sac-format

C++20 SAC-file Library
User Manual

Alexander R. Blanchette*

2023-11-17

Contents

*arbCoding@gmail.com

1 Introduction

sac-format is a single-header statically linked library designed to make working with binary SAC-files as easy as possible. Written in C++20, it follows a modern and easy to read programming-style while providing the high performance brought by C++.

sac-format's actively developed on GitHub!

Download an offline version of the documentation (PDF).

1.1 Why sac-format

sac-format is Free and Open Source Software (FOSS) released under the MIT license. Anyone can use it, for any purpose (including proprietary software), anywhere in the world. sac-format is operating system agnostic and confirmed working on Windows, macOS, and Linux systems.

1.1.1 High-Level Trace Class

sac-format includes the `Trace` class for seismic traces, providing high-level object-oriented abstraction to seismic data. With the `Trace` class, you don't need to worry about manually reading SAC-files word-by-word. It's compatible with `v6` and `v7` SAC-files and can automatically detect the version upon reading. File output defaults to `v7` SAC-files and there is a *legacy-write* function for `v6` output.

1.1.2 Low-Level I/O

If you want to roll your own SAC-file processing workflow you can use the low-level I/O functionality built into sac-format. All functions tested and confirmed working—they're used to build the `Trace` class!

1.1.3 Safe

sac-format is **safe**—testing is an important part of software development. The sac-format library is extensively tested using the Catch2 testing framework. Everything from low-level binary conversions to high-level `Trace` reading/writing are tested and confirmed working. New tests added when they're imagined. Check and run the tests yourself—see `utests.cpp`.

1.1.4 Fast

sac-format is **fast**—it's written in C++ and extensively benchmarked. You can run the benchmarks yourself to find out how sac-format performs on your system.

1.1.5 Easy

sac-format is **easy**—single-header makes integration to any project simple. CMake makes building on different platforms a breeze. Object-oriented design makes use easy and intuitive. See the Quickstart section to get up and running.

1.1.6 Small

sac-format is **small**—in total (header + implementation–excluding comments) it's fewer than 1500 lines of code (compiles to ~200 KB). Small size opens to door to using on any sort of hardware (old or new) and makes it easy to expand upon.

1.1.7 Documented

sac-format is extensively **documented**—both online and in the code. Nothing's hidden—nothing's obscured. Curious how something works? Check the documentation and in-code comments.

2 Quickstart

2.1 Manual Instructions

2.1.1 Build Instructions

Building is as easy as cloning the repository, running CMake for your preferred build tool, and then building.

1. Ninja

   ```
   git clone https://github.com/arbCoding/sac-format.git
   mkdir bin && cd bin/
   cmake -G Ninja .. && ninja
   ```

2. Make

   ```
   git clone https://github.com/arbCoding/sac-format.git
   mkdir bin && cd bin/
   cmake .. && make
   ```

2.1.2 Use

To use link to the compiled library (`libsac-format.a` on Linux/macOS, `libsac-format.lib` on Windows) and include `src/sac_format.hpp`.

2.2 CMake Integration

To integrate sac-format into your CMake project, add it to your `CMakeLists.txt`.

```
include(FetchContent)
set(FETCHCONTENT_UPDATES_DISCONNECTED TRUE)
FetchContent_Declare(sac-format
   GIT_REPOSITORY https://github.com/arbCoding/sac-format
   GIT_TAG vX.X.X)
FetchContent_MakeAvailable(sac-format)
include_directory(${sacformat_SOURCE_DIR/src})

project (your_project
   LANGUAGES CXX)

add_executable(your_executable
   your_sources
   sac_format.hpp)

target_link_libraries_library(your_executable
   PRIVATE sac-format)
```

2.3 Examples

2.3.1 Reading and Writing

```
#include <filesystem>
#include <iostream>
#include <sac_format.hpp>

using namespace sacfmt;
namespace fs = std::filesystem;
```

```
int main() {
    Trace trace1{};
    // Change header variable
    trace1.kstnm("Station1");
    fs::path file{"./test.SAC"};
    // Write
    trace1.write(file);
    // Read
    Trace trace2 = Trace(file);
    // Confirm equality
    std::cout << (trace1 == trace2) << '\n';
    fs::remove(file);
    return EXIT_SUCCESS;
}
```

3 Documentation

3.1 Dependencies

3.1.1 Automatic (CMake)

1. Xoshiro-cpp v1.12.0 (testing and benchmarking)
2. Catch2 v3.4.0 (testing and benchmarking)

3.2 Trace class

3.2.1 Reading SAC

3.2.2 Writing SAC

3.2.3 Getters and Setters

3.2.4 Internal Structure

1. `floats` array
2. `doubles` array
3. `ints` array
4. `bools` array
5. `strings` array
6. `data` array
7. Lookup Table

3.3 Low-Level I/O

3.4 Debugging

3.5 Unit-testing

3.6 Benchmarking

3.7 SAC-file format

The official and up-to-date documentation for the SAC-file format is available from the EarthScope Consortium (formerly IRIS/UNAVCO) here. The following subsections constitute my notes on the format. Below is a quick guide—all credit for the creation of, and documentation for, the SAC file-format belongs to its developers and maintainers (details here).

3.7.1 Floating-point (39)

32-bit (1 word, 4 bytes)

1. `depmin`
   Minimum value of the dependent variable (displacement/velocity/acceleration/volts/counts).
2. `depmen`
   Mean value of the dependent variable.
3. `depmax`
   Maximum value of the dependent variable.
4. `odelta`
   Modified (*observational*) value of `delta`.
5. `resp(0--9)`
   Instrument response parameters (poles, zeros, and a constant).
   Not used by SAC—they're free for other purposes.
6. `stel`
   Station elevation in meters above sea level (*m.a.s.l*).
   Not used by SAC—free for other purposes.
7. `stdp`
   Station depth in meters below surface (borehole/buried vault).
   Not used by SAC—free for other purposes.
8. `evel`
   Event elevation *m.a.s.l.*
   Not used by SAC—free for other purposes.
9. `evdp`
   Event depth in kilometers (*previously meters*) below surface.
10. `mag`
    Event magnitude.
11. `user(0--9)`
    Storage for user-defined values.
12. `dist`
    Station–Event distance in kilometers.
13. `az`
    Azimuth $(\mathrm{Event} \rightarrow \mathrm{Station})$, decimal degrees from North.
14. `baz`
    Back-azimuth $(\mathrm{Station} \rightarrow \mathrm{Event})$, decimal degrees from North.
15. `gcarc`
    Station–Event great circle arc-length, decimal degrees.
16. `cmpaz`
    Instrument measurement azimuth, decimal degrees from North.

Value	Direction
0°	North
90°	East
180°	South
270°	West
Other	1/2/3

17. `cmpinc`

    Instrument measurement incident angle, decimal degrees from upward vertical (incident 0° = dip -90°).

Value	Direction
0°	Up
90°	Horizontal
180°	Down
270°	Horizontal

    NOTE: SEED/MINISEED use dip angle, decimal degrees down from horizontal (dip 0° = incident 90°).

18. `xminimum`

    Spectral-only equivalent of `depmin` (f_0 or ω_0).

19. `xmaximum`

    Spectral-only equivalent of `depmax` (f_{max} or ω_{max}).

20. `yminimum`

    Spectral-only equivalent of `b`.

21. `ymaximum`

    Spectral-only equivalent of `e`.

3.7.2 Double (22)

64-bit (2 words, 8 bytes)

NOTE: in the header section these are floats—they're doubles in the footer section of `v7` SAC-files. In memory they're stored as doubles regardless of the SAC-file version.

1. `delta`

   Increment between evenly spaced samples (Δt for timeseries, Δf or $\Delta\omega$ for spectra).

2. `b`

   First value (*begin*) of independent variable (t_0).

3. `e`

   Final value (*end*) of independent variable (t_{max}).

4. `o`

   Event *origin* time, in seconds relative to the reference time.

5. `a`

   Event first *arrival* time, in seconds relative to the reference time.

6. `t(0--9)`

   User defined *time* values, in seconds relative to the reference time.

7. `f`

   Event end (*fini*) time, in seconds relative to the reference time.

8. `stla`
   Station latitude in decimal degrees, N/S–positive/negative.
9. `stlo`
   Station longitude in decimal degrees, E/W–positive/negative.
10. `evla`
    Event latitude in decimal degrees, N/S–positive/negative.
11. `evlo`
    Event longitude in decimal degrees, E/W–positive/negative.
12. `sb`
    Original (*saved*) `b` value.
13. `sdelta`
    Original (*saved*) `delta` value.

3.7.3 Integer (26)

32-bit (1 word, 4 bytes)

1. `nzyear`
   Reference time GMT year.
2. `nzjday`
   Reference time GMT day-of-year (often called Julian Date) (1–366).
3. `nzhour`
   Reference time GMT hour (00–23).
4. `nzmin`
   Reference time GMT minute (0–59).
5. `nzsec`
   Reference time GMT second (0–59).
6. `nzmsec`
   Reference time GMT Millisecond (0–999).
7. `nvhdr`
   SAC-file version.

Version	Description
`v7`	Footer (2020+, sac 102.0+)
`v6`	No footer (pre-2020, sac 101.6a-)

8. `norid`
   Origin ID.
9. `nevid`
   Event ID.
10. `npts`
    Number of points in data.

11. `nsnpts`

    Original (*saved*) `npts`.

12. `nwfid`

    Waveform ID.

13. `nxsize`

    Spectral-only equivalent of `npts` (length of spectrum).

14. `nysize`

    Spectral-only, width of spectrum.

15. `iftype`

    File type.

Value	Type	Description
01	ITIME	Time-series
02	IRLIM	Spectral (real/imaginary)
03	IAMPH	Spectral (amplitude/phase)
04	IXY	General XY file
??	IXYZ∗	General XYZ file

    ∗Value not listed in the standard.

16. `idep`

    Dependent variable type.

Value	Type	Description
05	IUNKN	Unknown
06	IDISP	Displacement (nm)
07	IVEL	Velocity $\left(\frac{\mathrm{nm}}{\mathrm{s}}\right)$
08	IACC	Acceleration $\left(\frac{\mathrm{nm}}{\mathrm{s}^2}\right)$
50	IVOLTS	Velocity (volts)

17. `iztype`

    Reference time equivalent.

Value	Type	Description
05	IUNKN	Unknown
09	IB	Recording start time
10	IDAY	Midnight reference GMT day
11	IO	Event origin time
12	IA	First arrival time
13–22	IT(0–9)	User defined time (t) pick

18. `iinst`

    Recording instrument type.

    Not used by SAC—free for other purposes.

19. `istreg`

    Station geographic region.

    Not used by SAC—free for other purposes.

20. `ievreg`

    Event geographic region.

    Not used by SAC—free for other purposes.

21. `ievtyp`

    Event type.

Value	Type	Description
05	IUNKN	Unknown
11	IO	Other source of known origin
37	INUCL	Nuclear
38	IPREN	Nuclear pre-shot
39	IPOSTN	Nuclear post-shot
40	IQUAKE	Earthquake
41	IPREQ	Foreshock
42	IPOSTQ	Aftershock
43	ICHEM	Chemical explosion
44	IOTHER	Other
72	IQB	Quarry/mine blast—confirmed by quarry/mine
73	IQB1	Quarry/mine blast—designed shot info-ripple fired
74	IQB2	Quarry/mine blast—observed shot info-ripple fired
75	IQBX	Quarry/mine blast—single shot
76	IQMT	Quarry/mining induced events—tremor and rockbursts
77	IEQ	Earthquake
78	IEQ1	Earthquake in a swarm or in an aftershock sequence
79	IEQ2	Felt earthquake
80	IME	Marine explosion
81	IEX	Other explosion
82	INU	Nuclear explosion
83	INC	Nuclear cavity collapse
85	IL	Local event of unknown origin
86	IR	Region event of unknown origin
87	IT	Teleseismic event of unknown origin
88	IU	Undetermined/conflicting information

22. `iqual`

    Quality of data.

Value	Type	Description
44	IOTHER	Other
45	IGOOD	Good
46	IGLCH	Glitches
47	IDROP	Dropouts
48	ILOWSN	Low signal-to-noise ratio

    Not used by SAC—free for other purposes.

23. `isynth`

    Synthetic data flag.

Value	Type	Description
49	IRLDATA	Real data
XX	$*$	Synthetic

    $*$Values and types not listed in the standard.

24. `imagtyp`

    Magnitude type.

Value	Type	Description
52	IMB	Body-wave magnitude (M_b)
53	IMS	Surface-wave magnitude (M_s)
54	IML	Local magnitude (M_l)
55	IMW	Moment magnitude (M_w)
56	IMD	Duration magnitude (M_d)
57	IMX	User-defined magnitude (M_x)

25. `imagsrc`

    Source of magnitude information.

Value	Type	Description
58	INEIC	National Earthquake Information Center
61	IPDE	Preliminary Determination of Epicenter
62	IISC	Internation Seismological Centre
63	IREB	Reviewed Event Bulletin
64	IUSGS	U.S. Geological Survey
65	IBRK	UC Berkeley
66	ICALTECH	California Institute of Technology
67	ILLNL	Lawrence Livermore National Laboratory
68	IEVLOC	Event location (computer program)
69	IJSOP	Joint Seismic Observation Program
70	IUSER	The user
71	IUNKNOWN	Unknown

26. `ibody`

    Body/spheroid definition used to calculate distances.

Value	Type	Name	Semi-major axis (a [m])	Inverse Flattening (*f*)
-12345	UNDEF	Earth (*Historic*)	6378160.0	0.00335293
98	ISUN	Sun	696000000.0	8.189e-6
99	IMERCURY	Mercury	2439700.0	0.0
100	IVENUS	Venus	6051800.0	0.0
101	IEARTH	Earth (*WGS84*)	6378137.0	0.0033528106647474805
102	IMOON	Moon	1737400.0	0.0
103	IMARS	Mars	3396190.0	0.005886007555525457

3.7.4 Boolean (4)

32-bit (1 word, 4 bytes) in-file/8-bit (1 byte) in-memory

1. `leven`

   REQUIRED

   Evenly-spaced data flag.

   If true, then data is evenly spaced.

2. `lpspol`

   Station polarity flag.

   If true, then station has positive-polarity—it follows the left-hand convention (for example, North-East-Up [NEZ]).

3. `lovrok`

   File overwrite flag.

   If true, then it's okay to overwrite the file.

4. `lcalda`

   Calculate geometry flag.

   If true, then calculate `dist`, `az`, `baz`, and `gcarc` from `stla`, `stlo`, `evla`, and `evlo`.

3.7.5 String (23)

32/64-bit (2/4 words, 8/16 bytes, 8/16 characters)

1. `kstnm`

   Station name.

2. `kevnm*`

   Event name.

   *This is the **only** four word (16 character) string.

3. `khole`

   Nuclear—hole identifier.

   Other—Location identifier (LOCID).

4. `ko`

   Text for `o`.

5. `ka`

   Text for `a`.

6. `kt(0--9)`

   Text for `t(0--9)`.

7. `kf`

   Text for `f`.

8. `kuser(0--2)`

   Text for the first three of `user(0--9)`.

9. `kdatrd`

   Date the data was read onto a computer.

10. `kinst`

    Text for `iinst`.

3.7.6 Data (2)

32-bit (2 words, 8 bytes) in-file/64-bit (4 words, 16 bytes) in-memory
Stored as floating-point (32-bit) values in SAC-files; stored as double-precision in memory.

1. `data1`

   The first data vector—**always** present in a SAC-file and begins at word 158.

2. `data2`

   The second data vector—**conditionally** present and begins after `data1`.

   Required if `leven` is false, or if `iftype` is spectral/XY/XYZ.

4 Notes

4.1 Why C++20 and not C++23

Compiler restrictions—C++23 support requires GCC-13+ and Clang-16+. Many systems, still use GCC-12 and Clang-15—which has near complete support for C++20.

sac-format strives for accessibility, modernity, safety, and speed—C++20 provides the best fit.