import os

import calendar

import datetime

import logging

from primitives import \*

from constants import \*

from helpers import OrderedAttrDict, utc

"""

The AS types and their FLV representations.

"""

log = logging.getLogger('flvlib.astypes')

class MalformedFLV(Exception):

pass

# Number

def get\_number(f, max\_offset=None):

return get\_double(f)

def make\_number(num):

return make\_double(num)

# Boolean

def get\_boolean(f, max\_offset=None):

value = get\_ui8(f)

return bool(value)

def make\_boolean(value):

return make\_ui8((value and 1) or 0)

# String

def get\_string(f, max\_offset=None):

# First 16 bits are the string's length

length = get\_ui16(f)

# Then comes the string itself

ret = f.read(length)

return ret

def make\_string(string):

if isinstance(string, unicode):

# We need a blob, not unicode. Arbitrarily choose UTF-8

string = string.encode('UTF-8')

length = make\_ui16(len(string))

return length + string

# Longstring

def get\_longstring(f, max\_offset=None):

# First 32 bits are the string's length

length = get\_ui32(f)

# Then comes the string itself

ret = f.read(length)

return ret

def make\_longstring(string):

if isinstance(string, unicode):

# We need a blob, not unicode. Arbitrarily choose UTF-8

string = string.encode('UTF-8')

length = make\_ui32(len(string))

return length + string

# ECMA Array

class ECMAArray(OrderedAttrDict):

pass

def get\_ecma\_array(f, max\_offset=None):

length = get\_ui32(f)

log.debug("The ECMA array has approximately %d elements", length)

array = ECMAArray()

while True:

if max\_offset and (f.tell() == max\_offset):

log.debug("Prematurely terminating reading an ECMA array")

break

marker = get\_ui24(f)

if marker == 9:

log.debug("Marker!")

break

else:

f.seek(-3, os.SEEK\_CUR)

name, value = get\_script\_data\_variable(f, max\_offset=max\_offset)

array[name] = value

return array

def make\_ecma\_array(d):

length = make\_ui32(len(d))

rest = ''.join([make\_script\_data\_variable(name, value)

for name, value in d.iteritems()])

marker = make\_ui24(9)

return length + rest + marker

# Strict Array

def get\_strict\_array(f, max\_offset=None):

length = get\_ui32(f)

log.debug("The length is %d", length)

elements = [get\_script\_data\_value(f, max\_offset=max\_offset)

for \_ in xrange(length)]

return elements

def make\_strict\_array(l):

ret = make\_ui32(len(l))

rest = ''.join([make\_script\_data\_value(value) for value in l])

return ret + rest

# Date

def get\_date(f, max\_offset=None):

timestamp = get\_number(f) / 1000.0

# From the following document:

# http://opensource.adobe.com/wiki/download/

# attachments/1114283/amf0\_spec\_121207.pdf

#

# Section 2.13 Date Type

#

# (...) While the design of this type reserves room for time zone offset

# information, it should not be filled in, nor used (...)

\_ignored = get\_si16(f)

return datetime.datetime.fromtimestamp(timestamp, utc)

def make\_date(date):

if date.tzinfo:

utc\_date = date.astimezone(utc)

else:

# assume it's UTC

utc\_date = date.replace(tzinfo=utc)

ret = make\_number(calendar.timegm(utc\_date.timetuple()) \* 1000)

offset = 0

return ret + make\_si16(offset)

# Null

def get\_null(f, max\_offset=None):

return None

def make\_null(none):

return ''

# Object

class FLVObject(OrderedAttrDict):

pass

def get\_object(f, max\_offset=None):

ret = FLVObject()

while True:

if max\_offset and (f.tell() == max\_offset):

log.debug("Prematurely terminating reading an object")

break

marker = get\_ui24(f)

if marker == 9:

log.debug("Marker!")

break

else:

f.seek(-3, os.SEEK\_CUR)

name, value = get\_script\_data\_variable(f)

setattr(ret, name, value)

return ret

def make\_object(obj):

# If the object is iterable, serialize keys/values. If not, fall

# back on iterating over \_\_dict\_\_.

# This makes sure that make\_object(get\_object(StringIO(blob))) == blob

try:

iterator = obj.iteritems()

except AttributeError:

iterator = obj.\_\_dict\_\_.iteritems()

ret = ''.join([make\_script\_data\_variable(name, value)

for name, value in iterator])

marker = make\_ui24(9)

return ret + marker

# MovieClip

class MovieClip(object):

def \_\_init\_\_(self, path):

self.path = path

def \_\_eq\_\_(self, other):

return isinstance(other, MovieClip) and self.path == other.path

def \_\_repr\_\_(self):

return "<MovieClip at %s>" % self.path

def get\_movieclip(f, max\_offset=None):

ret = get\_string(f)

return MovieClip(ret)

def make\_movieclip(clip):

return make\_string(clip.path)

# Undefined

class Undefined(object):

def \_\_eq\_\_(self, other):

return isinstance(other, Undefined)

def \_\_repr\_\_(self):

return '<Undefined>'

def get\_undefined(f, max\_offset=None):

return Undefined()

def make\_undefined(undefined):

return ''

# Reference

class Reference(object):

def \_\_init\_\_(self, ref):

self.ref = ref

def \_\_eq\_\_(self, other):

return isinstance(other, Reference) and self.ref == other.ref

def \_\_repr\_\_(self):

return "<Reference to %d>" % self.ref

def get\_reference(f, max\_offset=None):

ret = get\_ui16(f)

return Reference(ret)

def make\_reference(reference):

return make\_ui16(reference.ref)

as\_type\_to\_getter\_and\_maker = {

VALUE\_TYPE\_NUMBER: (get\_number, make\_number),

VALUE\_TYPE\_BOOLEAN: (get\_boolean, make\_boolean),

VALUE\_TYPE\_STRING: (get\_string, make\_string),

VALUE\_TYPE\_OBJECT: (get\_object, make\_object),

VALUE\_TYPE\_MOVIECLIP: (get\_movieclip, make\_movieclip),

VALUE\_TYPE\_NULL: (get\_null, make\_null),

VALUE\_TYPE\_UNDEFINED: (get\_undefined, make\_undefined),

VALUE\_TYPE\_REFERENCE: (get\_reference, make\_reference),

VALUE\_TYPE\_ECMA\_ARRAY: (get\_ecma\_array, make\_ecma\_array),

VALUE\_TYPE\_STRICT\_ARRAY: (get\_strict\_array, make\_strict\_array),

VALUE\_TYPE\_DATE: (get\_date, make\_date),

VALUE\_TYPE\_LONGSTRING: (get\_longstring, make\_longstring)

}

type\_to\_as\_type = {

bool: VALUE\_TYPE\_BOOLEAN,

int: VALUE\_TYPE\_NUMBER,

long: VALUE\_TYPE\_NUMBER,

float: VALUE\_TYPE\_NUMBER,

# WARNING: not supporting Longstrings here.

# With a max length of 65535 chars, noone will notice.

str: VALUE\_TYPE\_STRING,

unicode: VALUE\_TYPE\_STRING,

list: VALUE\_TYPE\_STRICT\_ARRAY,

dict: VALUE\_TYPE\_ECMA\_ARRAY,

ECMAArray: VALUE\_TYPE\_ECMA\_ARRAY,

datetime.datetime: VALUE\_TYPE\_DATE,

Undefined: VALUE\_TYPE\_UNDEFINED,

MovieClip: VALUE\_TYPE\_MOVIECLIP,

Reference: VALUE\_TYPE\_REFERENCE,

type(None): VALUE\_TYPE\_NULL

}

# SCRIPTDATAVARIABLE

def get\_script\_data\_variable(f, max\_offset=None):

name = get\_string(f)

log.debug("The name is %s", name)

value = get\_script\_data\_value(f, max\_offset=max\_offset)

log.debug("The value is %r", value)

return (name, value)

def make\_script\_data\_variable(name, value):

log.debug("The name is %s", name)

log.debug("The value is %r", value)

ret = make\_string(name) + make\_script\_data\_value(value)

return ret

# SCRIPTDATAVALUE

def get\_script\_data\_value(f, max\_offset=None):

value\_type = get\_ui8(f)

log.debug("The value type is %r", value\_type)

try:

get\_value = as\_type\_to\_getter\_and\_maker[value\_type][0]

except KeyError:

raise MalformedFLV("Invalid script data value type: %d", value\_type)

log.debug("The getter function is %r", get\_value)

value = get\_value(f, max\_offset=max\_offset)

return value

def make\_script\_data\_value(value):

value\_type = type\_to\_as\_type.get(value.\_\_class\_\_, VALUE\_TYPE\_OBJECT)

log.debug("The value type is %r", value\_type)

# KeyError can't happen here, because we always fall back on

# VALUE\_TYPE\_OBJECT when determining value\_type

make\_value = as\_type\_to\_getter\_and\_maker[value\_type][1]

log.debug("The maker function is %r", make\_value)

type\_tag = make\_ui8(value\_type)

ret = make\_value(value)

return type\_tag + ret