import struct

"""

The internal FLV representations of numbers.

"""

\_\_all\_\_ = ['get\_ui32', 'make\_ui32', 'get\_si32\_extended', 'make\_si32\_extended',

'get\_ui24', 'make\_ui24', 'get\_ui16', 'make\_ui16',

'get\_si16', 'make\_si16', 'get\_ui8', 'make\_ui8',

'get\_double', 'make\_double', 'EndOfFile']

class EndOfFile(Exception):

pass

# UI32

def get\_ui32(f):

try:

ret = struct.unpack(">I", f.read(4))[0]

except struct.error:

raise EndOfFile

return ret

def make\_ui32(num):

return struct.pack(">I", num)

# SI32 extended

def get\_si32\_extended(f):

# The last 8 bits are the high 8 bits of the whole number

# That's how Adobe likes it. Go figure...

low\_high = f.read(4)

if len(low\_high) < 4:

raise EndOfFile

combined = low\_high[3] + low\_high[:3]

return struct.unpack(">i", combined)[0]

def make\_si32\_extended(num):

ret = struct.pack(">i", num)

return ret[1:] + ret[0]

# UI24

def get\_ui24(f):

try:

high, low = struct.unpack(">BH", f.read(3))

except struct.error:

raise EndOfFile

ret = (high << 16) + low

return ret

def make\_ui24(num):

ret = struct.pack(">I", num)

return ret[1:]

# UI16

def get\_ui16(f):

try:

ret = struct.unpack(">H", f.read(2))[0]

except struct.error:

raise EndOfFile

return ret

def make\_ui16(num):

return struct.pack(">H", num)

# SI16

def get\_si16(f):

try:

ret = struct.unpack(">h", f.read(2))[0]

except struct.error:

raise EndOfFile

return ret

def make\_si16(num):

return struct.pack(">h", num)

# UI8

def get\_ui8(f):

try:

ret = struct.unpack("B", f.read(1))[0]

except struct.error:

raise EndOfFile

return ret

def make\_ui8(num):

return struct.pack("B", num)

# DOUBLE

def get\_double(f):

data = f.read(8)

try:

ret = struct.unpack(">d", data)[0]

except struct.error:

raise EndOfFile

return ret

def make\_double(num):

return struct.pack(">d", num)