```
import struct
import time
import re

import Gseos
import GseosDecoder
import GseosMonitor
import GseosNet
from   GseosBlocks import Blocks
from core.util.PausableSequencer import PausableSequencer as Seq
from core.util import logging as log

DELPHY_MESSAGE     = Blocks["DELPHY_MESSAGE"]
DELPHY_COMPLETION  = Blocks["DELPHY_COMPLETION"]
DELPHY_ACKNOWLEDGE = Blocks["DELPHY_ACKNOWLEDGE"]
DELPHY_IN          = Blocks["DELPHY_IN"]
DELPHY_OUT         = Blocks["DELPHY_OUT"]
DELPHY_STATUS      = Blocks["DELPHY_STATUS"]

PKT_TYPE_IDENTITY = 10
PKT_TYPE_CONTROL  = 8
PKT_TYPE_SCRIPT   = 6
PKT_TYPE_MESSAGE  = 4
PKT_TYPE_ACK      = 0
PKT_TYPE_COMPLETE = 12
NET_CLIENT        = 'DELPHY_CLIENT'
MACHINE_ID        = 14
SUCCESSFUL        = 0
ABORTED           = 1
EXCEPTION         = 2


def client_connect_handler(client_name, connected):
    DELPHY_STATUS.Connected = connected

    if connected == True:
        # On connect, need to send an ID packet or DELPHY will kick us out
        pkt = MakePacket(session, PKT_TYPE_IDENTITY,MACHINE_ID)
        SendPacket(pkt)
    else:
        # On disconnect we may lose control authority
        DELPHY_STATUS.CtrlRequest = 0

    DELPHY_STATUS.SendBlock(True)

GseosNet.ClientAddConnectHandler(NET_CLIENT, client_connect_handler)

# Class definitions
class Session_t:
    time     = time.time()
    packetID = 1
    buffer   = []
    fd       = 0      # file descriptor.  Used for streaming packets to a file

class PCSPacket:
    type        = 0
    id          = 0
    sessionTime = 0.0
    packetTime  = 0.0
    length      = 0
    data        = None
    sync        = 0

class TimeoutError(Exception):
   def __str__(self):
      return 'Time Out Error'
```

```
67
68  class MsgObj_t:
69
70      # Private Attributes
71      _flag    = False
72      _oseq    = None
73      _monitor = None
74      _regexp  = None
75      _type    = None
76
77      # Public Atributes
78      msg  = ''
79
80      def __init__(self, type, oseq = None):
81          self._oseq = oseq
82          self._type = type
83
84      def Arm(self):
85          if self._monitor == None:
86              raise UserWarning('Object must be registered to a block before Arming')
87
88          self._flag = 0              # set the trap
89          self._monitor.bEnable = 1 # enable monitor after clearing ready flag
90
91
92      def Wait (self, timeout = 0):
93          if timeout > 0:
94              t = time.time()+timeout
95          else:
96              t = time.time() + 1000000
97
98          while self._flag == False and time.time() < t:
99              if (self._oseq):
100                 self._oseq.Sleep(0.250)
101             else:
102                 time.sleep(0.250)
103
104         if self._flag == False:  # will be false if timed out
105             raise TimeoutError
106
107     def _Ready(self,blk):
108         if self._flag == True:
109             return   # we already have a squirrel in the trap
110                      # -- the monitor should be disabled, so this shouldn't
111                      #    happen.
112
113         msg = blk.Message.ReadBytes(0,blk.Len)
114
115         try:
116             if (self._regexp == None) or (self._regexp != None and re.search(self._regexp,
            msg)):
117                 # disable monitor to help prevent thread access issues w/ self._flag
118                 self._monitor.bEnable = 0
119                 self.msg = msg
120                 if self._type == PKT_TYPE_MESSAGE:
121                     self.level = blk.Level
122
123                 elif self._type == PKT_TYPE_ACK:
124                     self.code = blk.Code
125                     self.id = blk.ID
126
127                 elif self._type == PKT_TYPE_COMPLETE:
128                     self.code = blk.Code
129
130                 self._flag = True
131         except: # bad filter; take no action
```

```
132             pass
133
134     def Register(self,identifier, blk):
135         self._monitor = GseosMonitor.TMonitor(identifier,self._Ready)
136         self._monitor.bEnable = 0 # make sure monitor is not enabled before registering it
137         blk.Monitors.append(self._monitor)
138
139     def Delete(self):
140         if self._monitor != None:
141             self._monitor.Delete()
142
143
144 def MakePacket(session, type,data):
145     pkt             = PCSPacket()
146     pkt.sync        = 0xDEADBEEF
147     pkt.id          = session.packetID
148     pkt.sessionTime = session.time
149     pkt.packetTime  = time.time()
150     pkt.type        = type
151
152
153     if pkt.type == PKT_TYPE_IDENTITY:
154         pkt.data = struct.pack('!L', data)      # data = machine ID
155     elif pkt.type==PKT_TYPE_CONTROL or pkt.type==PKT_TYPE_SCRIPT:
156         pkt.data = struct.pack('!L',len(data))  #data = text
157         pkt.data += data
158
159     buffer = struct.pack('!LLLdd',pkt.sync,pkt.type,pkt.id,pkt.sessionTime,pkt.packetTime
        )
160
161     if pkt.data:
162         pkt.length = len(pkt.data)
163         buffer += struct.pack('!L',pkt.length) + pkt.data
164     else:
165         pkt.length = 0
166         buffer += struct.pack('!L',pkt.length)
167
168     session.packetID += 1
169
170     return buffer
171
172 def int32(raw):
173     s = bytes(raw)
174     return(struct.unpack('!L',s))[0] #convert string to int32
175
176 def float64(raw):
177     s = bytes(raw)
178     return (struct.unpack('!d',s))[0]
179
180 def DELPHYDecoder(blk):
181     global session                                                  # session
        information
182
183     session.buffer.extend(list(blk.Block[:blk.Len]))
184
185     # extract packets from byte stream
186     packet = ParsePCSStream(session.buffer)
187     while (packet):
188         if (packet.type == PKT_TYPE_MESSAGE):
189             level                   = int32(packet.data[:4])
190             msglen                  = int32(packet.data[4:8])
191             msg                     = packet.data[8:]
192             msgstr = "".join(chr(x) for x in msg if x!=0)
193             log.EventLog('DLPH', f'DELPHY:MSG :LEVEL  {level}: {msgstr}', wrap=False)
194
195             DELPHY_MESSAGE.Level      = level
```

```
            DELPHY_MESSAGE.Len        = msglen
            DELPHY_MESSAGE.Message[:] = msg
            DELPHY_MESSAGE.SendBlock()

        elif packet.type == PKT_TYPE_ACK:
            id     = int32(packet.data[:4])
            code   = int32(packet.data[4:8])
            msglen = int32(packet.data[8:12])
            msg    = packet.data[12:]
            msgstr = "".join(chr(x) for x in msg if x!=0)
            log.EventLog('DLPH', f'DELPHY:ACK :STATUS {code}: {msgstr}', wrap=False)

            DELPHY_ACKNOWLEDGE.ID         = id
            DELPHY_ACKNOWLEDGE.Code       = code
            DELPHY_ACKNOWLEDGE.Len        = msglen
            DELPHY_ACKNOWLEDGE.Message[:] = msg
            DELPHY_ACKNOWLEDGE.SendBlock()

        elif packet.type == PKT_TYPE_COMPLETE:
            code   = int32(packet.data[:4])
            msglen = int32(packet.data[4:8])
            msg    = packet.data[8:]
            msgstr = "".join(chr(x) for x in msg if x!=0)
            log.EventLog('DLPH', f'DELPHY:CPLT:STATUS {code}: {msgstr}', wrap=False)

            DELPHY_COMPLETION.Code        = code
            DELPHY_COMPLETION.Len         = msglen
            DELPHY_COMPLETION.Message[:]  = msg
            DELPHY_COMPLETION.SendBlock()

        packet = ParsePCSStream(session.buffer) # process all the packets


def ParsePCSStream(buffer):
    packet = PCSPacket()
    size = len(buffer)

    if (size) < 32:
        return None               # don't have enough data to parse yet

    # search for sync pattern
    sync = int32(buffer[:4])
    while (size >3 and sync != 0xDEADBEEF):
        del buffer[0]             # discard garbage bytes
        sync = int32(buffer[:4])  # look at the next 4 bytes
        size -= 1                 # keep track of the size

    if (size) < 32:
        return None               # not enough data after searching for sync pattern

    # if we get here, then we found a header

    packet.length = int32(buffer[28:32])  # get payload size
    if ((size - 32) < packet.length):     # check if we have a whole packet
        return None                       # don't have a full packet

    # retrieve the header
    packet.sync        = int32(buffer[0:4])
    packet.type        = int32(buffer[4:8])
    packet.id          = int32(buffer[8:12])
    packet.sessionTime = float64(buffer[12:20])
    packet.packetTime  = float64(buffer[20:28])

    # retrieve the data
    packet.data = buffer[32:32+packet.length]

```

```
262         # remove the packet from the buffer
263         del buffer[:32+packet.length] # consume the packet
264 
265         return packet
266 
267     def SendPacket(pkt):
268         DELPHY_OUT.Len = len(pkt)
269         DELPHY_OUT.Block.WriteBytes(0, pkt)
270         DELPHY_OUT.SendBlock()
271 
272     def WaitPkt(oseq, pkt, timeout=0):
273     # DESCRIPTION:  Wait for specific packet type to arrive.  This is work around
274     #   for a short coming of the TSequencer.wait() function.  The ACK packet, and
275     #  in some cases the COMPLETE packet, was arriving before TSequencer.Wait()
276     #  could catch it.  As a result TSequencer.Wait() would miss it and hang or
277     #  timeout.  To get around this, monitors were permanently attached to both
278     #   packet types and a flag is set to True when it arrived.  To make sure the
279     #  calling function retrieves the correct packet, it is copied to a pkt
280     #  object by the monitor and returned by this function.
281         if timeout == 0:
282             while (pkt.flag == False):
283                 oseq.Sleep(0.100)
284         else:
285             t = time.time() + timeout
286             while (pkt.flag == False and time.time() < t):
287                 oseq.Sleep(0.100)
288 
289         if pkt.flag == False:
290             return None
291         else:
292             return pkt
293 
294     def client_connected():
295        return GseosNet.ClientStatus(NET_CLIENT) == GseosNet.CONNECTED
296 
297     def request_control_seq(oseq, session, timeout):
298        "Request external control of DELPHY system"
299 
300        if not client_connected():
301          Gseos.MessageBox("Request Control Failed: Not connected to DELPHY",
302                           "Request Failed",
303                           wIcon = Gseos.MB_ICONSTOP)
304          DELPHY_STATUS.CtrlRequest = False
305          DELPHY_STATUS.SendBlock(1)
306          return
307 
308        DELPHY_STATUS.CtrlRequest = False
309 
310        ack = MsgObj_t(PKT_TYPE_ACK,oseq)
311        ack.Register('ControlRequest', DELPHY_ACKNOWLEDGE)
312        ack.Arm()
313        ack.id = 0
314 
315        pkt = MakePacket(session, PKT_TYPE_CONTROL, b"CoDICE GSE")
316        SendPacket(pkt)
317 
318        t = time.time() + timeout
319 
320           # wait for an ack with a specific packet ID
321        try:
322           while (ack.id != (session.packetID-1) and time.time() < t):
323               ack.Wait(timeout)   # wait up to the full time for an ack packet
324 
325           if ack.id != session.packetID-1:
326              log.EventLog('ERROR','GSE: Time out waiting for acknowledge')
327              Gseos.MessageBox('Time out waiting for acknowledge', 'REQUEST CONTROL')
```

```
328
329         if ack.code != SUCCESSFUL:
330             log.EventLog('ERROR','GSE: Control Request not successful: %s' % ack.msg, wrap=
                False)
331             Gseos.MessageBox(ack.msg,'REQUEST CONTROL',bModeless = True)
332
333         else:
334             DELPHY_STATUS.CtrlRequest = True
335     except TimeoutError as e:
336         log.EventLog('ERROR','GSE: Time out waiting for acknowledge', wrap=False)
337         Gseos.MessageBox('Time out waiting for acknowledge', 'REQUEST CONTROL')
338
339     DELPHY_STATUS.SendBlock(1)
340
341 def request_control(timeout):
342     global session
343     return Seq("DELPHY Control Request", request_control_seq, (session, timeout))
344
345
346 def send_command_seq(oseq, command, timeout=0):
347 #   test_info.start_test()
348     global session
349
350     command = bytes(command, 'utf-8')
351     abort = False
352     pkt = MakePacket(session, PKT_TYPE_SCRIPT, command)
353
354     ack  = MsgObj_t(PKT_TYPE_ACK,oseq)
355     cplt = MsgObj_t(PKT_TYPE_COMPLETE, oseq)
356
357     ack.Register('WaitForAck',DELPHY_ACKNOWLEDGE)
358     cplt.Register('WaitForComplete',DELPHY_COMPLETION)
359
360     ack.Arm()    # ready to receive ack packet
361     cplt.Arm()   # ready to receive completion packet
362     t = time.time()    # time stamp
363     SendPacket(pkt)
364
365     log.EventLog('DLPH', 'DELPHY Command: %s' % command, wrap=False)
366
367     try:
368         ack.Wait(5)                     #wait here for acknowledge packet
369         if (ack.code != SUCCESSFUL):
370             log.EventLog("ERROR", "ACK: STATUS %i: %s" % (ack.code, ack.msg), wrap=False)
371             abort = True
372     except TimeoutError:
373         log.EventLog("ERROR", "GSE: Timeout waiting for ACK" , wrap=False)
374         abort = True
375
376     if abort != True:
377         try:
378             cplt.Wait(timeout)                  # wait here for complete packet
379             if cplt.code != SUCCESSFUL:
380                 log.EventLog("ERROR", "CPLT: STATUS %i: %s" % (cplt.code, cplt.msg), wrap=
                    False)
381             else:
382                 log.EventLog("DLPH", "DELPHY Command: SUCCESS: %0.3f seconds" % (time.time()-
                    t), wrap=False)
383
384         except TimeoutError:
385             log.EventLog("ERROR", "GSE: Timeout waiting for COMPLETE", wrap=False)
386
387     ack.Delete()
388     cplt.Delete()
389
390     return not abort
```

```
391   #   test_info.end_test()
392
393   def send_command(command, timeout=0):
394       return Seq('DELPHY Send Command', send_command_seq, (command, timeout))
395
396   def  _connect(oseq):
397      global session
398      GseosNet.ClientConnect(NET_CLIENT)
399      while GseosNet.ClientStatus(NET_CLIENT) == GseosNet.CONNECTING:
400        oseq.Sleep(0.010)
401        pass
402
403      if not client_connected():
404         text = 'Failed to connect to %s' % (NET_CLIENT,)
405         Gseos.MessageBox(text, bModeless = True)
406         log.EventLog('ERROR','GSE: %s' % (text,), wrap=False)
407         DELPHY_STATUS.Connected = False
408      else:
409         # we don't get an ack or completion packet on the ID.  We are going to just
410         # assume it works and not check for a valid response.  If we really wanted
411         # to we could parse the message packets for an external connection msg.
412         pkt = MakePacket(session, PKT_TYPE_IDENTITY,MACHINE_ID)
413         SendPacket(pkt)
414         log.EventLog('REMRK', 'GSE: Connected to %s' % NET_CLIENT, wrap=False)
415         DELPHY_STATUS.Connected = True
416
417      DELPHY_STATUS.SendBlock(1)
418
419   def connect():
420       return Seq("DELPHY Connect", _connect)
421
422
423   session = Session_t()
424   decoder = GseosDecoder.TDecoder('DELPHY_PKT', DELPHYDecoder, [DELPHY_MESSAGE,
      DELPHY_ACKNOWLEDGE, DELPHY_COMPLETION])
425   DELPHY_IN.Decoders.append(decoder)
426
```