import os

import time

import datetime

from StringIO import StringIO

from UserDict import DictMixin

class UTC(datetime.tzinfo):

"""

A UTC tzinfo class, based on

http://docs.python.org/library/datetime.html#datetime.tzinfo

"""

ZERO = datetime.timedelta(0)

def utcoffset(self, dt):

return self.ZERO

def tzname(self, dt):

return "UTC"

def dst(self, dt):

return self.ZERO

utc = UTC()

class OrderedAttrDict(DictMixin):

"""

A dictionary that preserves insert order and also has an attribute

interface.

Values can be transparently accessed and set as keys or as attributes.

"""

def \_\_init\_\_(self, dict=None, \*\*kwargs):

self.\_\_dict\_\_["\_order\_priv\_"] = []

self.\_\_dict\_\_["\_data\_priv\_"] = {}

if dict is not None:

self.update(dict)

if len(kwargs):

self.update(kwargs)

# Mapping interface

def \_\_setitem\_\_(self, key, value):

if key not in self:

self.\_order\_priv\_.append(key)

self.\_data\_priv\_[key] = value

def \_\_getitem\_\_(self, key):

return self.\_data\_priv\_[key]

def \_\_delitem\_\_(self, key):

del self.\_data\_priv\_[key]

self.\_order\_priv\_.remove(key)

def keys(self):

return list(self.\_order\_priv\_)

# Attribute interface

def \_\_getattr\_\_(self, name):

try:

return self[name]

except KeyError:

raise AttributeError(name)

def \_\_setattr\_\_(self, name, value):

self[name] = value

def \_\_delattr\_\_(self, name):

try:

del self[name]

except KeyError:

raise AttributeError(name)

# Equality

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

try:

my\_iter = self.iteritems()

his\_iter = other.iteritems()

except AttributeError:

return False

my\_empty = False

his\_empty = False

while True:

try:

my\_key, my\_val = my\_iter.next()

except StopIteration:

my\_empty = True

try:

his\_key, his\_val = his\_iter.next()

except StopIteration:

his\_empty = True

if my\_empty and his\_empty:

return True

if my\_empty or his\_empty:

return False

if (my\_key, my\_val) != (his\_key, his\_val):

return False

# String representation

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

return '<%s %s>' % (self.\_\_class\_\_.\_\_name\_\_, self)

def \_\_str\_\_(self):

return '{' + ', '.join([('%r: %r' % (key, self[key]))

for key in self.\_order\_priv\_]) + '}'

class ASPrettyPrinter(object):

"""Pretty printing of AS objects"""

def pformat(cls, val, indent=0):

cls.io = StringIO()

cls.pprint\_lookup(val, indent)

return cls.io.getvalue()

pformat = classmethod(pformat)

def pprint(cls, val):

print cls.pformat(val)

pprint = classmethod(pprint)

def pprint\_lookup(cls, val, ident):

if isinstance(val, basestring):

return cls.pprint\_string(val)

if isinstance(val, (int, long, float)):

return cls.pprint\_number(val)

if isinstance(val, datetime.datetime):

return cls.pprint\_datetime(val)

if hasattr(val, 'iterkeys'):

# dict interface

return cls.pprint\_dict(val, ident)

if hasattr(val, 'append'):

# list interface

return cls.pprint\_list(val, ident)

# Unknown type ?

cls.io.write("%r" % (val, ))

return False

pprint\_lookup = classmethod(pprint\_lookup)

def pprint\_string(cls, val):

if isinstance(val, unicode):

cls.io.write("u'%s'" % val.encode("UTF8"))

else:

cls.io.write("'%s'" % val)

return False

pprint\_string = classmethod(pprint\_string)

def pprint\_number(cls, val):

cls.io.write(str(val))

return False

pprint\_number = classmethod(pprint\_number)

def pprint\_datetime(cls, val):

cls.io.write(val.replace(microsecond=0).isoformat(' '))

return False

pprint\_datetime = classmethod(pprint\_datetime)

def pprint\_dict(cls, val, indent):

def pprint\_item(k):

last\_pos = cls.io.tell()

cls.io.write(repr(k))

cls.io.write(": ")

new\_indent = indent + cls.io.tell() - last\_pos + 1

return cls.pprint\_lookup(val[k], new\_indent)

cls.io.write('{')

indented = False

keys = list(val.iterkeys())

if keys:

for k in keys[:-1]:

indented |= pprint\_item(k)

cls.io.write(",\n%s " % (" "\*indent))

indented |= pprint\_item(keys[-1])

cls.io.write('}')

return (len(keys) > 1) | indented

pprint\_dict = classmethod(pprint\_dict)

def pprint\_list(cls, val, indent):

last\_pos = cls.io.tell()

cls.io.write('[')

new\_indent = indent + cls.io.tell() - last\_pos

indented = False

values = list(iter(val))

if values:

for v in values[:-1]:

indented |= cls.pprint\_lookup(v, new\_indent)

cls.io.write(",\n%s" % (" "\*new\_indent))

indented |= cls.pprint\_lookup(values[-1], new\_indent)

cls.io.write(']')

return (len(values) > 1) | indented

pprint\_list = classmethod(pprint\_list)

pformat = ASPrettyPrinter.pformat

pprint = ASPrettyPrinter.pprint

def force\_remove(path):

try:

os.remove(path)

except OSError:

pass