TEST;2")


def addtesttomodel():
    psdb = TestSrv_Database()
```

```
63      psdb.open("setup.xml")
64      mid = "IGEP0046RA01"
65      psdb.add_testdef_to_group("IGEP0046RA01", "DMESG", "dmesg")
66      psdb.add_testdef_to_group("IGEP0046RA01", "CPUINFO", "cat /proc/cpuinfo"
    )
67
68  def finish_test():
69      psdb = TestSrv_Database()
70      psdb.open("setup.xml")
71      psdb.close_testbatch(globalVar.g_uuid, globalVar.testid_ctl)
72
73  def main():
74      #addtesttomodel()
75      #addtestdef()
76      create_board()
77      #globalVar.g_uuid = "1f59c654-0cc6-11e8-8d51-e644f56b8edd"
78      try:
79          os.remove("test_results.dat")
80      except:
81          print("Cant't remove file")
82      runner = SimpleTestRunner()
83      runner.run(testsuite())
84      finish_test()
85
86
87  if __name__ == "__main__":
88      main()
89
```

File - /home/calonso/test_in_board/test/files/setup.xml

```xml
<?xml version="1.0"?>
<data>
    <setup>
        <test idline="1"/> <!-- Test line identify -->
        <db dbname="testsrv" type="PgSQLConnection" host="192.168.2.1" port="5432" user="user" password="password" /> <!-- database setup -->
    </setup>
</data>
```

File - /home/calonso/test_in_board/test/tests/qi2c.py

```
 1 from test.helpers.syscmd import SysCommand
 2 import unittest
 3
 4 class Qi2c(unittest.TestCase):
 5
 6     def __init__(self, testname, testfunc, busnum, register):
 7         super(Qi2c, self).__init__(testfunc)
 8         self.__busnum = busnum
 9         self.__register = register.split("/")
10         self.__devices=[]
11         self._testMethodDoc = testname
12
13     def execute(self):
14         str_cmd= "i2cdetect -a -y -r {}".format(self.__busnum)
15         i2c_command = SysCommand("i2cdetect", str_cmd)
16         if i2c_command.execute() == 0:
17             self.__raw_out = i2c_command.getOutput()
18             if self.__raw_out == "":
19                 return -1
20             lines=self.__raw_out.decode('ascii').splitlines()
21             for i in range(len(lines)):
22                 if (lines[i].count('UU')):
23                     if (lines[i].find("UU")):
24                         self.__devices.append("0x{}{}".format((i - 1), hex(
   int((lines[i].find("UU") - 4) / 3)).split('x')[-1]))
25             for i in range(len(self.__register)):
26                 if not(self.__register[i] in self.__devices):
27                     self.fail("failed: device {} not found in bus i2c-{}".
   format(self.__register[i], self.__busnum))
28         else:
29             self.fail("failed: could not complete i2cdedtect command")
30
```

File - /home/calonso/test_in_board/test/tests/qram.py

```
1  from test.helpers.syscmd import SysCommand
2  import unittest
3
4  class Qram(unittest.TestCase):
5
6      def __init__(self, testname, testfunc, memSize):
7          super(Qram, self).__init__(testfunc)
8          self.__memSize = memSize
9          self._testMethodDoc = testname
10
11     def execute(self):
12         str_cmd= "free -m"
13         free_command = SysCommand("free_ram", str_cmd)
14         if free_command.execute() == 0:
15             self.__raw_out = free_command.getOutput()
16             if self.__raw_out == "":
17                 return -1
18             lines = free_command.getOutput().splitlines()
19             aux = [int(s) for s in lines[1].split() if s.isdigit()]
20             self.failUnless(int(aux[0])>int(self.__memSize),"failed: total
   ram memory size lower than expected")
21         else:
22             self.fail("failed: could not complete iperf command")
23
```

File - /home/calonso/test_in_board/test/tests/qusb.py

```
from test.helpers.syscmd import SysCommand
import unittest

class Qusb(unittest.TestCase):

    def __init__(self, testname, testfunc, devLabel, numPorts):
        super(Qusb, self).__init__(testfunc)
        self.__numPorts = numPorts
        self._testMethodDoc = testname
        self.__devLabel = devLabel
        if testname=="USBOTG":
            self.__usbFileName = "/this_is_an_usb_otg"
            self.__usbtext = "USBOTG"
        else:
            self.__usbFileName = "/this_is_an_usb_host"
            self.__usbtext = "USBHOST"
        self.__numUsbFail=[]

    def execute(self):
        str_cmd= "lsblk -o LABEL"
        lsblk_command = SysCommand("lsblk", str_cmd)
        if lsblk_command.execute() == 0:
            self.__raw_out = lsblk_command.getOutput()
            if self.__raw_out == "":
                return -1
            lines = lsblk_command.getOutput().splitlines()
            host_list=[]
            for i in range(len(lines)):
                if str(lines[i].decode('ascii'))==self.__devLabel:
                    host_list.append(i)
            if len(host_list)==int(self.__numPorts):
                str_cmd = "lsblk -o MOUNTPOINT"
                lsblk_command = SysCommand("lsblk", str_cmd)
                if lsblk_command.execute() == 0:
                    self.__raw_out = lsblk_command.getOutput()
                    if self.__raw_out == "":
                        print("failed: no command output")
                        self.fail("failed: no command output")
                    else:
                        lines = lsblk_command.getOutput().splitlines()
                        for i in range(len(host_list)):
                            file_path=str(lines[host_list[i]].decode('ascii')
) + self.__usbFileName
                            usb_file = open(file_path, 'r')
                            read=usb_file.read()
                            if read.find(self.__usbtext)!=-1:
                                print(file_path + "  -->  OK!")
                            else:
                                self.fail("failed: could not read from usb
{}".format(lines[host_list[i]].decode('ascii')))
                                self.__numUsbFail.append(host_list[i])
                            usb_file.close()
                else:
                    self.fail("failed: couldn't execute lsblk command")

            else:
                self.fail("failed: reference and real usb host devices number
mismatch")
        else:
            self.fail("failed: couldn't execute lsblk command")
```

File - /home/calonso/test_in_board/test/tests/qaudio.py

```
1  from test.helpers.syscmd import SysCommand
2  import unittest
3  #class name
4  class Qaudio(unittest.TestCase):
5      # Initialize the variables
6
7      def __init__(self, testname, testfunc, dtmfFile):
8          # Doing this we will initialize the class and later on perform a
   particular method inside this class
9          super(Qaudio, self).__init__(testfunc)
10         self._testMethodDoc = testname
11         self._dtmfFile=dtmfFile
12         self.__sum=0
13         self.__refSum = 25 # 1+3+5+7+9
14
15     def execute(self):
16         str_cmd = "aplay test/files/dtmf-13579.wav & arecord -r 8000 -d 1
   recorded.wav" #.format(self.__dtmfFile)
17         audio_loop = SysCommand("command-name", str_cmd)
18         if audio_loop.execute() == -1:# BUG: Returns -1 but work
19             lines = audio_loop.getOutput().splitlines()
20             str_cmd = "multimon -t wav -a DTMF recorded.wav -q"
21             dtmf_decoder = SysCommand("command-name", str_cmd)
22             a=dtmf_decoder.execute()
23             if dtmf_decoder.execute() == -1: # BUG: Returns -1 but work
24                 self.__raw_out = dtmf_decoder.getOutput()
25                 if self.__raw_out == "":
26                     return -1
27                 lines = dtmf_decoder.getOutput().splitlines()
28                 for i in range(0, 5):
29                     aux=[int(s) for s in lines[i].split() if s.isdigit()]
30                     self.__sum=self.__sum+aux[0]
31                 self.failUnless(self.__sum == self.__refSum), "failed:
   incorrect dtmf code" + str(self.__sum)
32             else:
33                 self.fail("failed: fail reading recorded file")
34                 return -1
35         else:
36             self.fail("failed: fail playing/recording file")
37             return -1
38         return 0
```

File - /home/calonso/test_in_board/test/tests/qflash.py

```
1  from test.helpers.syscmd import SysCommand
2  import unittest
3  from test.helpers.globalVariables import globalVar
4
5  class Qflasher(unittest.TestCase):
6
7      def __init__(self, testname, testfunc, busnum, register):
8          self.__busnum = busnum
9          self.__register = register.split("/")
10         self.__devices = []
11         self._testMethodDoc = testname
12         model=globalVar.g_mid
13         if model.find("IGEP0046") == 0:
14             flash_method = "mx6"
15         elif model.find("IGEP0034") == 0:
16             flash_method = "nandti"
17         elif model.find("OMAP3") == 0:
18             flash_method = "nandti"
19         elif model.find("OMAP5") == 0:
20             flash_method = "nandti"
21
22     def mx6(self):
23         str_cmd= "i2cdetect -a -y -r {}".format(self.__busnum)
24         flash_command = SysCommand("flash_command", str_cmd)
25         if flash_command.execute() == 0:
26             print("Flashed EMMC")
27         else:
28             self.fail("failed: could not complete flash eMMC commands")
29
30     def nandti(self):
31         str_cmd= "i2cdetect -a -y -r {}".format(self.__busnum)
32         flash_command = SysCommand("flash_command", str_cmd)
33         if flash_command.execute() == 0:
34             print("Flashed NAND")
35         else:
36             self.fail("failed: could not complete flash NAND commnds")
37
```

File - /home/calonso/test_in_board/test/tests/qiperf.py

```
1  from test.helpers.syscmd import SysCommand
2
3
4  class QIperf(object):
5      __sip = None
6      __raw_out = None
7      __MB_req = None
8      __MB_real = None
9      __BW_real = None
10     __dat_list = None
11     __bind = None
12
13     def __init__(self, sip = None):
14         if sip is not None:
15             self.__sip = sip
16         self.__MB_req = '10'
17
18     def execute(self, sip = None, bind = None):
19         if sip is not None:
20             self.__sip = sip
21
22         if bind is None:
23             str_cmd = "iperf -c {} -x CMSV -n {}M".format(self.__sip, self.
   __MB_req)
24         else:
25             self.__bind = bind
26             str_cmd = "iperf -c {} -x CMSV -n {}M -B {}".format(self.__sip,
   self.__MB_req, self.__bind)
27         t = SysCommand("iperf", str_cmd)
28         if t.execute() == 0:
29             self.__raw_out = t.getOutput()
30             if self.__raw_out == "":
31                 return -1
32             lines = t.getOutput().splitlines()
33             dat = lines[1]
34             dat = dat.decode('ascii')
35             dat_list = dat.split( )
36             for d in dat_list:
37                 a = dat_list.pop(0)
38                 if a == "sec":
39                     break
40             self.__MB_real = dat_list[0]
41             self.__BW_real = dat_list[2]
42             self.__dat_list = dat_list
43             print(self.__MB_real)
44             print(self.__BW_real)
45         else:
46             return -1
47         return 0
48
49     def get_Total_MB(self):
50         return self.__MB_real;
51
52     def get_Total_BW(self):
53         return self.__MB_real;
54
```

File - /home/calonso/test_in_board/test/tests/qbutton.py

```
#!/usr/bin/env python

"""
User button Test Cases modules for unittest

"""

import unittest

class TestButton(unittest.TestCase):
    """ Generic test for user button.

    Keyword arguments:
        - gpio_in: GPIO input mapped to user button.
    """
    def __init__(self, testname, gpio_in):
        """ init """
        super(TestButton, self).__init__(testname)

    def test_button(self):
        """ aaaa bbbb"""
        self.assertTrue(False)

    def test_button2(self):
        """ ccc ddd"""
        self.assertTrue(True)


#if __name__ == '__main__':
#    unittest.main(verbosity=2)

```

File - /home/calonso/test_in_board/test/tests/qeeprom.py

```
1  from test.helpers.syscmd import SysCommand
2  import unittest
3  import uuid
4
5  class Qeeprom(unittest.TestCase):
6
7      def __init__(self, testname, testfunc):
8          super(Qeeprom, self).__init__(testfunc)
9          self._testMethodDoc = testname
10
11     def execute(self):
12         str_cmd = "find /sys/ -iname 'eeprom'"
13         eeprom_location = SysCommand("eeprom_location", str_cmd)
14         if eeprom_location.execute() == 0:
15             self.__raw_out = eeprom_location.getOutput()
16             if self.__raw_out == "":
17                 self.fail("Unable to get EEPROM location. IS EEPROM CONNECTED
   ?")
18                 return -1
19             eeprom=self.__raw_out.decode('ascii')
20             test_uuid = uuid.uuid4()
21             str_cmd="echo '{}' > {}".format(str(test_uuid), eeprom)
22             eeprom_write = SysCommand("eeprom_write", str_cmd)
23             if eeprom_write.execute() == 0:
24                 self.__raw_out = eeprom_write.getOutput()
25                 if self.__raw_out == "":
26                     self.fail("Unable to write on the EEPROM?")
27                     return -1
28                 str_cmd = "head -2 {}".format(eeprom)
29                 eeprom_read = SysCommand("eeprom_read", str_cmd)
30                 if eeprom_read.execute() == 0:
31                     self.__raw_out = eeprom_read.getOutput()
32                     if self.__raw_out == "":
33                         self.fail("Unable to read from the EEPROM?")
34                         return -1
35                     if(str(self.__raw_out).find(str(test_uuid)) == -1):
36                         self.fail("failed: READ/WRITE mismatch")
37         else:
38             self.fail("failed: could not complete find eeprom command")
39
```

File - /home/calonso/test_in_board/test/tests/qscreen.py

```
 1 from test.helpers.syscmd import SysCommand
 2 import unittest
 3 import time
 4 from test.helpers.cv_display_test import pattern_detect
 5
 6 class Qscreen(unittest.TestCase):
 7
 8     def __init__(self, testname, testfunc, display):
 9         super(Qscreen, self).__init__(testfunc)
10         self.__display = display
11         self._testMethodDoc = testname
12
13     def execute(self):
14         str_cmd = "fbi -T 1 --noverbose -d /dev/{} test/files/test_pattern.
   png".format(self.__display)
15         display_image = SysCommand("display_image", str_cmd)
16         if display_image.execute() == -1:
17             test_screen = pattern_detect(1)
18             if not test_screen=="0":
19                 self.fail("failed: {}".format(test_screen))
20         else:
21             self.fail("failed: could not display the image")
22
```

File - /home/calonso/test_in_board/test/tests/qserial.py

```
1  from test.helpers.syscmd import SysCommand
2  import unittest
3  import uuid
4  import serial
5  import time
6
7  class Qserial(unittest.TestCase):
8
9      def __init__(self, testname, testfunc, port, baudrate):
10         super(Qserial, self).__init__(testfunc)
11         self.__port = port
12         self.__serial = serial.Serial(self.__port, timeout=1)
13         self.__serial.baudrate = baudrate
14         self._testMethodDoc = testname
15
16     def __del__(self):
17         self.__serial.close()
18
19     def execute(self):
20         self.__serial.flushInput()
21         self.__serial.flushOutput()
22         test_uuid = str(uuid.uuid4()).encode()
23         self.__serial.write(test_uuid)
24         time.sleep(0.05)  # there might be a small delay
25         if self.__serial.inWaiting() == 0:
26             self.fail("failed: PORT {} wait timeout exceded, wrong
   communication?".format(self.__port))
27         else:
28             if (self.__serial.readline() != test_uuid):
29                 self.fail("failed: PORT {} write/read mismatch".format(self.
   __port))
30
```

File - /home/calonso/test_in_board/test/tests/qethernet.py

```
1  from test.helpers.syscmd import SysCommand
2  import unittest
3
4
5  class Qethernet(unittest.TestCase):
6      __sip = None
7      __raw_out = None
8      __MB_req = None
9      __MB_real = None
10     __BW_real = None
11     __dat_list = None
12     __bind = None
13     __OKBW = None
14
15     def __init__(self, testname, testfunc, sip = None, OKBW=100, bind=None):
16         super(Qethernet, self).__init__(testfunc)
17         if sip is not None:
18             self.__sip = sip
19         if sip is not None:
20             self.__bind = bind
21         self.__MB_req = '10'
22         self.__OKBW=OKBW
23         self._testMethodDoc = testname
24
25     def execute(self):
26         print
27         if self.__bind is None:
28             str_cmd = "iperf -c {} -x CMSV -n {}M".format(self.__sip, self.
   __MB_req)
29         else:
30             str_cmd = "iperf -c {} -x CMSV -n {}M -B {}".format(self.__sip,
   self.__MB_req, self.__bind)
31         iperf_command = SysCommand("iperf", str_cmd)
32         if iperf_command.execute() == 0:
33             self.__raw_out = iperf_command.getOutput()
34             if self.__raw_out == "":
35                 return -1
36             lines = iperf_command.getOutput().splitlines()
37             dat = lines[1]
38             dat = dat.decode('ascii')
39             dat_list = dat.split( )
40             for d in dat_list:
41                 a = dat_list.pop(0)
42                 if a == "sec":
43                     break
44             self.__MB_real = dat_list[0]
45             self.__BW_real = dat_list[2]
46             self.__dat_list = dat_list
47             #print(self.__MB_real)
48             #print(self.__BW_real)
49             self.failUnless(float(self.__BW_real)>float(self.__OKBW)*0.9,"
   failed: speed is lower than spected. Speed(MB/s)" + str(self.__BW_real))
50         else:
51             self.fail("failed: could not complete iperf command")
52
53     def get_Total_MB(self):
54         return self.__MB_real;
55
56     def get_Total_BW(self):
57         return self.__MB_real;
58
```

File - /home/calonso/test_in_board/test/tests/qtemplate.py

```
#IF COMMAND IS NEEDED
from test.helpers.syscmd import SysCommand
import unittest
#class name
class Qtemplate(unittest.TestCase):
    # Initialize the variables
    __variable1 = "Value-a"
    __variable2 = "Value-b"
    #....
    __variablen = "Value-n"

    def __init__(self, testname, testfunc, input1=None, inputn=None):
        # Doing this we will initialize the class and later on perform a particular method inside this class
        super(Qtemplate, self).__init__(testfunc)
        self.__testname = testname
        self.__input1 = input1
        self.__inputn = inputn
        self._testMethodDoc = testname



    def execute(self):
        str_cmd = "command"
        t = SysCommand("command-name", str_cmd)
        if t.execute() == 0:
            self.__raw_out = t.getOutput()
            if self.__raw_out == "":
                return -1
            lines = t.getOutput().splitlines()
            dat = lines[1]
            dat = dat.decode('ascii')
            dat_list = dat.split( )
            for d in dat_list:
                a = dat_list.pop(0)
                if a == "sec":
                    break
            self.__MB_real = dat_list[0]
            self.__BW_real = dat_list[2]
            self.__dat_list = dat_list
            print(self.__MB_real)
            print(self.__BW_real)
            self.failUnless(int(self.__BW_real)>int(self.__OKBW)*0.9,"FAIL: BECAUSE...")
        else:
            return -1
        return 0

    def get_Total_MB(self):
        return self.__MB_real;

    def get_Total_BW(self):
        return self.__MB_real;

```

File - /home/calonso/test_in_board/test/helpers/syscmd.py

```
import unittest
import subprocess
from test.helpers.globalVariables import globalVar


class TestSysCommand(unittest.TestCase):
    __str_cmd = None
    __testname = None
    __outfilename = None
    __outdata = None
    __outtofile = False

    def __init__(self, testname, testfunc, str_cmd, outtofile = False):
        """ init """
        super(TestSysCommand, self).__init__(testfunc)
        self.__str_cmd = str_cmd
        self.__testname = testname
        self.__outtofile = outtofile
        self._testMethodDoc = testname
        if self.__outtofile is True:
            self.__outfilename = '/tmp/{}.txt'.format(testname)

    def getName(self):
        return self.__testname

    def execute(self):
        res = -1
        try:
            completed = subprocess.run(
                self.__str_cmd,
                check=True,
                shell=True,
                stdout=subprocess.PIPE,
            )
            self.assertTrue(completed.returncode is 0)
            if completed.returncode is 0:
                if self.__outtofile is True:
                    f = open(self.__outfilename, 'wb')
                    f.write(completed.stdout)
                    f.close()
                res = 0
            else:
                res = -3
            outdata = completed.stdout
            self.longMessage=str(outdata).replace("'","")
            self.assertTrue(True)
        except subprocess.CalledProcessError as err:
            self.assertTrue(False)
            res = -1
        except Exception as t:
            res = -2
        return res

    def remove_file(self):
        pass

class SysCommand(object):
    __str_cmd = None
    __cmdname = None
    __outdata = None
    __errdata = None

    def __init__(self, cmdname, str_cmd):
        """ init """
        self.__str_cmd = str_cmd
        self.__cmdname = cmdname

```

```
68      def getName(self):
69          return self.__testname
70
71      def execute(self):
72          res = -1
73          try:
74              self.__outdata = None
75              self.__errdata = None
76              completed = subprocess.run(
77                  self.__str_cmd,
78                  check=True,
79                  shell=True,
80                  stdout=subprocess.PIPE,
81                  stderr=subprocess.PIPE
82              )
83              self.__outdata = completed.stdout
84              if completed.returncode is 0:
85                  res = 0
86                  if completed.stderr.decode('ascii') != "":
87                      res = -1
88                      self.__errdata = completed.stderr
89          except subprocess.CalledProcessError as err:
90              res = -2
91          except Exception as t:
92              res = -3
93          return res
94
95      def getOutput(self):
96          return self.__outdata
97
98      def getOutErr(self):
99          return self.__errdata
100
101     def getOutputlines(self):
102         return self.__outdata.splitlines()
103
104     def save_file(self, fname):
105         f = open(fname, 'wb')
106         f.write(self.__outdata)
107         f.close()
108
109
```

File - /home/calonso/test_in_board/test/helpers/setup_xml.py

```
1  import xml.etree.ElementTree as XMLParser
2
3  class XMLSetup (object):
4      """aaaaa"""
5      __tree = None                    # Parser
6      __dbType = None                  # database connection required:
   PgSQLConnection
7      __dbConnectionRaw = None         # Connection string in raw
8      __dbConnectionStr = None         # Connection string to use in sql object
   connection
9
10     def __init__(self, filename):
11         """aaaaa"""
12         self.__tree = XMLParser.parse(filename)
13
14     def __del__(self):
15         """aaaaa"""
16         pass
17
18     def getdbConnectionStr (self):
19         """aaaaa"""
20         if self.__dbConnectionRaw is not None:
21             return self.__dbConnectionRaw
22
23         for element in self.__tree.iter('db'):
24             self.__dbConnectionRaw = element.attrib
25             self.__dbType = self.__dbConnectionRaw['type']
26             if self.__dbType == "PgSQLConnection":
27                 self.__dbConnectionStr = self.getPostgresConnectionStr()
28             return self.__dbConnectionStr
29
30         return None
31
32     def getPostgresConnectionStr (self):
33         """aaaaa"""
34         str = self.__dbConnectionRaw
35         del str['type']
36         return str
37
38     def getMysqlConnectionStr (self):
39         """aaaaa"""
40         pass
```

File - /home/calonso/test_in_board/test/helpers/testsrv_db.py

```
 1 from psqldb import PgSQLConnection
 2 from setup_xml import XMLSetup
 3
 4 def find_between( s, first, last ):
 5     try:
 6         start = s.index( first ) + len( first )
 7         end = s.index( last, start )
 8         return s[start:end]
 9     except ValueError:
10         return ""
11
12
13 class TestSrv_Database(object):
14     ''' TestSrv Database Helper '''
15
16     __sqlObject = None
17     __xml_setup = None
18
19     def __init__(self):
20         pass
21
22     def open (self, filename):
23         '''Open database connection'''
24         self.__xml_setup = XMLSetup(filename)
25         self.__sqlObject = PgSQLConnection()
26         return self.__sqlObject.db_connect(self.__xml_setup.
   getdbConnectionStr())
27
28     def create_board(self, processor_id, model_id, variant, bmac = None):
29         '''create a new board'''
30         if bmac is None:
31             sql = "SELECT isee.create_board('{}', '{}', '{}', NULL);".format(
   processor_id, model_id, variant)
32         else:
33             sql = "SELECT isee.create_board('{}', '{}', '{}', '{}');".format(
   processor_id, model_id, variant, bmac)
34         print('>>>' + sql)
35         try:
36             res = self.__sqlObject.db_execute_query(sql)
37             print(res)
38             return res[0][0];
39         except Exception as err:
40             r = find_between(str(err), '#', '#')
41             print(r)
42         return None
43
44     def create_model(self, modid, variant, descr, tgid):
45         '''create new model'''
46         sql = "SELECT isee.create_model('{}', '{}', '{}', '{}')".format(modid
   , variant, descr, tgid)
47         print('>>>' + sql)
48         try:
49             res = self.__sqlObject.db_execute_query(sql)
50             print(res)
51             return res[0][0];
52         except Exception as err:
53             r = find_between(str(err), '#', '#')
54             print(r)
55         return None
56
57     def create_test_definition(self, testname, testdesc, testfunc):
58         '''Create a new definition and return definition id on fail (testname
    already exist) return -1'''
59         sql = "SELECT isee.define_test('{}', '{}', '{}')".format(testname,
   testdesc, testfunc)
60         print('>>>' + sql)
61         try:
```

```
62              res = self.__sqlObject.db_execute_query(sql)
63              print(res)
64              return res[0][0];
65          except Exception as err:
66              r = find_between(str(err), '#', '#')
67              print(r)
68          return None
69
70      def add_testdef_to_group(self, testgroupid, testname, testparam):
71          '''Assign definition to group test return  true on success or false
    if it fails'''
72          sql = "SELECT isee.add_test_to_group('{}', '{}', '{}')".format(
    testgroupid, testname, testparam)
73          print('>>>' + sql)
74          try:
75              res = self.__sqlObject.db_execute_query(sql)
76              print(res)
77              return res[0][0];
78          except Exception as err:
79              r = find_between(str(err), '#', '#')
80              print(r)
81          return None
82
83      def getboard_test_list(self, board_uuid):
84          '''get the board test list'''
85          sql = "SELECT isee.gettestlist('{}')".format(board_uuid)
86          print('>>>' + sql)
87          try:
88              res = self.__sqlObject.db_execute_query(sql)
89              print(res)
90              return res;
91          except Exception as err:
92              r = find_between(str(err), '#', '#')
93              print(r)
94          return None
95
96      def getboard_comp_test_list(self, board_uuid):
97          '''get the board test list'''
98          sql = "SELECT isee.gettestcompletelist('{}')".format(board_uuid)
99          print('>>>' + sql)
100         try:
101             res = self.__sqlObject.db_execute_query(sql)
102             print(res)
103             return res;
104         except Exception as err:
105             r = find_between(str(err), '#', '#')
106             print(r)
107         return None
108
109     def open_testbatch(self, board_uuid):
110         '''get the board test list'''
111         sql = "SELECT isee.open_testbatch('{}')".format(board_uuid)
112         print('>>>' + sql)
113         try:
114             res = str(self.__sqlObject.db_execute_query(sql)[0])
115             res = res.replace('(', '')
116             res = res.replace(')', '')
117             res = res.replace(',', '')
118             print(res)
119             return res;
120         except Exception as err:
121             r = find_between(str(err), '#', '#')
122             print(r)
123         return None
124
125     def add_test_to_batch(self, board_uuid, testid, testid_ctl, result, data
    ):
```

```
126         '''get the board test list'''
127         sql = "SELECT isee.add_test_to_batch_c('{}','{}','{}','{}','{}')".
    format(board_uuid, testid, testid_ctl, result, data)
128         print('>>>' + sql)
129         try:
130             res = self.__sqlObject.db_execute_query(sql)
131             print(res)
132             return res;
133         except Exception as err:
134             r = find_between(str(err), '#', '#')
135             print(r)
136         return None
137 
138     def close_testbatch(self, board_uuid, testid_ctl):
139         '''get the board test list'''
140         sql = "SELECT isee.close_testbatch('{}','{}')".format(board_uuid,
    testid_ctl)
141         print('>>>' + sql)
142         try:
143             res = self.__sqlObject.db_execute_query(sql)
144             print(res)
145             return res;
146         except Exception as err:
147             r = find_between(str(err), '#', '#')
148             print(r)
149         return None
150 
```

File - /home/calonso/test_in_board/test/helpers/barcode_reader.py

```
#!/usr/bin/python
import evbc_reader
from evbc_reader import Inputbc_readerice, categorize, ecodes
bc_reader = Inputbc_readerice('/bc_reader/input/by-id/usb-Manufacturer_Barcode_Reader-event-kbd')

# ASCII to character. If u before it is character if not it is action
normal_codes = {
    # Scancode: ASCIICode
    0: None, 1: u'ESC', 2: u'1', 3: u'2', 4: u'3', 5: u'4', 6: u'5', 7: u'6', 8: u'7', 9: u'8',
    10: u'9', 11: u'0', 12: u'-', 13: u'=', 14: u'BKSP', 15: u'TAB', 16: u'q', 17: u'w', 18: u'e', 19: u'r',
    20: u't', 21: u'y', 22: u'u', 23: u'i', 24: u'o', 25: u'p', 26: u'[', 27: u']', 28: u'CRLF', 29: u'LCTRL',
    30: u'a', 31: u's', 32: u'd', 33: u'f', 34: u'g', 35: u'h', 36: u'j', 37: u'k', 38: u'l', 39: u';',
    40: u'"', 41: u'`', 42: u'LSHFT', 43: u'\\', 44: u'z', 45: u'x', 46: u'c', 47: u'v', 48: u'b', 49: u'n',
    50: u'm', 51: u',', 52: u'.', 53: u'/', 54: u'RSHFT', 56: u'LALT', 57: u' ', 100: u'RALT'
}

capital_codes = {
    0: None, 1: u'ESC', 2: u'!', 3: u'@', 4: u'#', 5: u'$', 6: u'%', 7: u'^', 8: u'&', 9: u'*',
    10: u'(', 11: u')', 12: u'_', 13: u'+', 14: u'BKSP', 15: u'TAB', 16: u'Q', 17: u'W', 18: u'E', 19: u'R',
    20: u'T', 21: u'Y', 22: u'U', 23: u'I', 24: u'O', 25: u'P', 26: u'{', 27: u'}', 28: u'CRLF', 29: u'LCTRL',
    30: u'A', 31: u'S', 32: u'D', 33: u'F', 34: u'G', 35: u'H', 36: u'J', 37: u'K', 38: u'L', 39: u':',
    40: u'\'', 41: u'~', 42: u'LSHFT', 43: u'|', 44: u'Z', 45: u'X', 46: u'C', 47: u'V', 48: u'B', 49: u'N',
    50: u'M', 51: u'<', 52: u'>', 53: u'?', 54: u'RSHFT', 56: u'LALT',  57: u' ', 100: u'RALT'
}

result = ''
cap_active = False # Initial state by default

#grab provides exclusive access to the bc_readerice
bc_reader.grab()
print("SCAN YOUR CODE...")
#loop
for event in bc_reader.read_loop(): #Read all the event
    if event.type == ecodes.EV_KEY:
        raw_data = categorize(event)  # Save the event temporarily to introspect it
        if raw_data.scancode == 42:
            # Change bt normal and capital letters
            if raw_data.keystate == 1:
                cap_active = True
            if raw_data.keystate == 0:
                cap_active = False
        if raw_data.keystate == 1:
            # Decode each KEY event
            if cap_active:
                key_lookup = u'{}'.format(capital_codes.get(raw_data.scancode)) or u'UNKNOWN:[{}]'.format(raw_data.scancode)  # Lookup or return UNKNOWN:XX
            else:
                key_lookup = u'{}'.format(normal_codes.get(raw_data.scancode)) or u'UNKNOWN:[{}]'.format(raw_data.scancode)  # Lookup or return UNKNOWN:XX
            if (raw_data.scancode != 42) and (raw_data.scancode != 28) and (raw_data.scancode != 0):
                x += key_lookup
```

File - /home/calonso/test_in_board/test/helpers/barcode_reader.py

```
50              if(raw_data.scancode == 28): #CRLF is the end of the code
51                  print("CODE: \t" + result + "\n\n")          # Result code
52                  break # exit the for loop
53
54
55
56
```

File - /home/calonso/test_in_board/test/helpers/globalVariables.py

```
1
2 def globalVar():
3     guuid=""
4     procid = ""
5     mid = ""
6     testid_ctl = ""
7     outdata = "NONE"
```

File - /home/calonso/test_in_board/test/runners/simple.py

```
#!/usr/bin/env python

"""
Simple Test Runner for unittest module

"""

import sys
import unittest

from test.helpers.globalVariables import globalVar

from helpers.testsrv_db import TestSrv_Database


class SimpleTestRunner:
    """ A Test Runner that shows results in a simple human-readable format.

    As example, a common output is:
        This is a test short description : PASS
        This is another test short description : FAIL
        ----------------------------------------------

    """
    def __init__(self, stream=sys.stderr, verbosity=0):
        self.stream = stream
        self.verbosity = verbosity

    def writeUpdate(self, message):
        self.stream.write(message)

    def run(self, test):
        """ Run the given test case or Test Suite.

        """
        result = TextTestResult(self)
        test(result)
        result.testsRun
        # self.writeUpdate("----------------------------------------------\n")
        return result

class TextTestResult(unittest.TestResult):
    # Print in terminal with colors
    PASS = '\033[32mPASS\033[0m\n'
    FAIL = '\033[31mFAIL\033[0m\n'
    ERROR = '\033[31mERROR\033[0m\n'

    def __init__(self, runner):
        unittest.TestResult.__init__(self)
        self.runner = runner
        self.result = self.ERROR

    def startTest(self, test):
        unittest.TestResult.startTest(self, test)
        self.runner.writeUpdate("%s : " % test.shortDescription())

    def addSuccess(self, test):
        unittest.TestResult.addSuccess(self, test)
        self.result=self.PASS

    def addError(self, test, err):
        unittest.TestResult.addError(self, test, err)
        test.longMessage = err[1]
        self.result = self.ERROR

    def addFailure(self, test, err):
        unittest.TestResult.addFailure(self, test, err)
```

```
68          test.longMessage=err[1]
69          self.result = self.FAIL
70
71      def stopTest(self, test):
72          unittest.TestResult.stopTest(self, test)
73          # display: print test result
74          self.runner.writeUpdate(self.result)
75          # DB: PREPARE THE DATA TO BE INSERTED
76          dbdata = {}
77          dbdata['uuid'] = globalVar.g_uuid
78          dbdata['name'] = test.shortDescription()
79          dbdata['result'] = self.result
80          dbdata['msg'] = str(test.longMessage)
81          #DB: INSERT IN THE DATABASE
82          filename='test_results.dat'
83          testResult = open(filename, 'a')
84          testResult.write(str(dbdata))
85          testResult.write("\n")
86          testResult.close()
87          #CONVERT FANCY FAIL AND PASS
88          if self.result==self.PASS: simple_result="TRUE"
89          if self.result == self.FAIL: simple_result = "FALSE"
90          elif self.result == self.ERROR: simple_result = "FALSE"
91          #SEND TO DB THE RESULT OF THE TEST
92          psdb = TestSrv_Database()
93          psdb.open("setup.xml")
94          psdb.add_test_to_batch(globalVar.g_uuid, test.shortDescription(),
    globalVar.testid_ctl, simple_result, test.longMessage)
95
```