SCTE35 Encoder in Python

SCTE35Encoder.py

```
# -*- coding: utf-8 -*-
import bitstring
import sys, traceback
import pprint
import binascii
import pycrc
from crccheck.crc import Crc32Mpeg2, CrcXmodem
from crccheck.checksum import Checksum32
import crcmod.predefined
import base64
import os

class SCTE35Encoder():

    def __init__(self, _pid, _outfile,triggertype,_pts,_duration,_upid,_ptsadjustment = 0): # constructor
        self.pid =  _pid
        self.outputfilename =  _outfile
        self.pts = _pts
        self.duration = _duration
        print ("UPID: " + _upid)
        _upid = (str(_upid).zfill(32))                                      # pad upid with 0 so we have 38
bytes. the value 32 comes from we prepend LBTY+version+command automatically which is 6 bytes
        if str(triggertype) == "0":
            self.upid = [76,66,84,89,0,0] + [int(d) for d in (_upid)]      # automatically prepends LBTY +
Version + command (trigger or pre-trigger)
        elif str(triggertype) == "1":
            self.upid = [76,66,84,89,0,16] + [int(d) for d in (_upid)]     # automatically prepends LBTY +
Version + command (trigger or pre-trigger)
        else:
            raise Exception('triggertype not recognized: [' + str(triggertype) + ']')

        print (_upid)
        self.ptsadjustment = _ptsadjustment
        self.start()

    def __add_time_signal(self,bitarray,value):
        bitarray.append(bitstring.BitArray('bool=True'))               #  time_specified_flag
        #if above is true
        bitarray.append(bitstring.BitArray('uint:6=63'))               #  reserved
        bitarray.append(bitstring.BitArray('uint:33=' + str(value)))    #  pts_time

    def __add_splice_descriptor(self,duration,upid):                    #  a splice_descriptor encapsulates
segmentation descriptor

        subdesc = self.__get_segmentation_body(duration,upid);
        size_subdesc = len(subdesc.bytes) + 4;                          # +4 because we need to add the bytes
of CUEI string to the "size"
        desc = bitstring.BitArray()
        desc.append(bitstring.BitArray('uint:8=2'))                     #  splice_descriptor_tag
        desc.append(bitstring.BitArray('uint:8=' + str(size_subdesc)))  #  splice_descriptor_length
        desc.append(bitstring.BitArray('uint:32='+str(1129661769)))     #  identifier (CUEI = 1129661769)
        desc.append(subdesc)
        return desc
        # TODO: add Content first, then calculate length, then build message

    def __get_segmentation_body(self,duration,upid):

        seg_duration_flag = "True"
        body = bitstring.BitArray()
        body.append(bitstring.BitArray('uint:32=3'))                    #   DYNAMIC segmentation_event_id
        body.append(bitstring.BitArray('bool=False'))                   #   event_cancel_indicator
        body.append(bitstring.BitArray('uint:7=1'))                     #   reserved 7 bit
        # if cancel indicator == 0
        body.append(bitstring.BitArray('bool=True'))                    #   program_segmentation_flag
        body.append(bitstring.BitArray('bool=' + seg_duration_flag))    #   segmentation_duration_flag
```

```
        body.append(bitstring.BitArray('bool=True'))                    #   delivery_not_restricted_flag. Make
sure to implement sub-fields in case this value is False

        # TODO: if delivery_not_restricted_flag == 0, add  web_delivery_allowed_flag no_regional_blackout_flag
archive_allowed_flag  device_restrictions

        body.append(bitstring.BitArray('uint:5=1'))                     #   reserved because
delivery_not_restricted is true

        # TODO: if program_segmentation_flag == 0, add  component_count and for each component:
component_tag,  reserved,  pts_offset

        if seg_duration_flag=="True":
            body.append(bitstring.BitArray('uint:40=' + str(duration))) #   DYNAMIC segmentation duration

        body.append(bitstring.BitArray('uint:8=12'))                    #   segmentation_upid_type - MPU = 0xC
        body.append(bitstring.BitArray('uint:8=38'))                    #   segmentation_upid_length - The
value of segmentation_upid_length should be in accordance with the value of segmentation_upid_type.

        for char in upid:
            body.append(bitstring.BitArray('uint:8=' + str(char)))      #   appends the LBTY part (upid)

        body.append(bitstring.BitArray('uint:8=54'))                    #   segmentation_type_id 0x36
        body.append(bitstring.BitArray('uint:8=0'))                     #   segment_num # only if id is 36
        body.append(bitstring.BitArray('uint:8=0'))                     #   segments_expected # only if id is 36

        return body

    def dump(self,obj):
        for attr in dir(obj):
            print("obj.%s = %r" % (attr, getattr(obj, attr)))
    # insert package start code and PID

    def __CRC32_from_bitstring(self,bitstring):
        crc32_func = crcmod.predefined.mkCrcFun('crc-32-mpeg')
        buf = crc32_func((bitstring))
        return "%08X" % buf

    def start(self):
        #BASIC PACKET INFO
        o = bitstring.BitArray('uint:8=71')                     # Sync-Byte = 47 decimal or 0x71 hex
        o.append(bitstring.BitArray('bool=False'))              # Transport error indicator
        o.append(bitstring.BitArray('bool=True'))               # Payload start indicator, for SCTE Messages
always true
        o.append(bitstring.BitArray('bool=False'))              # Transport Priority, for SCTE Messages always
false
        o.append(bitstring.BitArray('uint:13=' + str((int(self.pid,0)))))
        o.append(bitstring.BitArray('0b00'))                    # 2bit Transport Scrambling Control
        o.append(bitstring.BitArray('0b01'))                    # 2bit Adaptation Field Control - for SCTE
Messages only 01 supported (no adaption field, just payload)

        o.append(bitstring.BitArray('uint:4=0'))                # Packet Counter, DYNAMIC
        o.append(bitstring.BitArray('uint:8=0'))                # Adaption Field always 0? Included bits: 1
Bit: discontinuity_indicator,1 Bit: random_access_indicator,1 Bit: elementary_stream_priority_indicator,1 Bit:
PCR_flag,1 Bit: OPCR_flag,1 Bit: splicing_point_flag,1 Bit: transport_private_data_flag,1 Bit:
adaptation_field_extension_flag

        o.append(bitstring.BitArray('hex:8=0xFC'))              # table_id This is an 8-bit field. Its value
shall be 0xFC.
        o.append(bitstring.BitArray('bool=False'))              # section_syntax_indicator The
section_syntax_indicator is a 1-bit field that should always be set to '0' indicating that MPEG short sections
are to be used.
        o.append(bitstring.BitArray('bool=False'))              # private  This is a 1-bit flag that shall be
set to 0.
        o.append(bitstring.BitArray('0b11'))                    # need to add 2 bits with value 1, reason yet
unknown

        #START OF SPLICE INFO SECTION as per SCTE2016

        # before adding the rest, we need to insert the size of the rest, use the section object to collect
everything
```

```
        section = bitstring.BitArray();
        section.append(bitstring.BitArray('uint:8=0'))       # protocol_version is an 8 bit unsigned integer
field whose function is to allow, in the future, this table type to carry parameters that may be structured
differently than those defined in the current protocol. At present, the only valid value for protocol_version
is zero. Non-zero values of protocol_version may be used by a future version of this standard to indicate
structurally different tables
        section.append(bitstring.BitArray('bool=False'))     # encrypted_packet - When this bit is set to '1'.
it indicates that portions of the splice_info_section, starting with splice_command_type and ending with and
including E_CRC_32, are encrypted. When this bit is set to '0', no part of this message is encrypted. The
potentially encrypted portions of the splice_info_table are indicated by an E in the Encrypted column of Table
5.
        section.append(bitstring.BitArray('uint:6=0'))       # encryption_algorithm This 6 bit unsigned
integer specifies which encryption algorithm was used to encrypt the #current message. When the
encrypted_packet bit is zero, this field is present but undefined. Refer to section 11, and specifically Table
26  Encryption algorithmfor details on the use of this field. @encryptionAlgorithm [Conditional Mandatory, xsd:
unsignedByte] If the EncryptedPacket Element is present this value shall be provided.  It is intended that the
first device that restamps pcr/pts/dts and that passes the cueing message will insert a value into the
pts_adjustment field, which is the delta time between this devices input time domain and its output time
domain. All subsequent devices, which also restamp pcr/pts/dts, may further alter the pts_adjustment field by
adding their delta time to the field's existing delta time and placing the result back in the pts_adjustment
field. Upon each alteration of the pts_adjustment field, the altering device shall recalculate and update the
CRC_32 field. The pts_adjustment shall, at all times, be the proper value to use for conversion of the pts_time
field to the curent time-base. The conversion is done by adding the two fields. In the presence of a wrap or
overflow condition the carry shall be ignored. ptsAdjustment [Optional, PTSType] See section 13.2.
        section.append(bitstring.BitArray('uint:33=' + str(self.ptsadjustment)))       # pts_adjustment
POSSIBLE DYNAMIC A 33 bit unsigned integer that appears in the clear and that shall be used by a splicing
device as an offset to be added to the (sometimes) encrypted pts_time field(s) throughout this message to
obtain the intended splice time(s). When this field has a zero value, then the pts_time field(s) shall be used
without an offset. Normally, the creator of a cueing message will place a zero value into this field. This
adjustment value is the means by which an upstream device, which restamps pcr/pts/dts, may convey to the
splicing device the means by which to convert the pts_time field of the message to a newly imposed time domain
        section.append(bitstring.BitArray('uint:8=0'))       # cw_index An 8 bit unsigned integer that conveys
which control word (key) is to be used to decrypt the message. The splicing device may store up to 256 keys
previously provided for this purpose. When the encrypted_packet bit is zero, this field is present but
undefined. @cwIndex [Conditional Mandatory, xsd:unsignedByte] If the EncryptedPacket Element is present this
value shall be provided
        section.append(bitstring.BitArray('hex:12=0xFFF'))    # tier A 12-bit value used by the SCTE 35 message
provider to assign messages to authorization tiers. This field may take any value between 0x000 and 0xFFF. The
value of 0xFFF provides backwards compatibility and shall be ignored by downstream equipment. When using tier,
the message provider should keep the entire message in a single transport stream packe
        section.append(bitstring.BitArray('uint:12=05'))     # splice_command_length (basically size of
time_signal = 5bytes)  a 12 bit length of the splice command. The length shall represent the number of bytes
following the splice_command_type up to but not including the descriptor_loop_length. Devices that are
compliant with this version of the standard shall populate this field with the actual length. The value of
0xFFF provides backwards compatibility and shall be ignored by downstream equipment.
        section.append(bitstring.BitArray('uint:8=06'))      # splice_command_type  An 8-bit unsigned integer
which shall be assigned one of the values shown in column labeled splice_command_type value in Table 6.

        # TIME SIGNAL
        self.__add_time_signal(section,self.pts)                 # time_signal component DYNAMIC

        # DESCRIPTOR LOOP

        descriptor = (self.__add_splice_descriptor(self.duration,self.upid))                  # adds CUEI
descriptor containing LBTY descriptor
        desc_size = len(descriptor.bytes)
        section.append(bitstring.BitArray('uint:16='+str(desc_size)))        # descriptor loop length
        section.append(descriptor)
        section_size = len(section.bytes)
        o.append(bitstring.BitArray('uint:12=' + str(section_size)))      # section length This is a 12-bit
field specifying the number of remaining bytes in the splice_info_section immediately following the
section_length field up to the end of the splice_info_section. The value in this field shall not exceed 4093
        o.append(section)

        #write output file
        f = open(self.outputfilename, 'wb')
        o.tofile(f)
        f.close()
        print ("Wrote output file: " + self.outputfilename)

        #calculate and write CRC32
```

```
        del o[0:8] # delete first 8 bits, start code 47, they are not included in checksum
        crc = (self.__CRC32_from_bitstring(o.tobytes()));
        f = open(self.outputfilename, 'ab')
        f.write(binascii.a2b_hex(crc))
        f.close()
        print ("Wrote checksum to file: " + crc )

        # output base64 for testing
        f = open(self.outputfilename, 'rb')
        f.seek(5)
        encoded_bytes = base64.b64encode(f.read())
        f.close()
        print ("Base64: " + str(encoded_bytes))

        #check filesize and append 0xFF until we have 188bytes
        statinfo = os.stat(self.outputfilename)
        fillsize = 188-statinfo.st_size
        print ("fillsize "+str(fillsize))
        if fillsize < 0:
            raise Exception("Error, outputfile " + self.outputfilename + " got bigger than 188 bytes!")
        f = open(self.outputfilename, 'ab')
        for x in range(0,fillsize):
            print (x)
            f.write(bytes([255]))
        f.close()



if __name__ == "__main__":
    SCTE35Encoder("c:\\temp\\file.bin",0,0,0,"abc-123-abc-123-az")
```