PySAC: Python Interface to the Seismic Analysis Code (SAC) File Format.

Jonathan MacCarthy, Los Alamos National Laboratory

Contents

1 PySAC

Python interface to the Seismic Analysis Code (SAC) file format.

File-type support:

- little and big-endian binary format
- alphanumeric format
- evenly-sampled data
- time-series, not spectra

Project page
Repository
Docs PDF

1.1 Goals

1. Expose the file format in a way that is intuitive to SAC users and to Python programmers
2. Maintaining header validity when converting between ObsPy Traces.

1.2 Features

1. **Read and write SAC binary or ASCII**
   - autodetect or specify expected byteorder
   - optional file size checking and/or header consistency checks
   - header-only reading and writing
   - "overwrite OK" checking ('lovrok' header)
2. **Convenient access and manipulation of relative and absolute time headers**
3. **User-friendly header printing/viewing**
4. **Fast access to header values from attributes**
   - With type checking, null handling, and enumerated value checking
5. **Convert to/from ObsPy Traces**
   - Conversion from ObsPy Trace to SAC trace retains detected previous SAC header values.
   - Conversion to ObsPy Trace retains the *complete* SAC header.

1.3 Usage examples

1.3.1 Read/write SAC files

```
# read from a binary file
sac = SACTrace.read(filename)

# read header only
sac = SACTrace.read(filename, headonly=True)

# write header-only, file must exist
sac.write(filename, headonly=True)
```

```
# read from an ASCII file
sac = SACTrace.read(filename, ascii=True)

# write a binary SAC file for a Sun machine
sac.write(filename, byteorder='big')
```

1.3.2 Reference-time and relative time headers

```
sac = SACTrace(nzyear=2000, nzjday=1, nzhour=0, nzmin=0, nzsec=0, nzmsec=0,
               t1=23.5, data=numpy.arange(100))

sac.reftime
sac.b, sac.e, sac.t1
```

```
2000-01-01T00:00:00.000000Z
(0.0, 99.0, 23.5)
```

Move reference time by relative seconds, relative time headers are preserved.

```
sac.reftime -= 2.5
sac.b, sac.e, sac.t1
```

```
(2.5, 101.5, 26.0)
```

Set reference time to new absolute time, relative time headers are preserved.

```
sac.reftime = UTCDateTime(2000, 1, 1, 0, 2, 0, 0)
sac.b, sac.e
```

```
(-120.0, -21.0, -96.5)
```

1.3.3 Quick header viewing

Print non-null header values.

```
sac = SACTrace()
print sac
```

```
Reference Time = 01/01/2000 (001) 00:00:00.000000
    iztype IB: begin time
b           = 0.0
cmpaz       = 0.0
cmpinc      = 0.0
delta       = 1.0
e           = 99.0
iftype      = itime
internal0   = 2.0
iztype      = ib
```

```
kcmpnm     = Z
lcalda     = False
leven      = True
lovrok     = True
lpspol     = True
npts       = 100
nvhdr      = 6
nzhour     = 0
nzjday     = 1
nzmin      = 0
nzmsec     = 0
nzsec      = 0
nzyear     = 2000
```

Print relative time header values.

```
sac.lh('picks')
```

```
Reference Time = 01/01/1970 (001) 00:00:00.000000
    iztype IB: begin time
    a          = None
    b          = 0.0
    e          = 0.0
    f          = None
    o          = None
    t0         = None
    t1         = None
    t2         = None
    t3         = None
    t4         = None
    t5         = None
    t6         = None
    t7         = None
    t8         = None
    t9         = None
```

1.3.4 Header values as attributes

Great for interactive use, with (ipython) tab-completion...

```
sac.<tab>
```

```
sac.a                sac.kevnm            sac.nzsec
sac.az               sac.kf               sac.nzyear
sac.b                sac.khole            sac.o
sac.baz              sac.kinst            sac.odelta
sac.byteorder        sac.knetwk           sac.read
sac.cmpaz            sac.ko               sac.reftime
sac.cmpinc           sac.kstnm            sac.scale
sac.copy             sac.kt0              sac.stdp
sac.data             sac.kt1              sac.stel
sac.delta            sac.kt2              sac.stla
```

```
sac.depmax            sac.kt3              sac.stlo
sac.depmen            sac.kt4              sac.t0
sac.depmin            sac.kt5              sac.t1
sac.dist              sac.kt6              sac.t2
sac.e                 sac.kt7              sac.t3
sac.evdp              sac.kt8              sac.t4
sac.evla              sac.kt9              sac.t5
sac.evlo              sac.kuser0           sac.t6
sac.f                 sac.kuser1           sac.t7
sac.from_obspy_trace  sac.kuser2           sac.t8
sac.gcarc             sac.lcalda           sac.t9
sac.idep              sac.leven            sac.to_obspy_trace
sac.ievreg            sac.lh               sac.unused23
sac.ievtyp            sac.listhdr          sac.user0
sac.iftype            sac.lovrok           sac.user1
sac.iinst             sac.lpspol           sac.user2
sac.imagsrc           sac.mag              sac.user3
sac.imagtyp           sac.nevid            sac.user4
sac.internal0         sac.norid            sac.user5
sac.iqual             sac.npts             sac.user6
sac.istreg            sac.nvhdr            sac.user7
sac.isynth            sac.nwfid            sac.user8
sac.iztype            sac.nzhour           sac.user9
sac.ka                sac.nzjday           sac.validate
sac.kcmpnm            sac.nzmin            sac.write
sac.kdatrd            sac.nzmsec
```

. . . and documentation!

```
sac.iztype?
```

```
Type:        property
String form: <property object at 0x106404940>
Docstring:
I    Reference time equivalence:
* IUNKN (5): Unknown
* IB (9): Begin time
* IDAY (10): Midnight of reference GMT day
* IO (11): Event origin time
* IA (12): First arrival time
* ITn (13-22): User defined time pick n, n=0,9
```

1.3.5 Convert to/from ObsPy Traces

```
from obspy import read
tr = read()[0]
print tr.stats
```

```
         network: BW
         station: RJOB
        location:
```

```
         channel: EHZ
       starttime: 2009-08-24T00:20:03.000000Z
         endtime: 2009-08-24T00:20:32.990000Z
   sampling_rate: 100.0
           delta: 0.01
            npts: 3000
           calib: 1.0
    back_azimuth: 100.0
     inclination: 30.0
```

```
sac = SACTrace.from_obspy_trace(tr)
print sac
```

```
Reference Time = 08/24/2009 (236) 00:20:03.000000
    iztype IB: begin time
b          = 0.0
cmpaz      = 0.0
cmpinc     = 0.0
delta      = 0.00999999977648
depmax     = 1293.77099609
depmen     = -4.49556303024
depmin     = -1515.81311035
e          = 29.9899993297
iftype     = itime
internal0  = 2.0
iztype     = ib
kcmpnm     = EHZ
knetwk     = BW
kstnm      = RJOB
lcalda     = False
leven      = True
lovrok     = True
lpspol     = True
npts       = 3000
nvhdr      = 6
nzhour     = 0
nzjday     = 236
nzmin      = 20
nzmsec     = 0
nzsec      = 3
nzyear     = 2009
scale      = 1.0
```

```
tr2 = sac.to_obspy_trace()
print tr2.stats
```

```
         network: BW
         station: RJOB
        location:
         channel: EHZ
       starttime: 2009-08-24T00:20:03.000000Z
         endtime: 2009-08-24T00:20:32.990000Z
   sampling_rate: 100.0
```

```
delta: 0.01
 npts: 3000
calib: 1.0
  sac: AttribDict({'cmpaz': 0.0, 'nzyear': 2009, 'nzjday': 236,
  'iztype': 9, 'evla': 0.0, 'nzhour': 0, 'lcalda': 0, 'evlo': 0.0,
  'scale': 1.0, 'nvhdr': 6, 'depmin': -1515.8131, 'kcmpnm': 'EHZ',
  'nzsec': 3, 'internal0': 2.0, 'depmen': -4.495563, 'cmpinc': 0.0,
  'depmax': 1293.771, 'iftype': 1, 'delta': 0.0099999998, 'nzmsec':
  0, 'lpspol': 1, 'b': 0.0, 'e': 29.99, 'leven': 1, 'kstnm': 'RJOB',
  'nzmin': 20, 'lovrok': 1, 'npts': 3000, 'knetwk': 'BW'})
```

2 Package Organization

2.1 pysac.header

SAC header specification, including documentation.

Header names, order, and types, nulls, as well as allowed enumerated values, are specified here. Header name strings, and their array order are contained in separate float, int, and string tuples. Enumerated values, and their allowed string and integer values, are in dictionaries. Header value documentation is in a dictionary, `DOC`, for reuse throughout the package.

2.2 pysac.util

PySAC helper functions and data. Contains functions to validate and convert enumerated values, byteorder consistency checking, and SAC reference time reading.

Two of the most important functions in this module are `sac_to_obspy_header` and `obspy_to_sac_header`. These contain the conversion routines between SAC header dictionaries and ObsPy header dictionaries. **These functions control the way ObsPy reads and writes SAC files,** which was one of the main motivations for authoring this package.

2.3 pysac.arrayio

Low-level array interface to the SAC file format.

Functions in this module work directly with numpy arrays that mirror the SAC format, and comprise much of the machinery that underlies the `SACTrace` class. The 'primitives' in this module are the float, int, and string header arrays, the float data array, and a header dictionary. Convenience functions are provided to convert between header arrays and more user-friendly dictionaries.

These read/write routines are very literal; there is almost no value or type checking, except for byteorder and header/data array length. File- and array- based checking routines are provided for additional checks where desired.

Reading and writing are done with `read_sac` and `write_sac` for binary SAC files, and `read_sac_ascii` and `write_sac_ascii` for alphanumeric files. Conversions between header dictionaries and the three SAC header arrays are done with the `header_arrays_to_dict` and `dict_to_header_arrays` functions. Validation of header values and data is managed by `validate_sac_content`, which can currently do six different tests.

2.4 pysac.sactrace

Contains the `SACTrace` class, which is the main user-facing interface to the SAC file format.

The `SACTrace` object maintains consistency between SAC headers and manages header values in a user-friendly way. This includes some value-checking, native Python logicals and nulls instead of SAC's header-dependent logical/null representation.

2.4.1 Reading and writing SAC files

PySAC can read and write evenly-spaced time-series files. It supports big or little-endian binary files, or alphanumeric/ASCII files.

```
# read from a binary file
sac = SACTrace.read(filename)

# read header only
sac = SACTrace.read(filename, headonly=True)

# write header-only, file must exist
sac.write(filename, headonly=True)

# read from an ASCII file
sac = SACTrace.read(filename, ascii=True)

# write a binary SAC file for a Sun machine
sac.write(filename, byteorder='big')
```

2.4.2 Headers

In the `SACTrace` class, SAC headers are implemented as properties, with appropriate *getters* and *setters*. The getters/setters translate user-facing native Python values like `True`, `False`, and `None` to the appropriate SAC header values, like `1`, `0`, `-12345`, `'-12345  '`, etc.

Header values that depend on the SAC `.data` vector are calculated on-the-fly, and fall back to the stored header value.

A convenient read-only dictionary of non-null, raw SAC header values is available as `SACTrace._header`. Formatted non-null headers are viewable using `print(sac)` or the `.lh()` or `listhdr()` methods. Relative time headers and picks are viewable with `lh('picks')`.

2.4.3 Reference time and relative time header handling

The SAC reference time is built from "nz..." time fields in the header, and it is available as the attribute `.reftime`, an ObsPy `UTCDateTime` instance. `reftime` can be modified in two ways: by resetting it with a new absolute `UTCDateTime` instance, or by adding/subtracting seconds from it. **Modifying the `reftime` will also modify all relative time headers such that they are still correct in an absolute sense**. This includes `a`, `b`, `e`, `f`, `o`, and `t1-t9`. This means that adjusting the reference time does not invalidate the origin time, the first sample time, or any picks!

Here, we build a 100-second `SACTrace` that starts at Y2K.

```
sac = SACTrace(nzyear=2000, nzjday=1, nzhour=0, nzmin=0, nzsec=0, nzmsec=0,
               t1=23.5, data=numpy.arange(101))

sac.reftime
sac.b, sac.e, sac.t1
```

```
2000-01-01T00:00:00.000000Z
(0.0, 100.0, 23.5)
```

Move reference time by relative seconds, relative time headers are preserved.

```
sac.reftime -= 2.5
sac.b, sac.e, sac.t1
```

```
(2.5, 102.5, 26.0)
```

Set reference time to new absolute time, two minutes later. Relative time headers are preserved.

```
sac.reftime = UTCDateTime(2000, 1, 1, 0, 2, 0, 0)
sac.b, sac.e
```

```
(-120.0, -20.0, -96.5)
```

3 Relationship to ObsPy

PySAC is largely re-written from the obspy.io.sac module, with the intention of eventually replacing it.

3.1 Why a re-write?

The sacio module underlying ObsPy's SAC handling was sometimes hard to follow, as it has a long inheritance, which made it hard to track down issues or make fixes. This re-write attempts to make the SAC plugin easier to understand and maintain, as well as offer some potential improvements.

3.2 Improve maintainability.

I've split out the header specification (header.py), the low-level array-based SAC file I/O (arrayio.py), and the object-oriented interface (sactrace.py), whereas it was previously all within one sacio.py module. I hope that the flow of how each plugs into the other is clear, so that bug tracking is straight-forward, and that hacking on one aspect of SAC handling is not cluttered/distracted by another.

3.3 Expand support for round-trip SAC file processing

This rewrite attempts to improve support for a common work flow: read one or more SAC files into ObsPy, do some processing, then (over)write them back as SAC files that look mostly like the originals. Previously, ObsPy Traces written to SAC files wrote only files based on the first sample time (iztype 9/'ib'). In `util.py:obspy_to_sac_header` of this module, if an old tr.stats.sac SAC header is found, the iztype and reference "nz" times are used and kept, and the "b" and "e" times of the Trace being written are adjusted according to this reference time. This preserves the absolute reference of any relative time headers, like t0-t9, carried from the old SAC header into the new file. This can only be done if SAC to Trace conversion preserves these "nz" time headers, which is possible with the current `debug_headers=True` flag.

4 License