

SAC-FORMAT

C++20 SAC-file Library

User Manual

Version: 0.4.0

ALEXANDER R. BLANCHETTE

Documentation Version: 2023-12-06 16:28:36-08:00

Contents

| | | |
|----------|-----------------------------------|-----------|
| 1 | Introduction | 3 |
| 1.1 | Why sac-format | 3 |
| 2 | Quickstart | 4 |
| 2.1 | Manual Instructions | 4 |
| 2.2 | Example Programs | 4 |
| 2.3 | CMake Integration | 4 |
| 2.4 | Example | 5 |
| 3 | Documentation | 5 |
| 3.1 | Convenience Functions | 5 |
| 3.2 | Trace class | 6 |
| 3.3 | Low-Level I/O | 8 |
| 3.4 | Testing | 9 |
| 3.5 | Benchmarking | 9 |
| 3.6 | Source File List | 9 |
| 3.7 | Dependencies | 10 |
| 3.8 | SAC-file format | 10 |
| 4 | Notes | 16 |
| 4.1 | Why C++20 and not C++23 | 16 |

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 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. Safe

sac-format is **safe**—it conforms to a strict set of C++ programming guidelines, chosen to ensure safe code-execution. The guideline conformance list is in `cpp-linter.yml` and can be cross-referenced against this [master list](#). Results of conformance checking is [here](#).

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. Check and run the tests yourself. See the Testing section for more information.

2. Fast

sac-format is **fast**—it's written in C++, carefully optimized, and extensively benchmarked. You can run the benchmarks yourself to find out how sac-format performs on your system. See the Benchmarking section for more information.

3. Easy

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

4. Small

sac-format is **small**—in total (header + implementation—excluding comments) it's fewer than 2000 lines of code. Small size opens the door to using on any sort of hardware (old or new) and makes it easy to expand upon.

5. 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.

6. Transparent

sac-format is **transparent**—all [analysis](#) and [coverage](#) information is publicly available online.

[CodeFactor Analysis](#)

7. 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.

8. 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!

2 Quickstart

2.1 Manual Instructions

1. Build Instructions

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

(a) GCC

```
git clone https://github.com/arbCoding/sac-format.git
cmake --preset gcc-release
cmake --build ./build/release/gcc
```

(b) Clang

```
git clone https://github.com/arbCoding/sac-format.git
cmake --preset clang-release
cmake --build ./build/release/clang
```

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 Example Programs

1. `list_sac`

`list_sac` is a command line program that takes a single SAC-file as its input argument. It reads the SAC-file and outputs the header/footer information, as well as the true size of the `data1` and `data2` vectors.

2.3 CMake Integration

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

```
1  include(FetchContent)
2  set(FETCHCONTENT_UPDATES_DISCONNECTED TRUE)
3  FetchContent_Declare(sac-format
4      GIT_REPOSITORY https://github.com/arbCoding/sac-format
5      GIT_TAG vX.X.X)
6  FetchContent_MakeAvailable(sac-format)
7  include_directories(${sacformat_SOURCE_DIR/src})
8
9  project(your_project
10     LANGUAGES CXX)
11
12  add_executable(your_executable
13     your_sources
14     sac_format.hpp)
15
16  target_link_libraries_library(your_executable
17     PRIVATE sac-format)
```

2.4 Example

1. Reading and Writing

```

1  #include <filesystem>
2  #include <iostream>
3  #include <sac_format.hpp>
4
5  using namespace sacfmt;
6  namespace fs = std::filesystem;
7
8  int main() {
9      Trace trace1{};
10     // Change header variable
11     trace1.kstnm("Station1");
12     fs::path file{"/test.SAC"};
13     // Write
14     trace1.write(file);
15     // Read
16     Trace trace2 = Trace(file);
17     // Confirm equality
18     std::cout << (trace1 == trace2) << '\n';
19     fs::remove(file);
20     return EXIT_SUCCESS;
21 }

```

3 Documentation

3.1 Convenience Functions

- degrees_to_radians

Convert decimal degrees to radians.

```
1  double radians{sacfmt::degrees_to_radians(degrees)};
```

- radians_to_degrees

Convert radians to decimal degrees.

```
1  double degrees{sacfmt::radians_to_degrees(radians)};
```

- gcarc

Calculate great-circle arc distance (spherical planet).

```
1  double gcarc{sacfmt::gcarc(latitude1, longitude1, latitude2, longitude2)};
```

- azimuth

Calculate azimuth between two points (spherical planet).

```
1  double azimuth{sacfmt::azimuth(latitude2, longitude2, latitude1,  
   ↪ longitude2)};  
2  double back_azimuth{sacfmt::azimuth(latitude1, longitude1, latitude2,  
   ↪ longitude2)};
```

- limit_360

Take arbitrary value of degrees and unwrap to [0,360].

```
1  double degrees_limited{sacfmt::limit_360(degrees)};
```

- limit_180

Take arbitrary value of degrees and unwrap to [−180,180]. Useful for longitude.

```
1  double degrees_limited{sacfmt::limit_180(degrees)};
```

- limit_90

Take arbitrary value of degrees and unwrap to [−90,90]. Useful for latitude.

```
1  double degrees_limited{sacfmt::limit_90(degrees)};
```

3.2 Trace class

The Trace class provides easy access to SAC-files in C++. Each SAC-file is a Trace; therefore, each Trace object is a seismic trace (seismogram).

1. Reading SAC

SAC-files can be read in by using the parameterized constructor with a `std::filesystem::path(<filesystem>)` or a `std::string(<string>)` variable that corresponds to the location of the SAC-file.

For example:

```
1  #include <filesystem>  
2  #include <sac_foramt.hpp>  
3  
4  int main() {  
5      std::filesystem::path my_file{"/home/user/data/ANMO.SAC"};  
6      sacfmt::Trace anmo = Trace(my_file);  
7      return EXIT_SUCCESS;  
8  }
```

2. Writing SAC

Writing SAC files can be done using one of two write functions.

(a) v7 files

Use `write` (for example `trace.write(filename)`).

(b) v6 files

Use `legacy_write` (for example `trace.legacy_write(filename)`).

3. Getters and Setters

Every SAC variable is accessed via getters and setters of the same name.

(a) Example Getters

- `trace.npts()`
- `trace.data1()`
- `trace.kstnm()`

(b) Example Setters

- `trace.kevnm("Event 1")`
- `trace.evla(32.89)`
- `trace.mag(3.21)`

4. Internal Structure

The SAC-trace stores the data internally in a series of pre-allocated `std::array (<array>)` container objects. Getters and setters access these via a lookup table. The internal components are below:

- (a) Lookup Table `sac_map`
- (b) `floats` array
- (c) `doubles` array
- (d) `ints` array
- (e) `bools` array
- (f) `strings` array
- (g) `data` array

5. Convenience Methods

- `calc_geometry`

Calculate `gcarc`, `dist`, `az`, and `baz` assuming spherical Earth.

```
1 trace.stla(45.3);
2 trace.stlo(34.5);
3 trace.evla(18.5);
4 trace.evlo(-34);
5 trace.calc_geometry();
6 std::cout << "GcArc: " << trace.gcarc() << '\n';
7 std::cout << "Dist: " << trace.dist() << '\n';
8 std::cout << "Azimuth: " << trace.az() << '\n';
9 std::cout << "BAzimuth: " << trace.baz() << '\n';
```

- `frequency`

Calculate frequency from `delta`.

```
1  double frequency{trace.frequency()};
```

- `date`

Return `std::string` formatted as `YYYY-JJJ` from `nzyear` and `nzjday`.

```
1  std::string date{trace.date()};
```

- `time`

Return `std::string` formatted as `HH:MM:SS.xxx` from `nzhour`, `nzmin`, `nzsec`, and `nzmsec`.

```
1  std::string time{trace.time()};
```

6. Exceptions

`sac-format` throws exceptions of type `sacfmt::io_error` in the event of a failure to read/write a SAC-file.

3.3 Low-Level I/O

Low-level I/O functions are as follows:

1. Binary conversion

- `int_to_binary` and `binary_to_int`
Conversion pair for binary representation of integer values.
- `float_to_binary` and `binary_to_float`
Conversion pair for binary representation of floating-point values.
- `double_to_binary` and `binary_to_double`
Conversion pair for binary representation of double-precision values.
- `string_to_binary` and `binary_to_string`
Conversion pair for binary representation of two-word (regular) string values.
- `long_string_to_binary` and `binary_to_long_string`
Conversion pair for binary representation of four-word (only `kstnm`) string values.

2. Reading/Writing

- `read_word`, `read_two_words`, `read_four_words`, and `read_data`
Functions to read one-, two-, and four-word variables (depending on the header) and an arbitrary amount of binary data (exclusive to `data1` and `data2`).
- `convert_to_word`, `convert_to_words`, and `bool_to_word`
Takes objects and converts them into `std::vector<char>` (`convert_to_word` and `bool_to_word`) or `std::array<char, N>` (`convert_to_words`, `N` = # of words).
- `write_words`
Writes input words (as `std::vector<char>`) to a binary SAC-file.

3. Utility

- (a) `concat_words`
Concatenates words taking into account the system endianness.
- (b) `bits_string` and `string_bits`
Template function that performs conversion of binary strings of arbitrary length to an arbitrary number of words.
- (c) `remove_leading_spaces` and `remove_trailing_spaces`
Remove leading and trailing blank spaces from strings assuming ASCII convention (space character is integer 32, below that value are control characters that also appear as blank spaces).
- (d) `string_cleaning`
Ensures string does not contain an internal termination character (`\0`) and removes it if present, then removes blank spaces.
- (e) `prep_string`
Performs `string_cleaning` followed by string truncation/padding to the necessary length.
- (f) `equal_within_tolerance`
Floating-point/double-precision equality within a provided tolerance (default is `f_eps`, defined in `sac_format.hpp`).

3.4 Testing

`utests.cpp` contains the unit- and integration-tests, using Catch2. Test coverage details are visible on [CodeCov.io](#) and [Codacy.com](#). All tests can be locally-run to ensure full functionality and compliance.

1. Errors only

By default `utests` prints out a pass summary, without details unless an error is encountered.

2. Full output

By passing the `--success` flag (`utests --success`) you can see the full results of all tests.

3. Compact output

The full output is verbose, using the compact reporter will condense the test results (`utests --reporter=compact --success`).

4. Additional options

To see additional options, run `utests -?`.

3.5 Benchmarking

`benchmark.cpp` contains the benchmarks. Running it locally will provide information on how long each function takes; benchmarks start with the low-level I/O function and build up to Trace reading, writing, and equality comparison.

To view available optional flags, run `benchmark -?`.

3.6 Source File List

1. Core

The two core files are split in the standard interface (`hpp`)/implementation (`cpp`) format.

- (a) `sac_format.hpp`
Interface—function declarations and constants.
- (b) `sac_format.cpp`
Implementation—function details.

2. Testing and Benchmarking

- (a) `util.hpp`
Utility functions and constants exclusive to testing and benchmarking. Not split into interface/implementation.
- (b) `utests.cpp`
- (c) `benchmark.cpp`

3. Example programs

- (a) `list_sac.cpp`

3.7 Dependencies

1. Automatic (CMake)

- (a) **Xoshiro-cpp v1.12.0** (testing and benchmarking)
- (b) **Catch2 v3.4.0** (testing and benchmarking)

3.8 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](#)).

1. Floating-point (39)

32-bit (1 word, 4 bytes)

- (a) `depmin`
Minimum value of the dependent variable (displacement/velocity/acceleration/volts/counts).
- (b) `depmen`
Mean value of the dependent variable.
- (c) `depmax`
Maximum value of the dependent variable.
- (d) `odelta`
Modified (*observational*) value of `delta`.
- (e) `resp(0--9)`
Instrument response parameters (poles, zeros, and a constant).
Not used by SAC—they’re free for other purposes.
- (f) `stel`
Station elevation in meters above sea level (*m.a.s.l.*).
Not used by SAC—free for other purposes.
- (g) `stdp`
Station depth in meters below surface (borehole/buried vault).
Not used by SAC—free for other purposes.
- (h) `evel`
Event elevation *m.a.s.l.*
Not used by SAC—free for other purposes.
- (i) `evdp`
Event depth in kilometers (*previously meters*) below surface.
- (j) `mag`
Event magnitude.

- (k) `user (0--9)`
Storage for user-defined values.
- (l) `dist`
Station–Event distance in kilometers.
- (m) `az`
Azimuth (Event \rightarrow Station), decimal degrees from North.
- (n) `baz`
Back-azimuth (Station \rightarrow Event), decimal degrees from North.
- (o) `gcarc`
Station–Event great circle arc-length, decimal degrees.
- (p) `cmpaz`
Instrument measurement azimuth, decimal degrees from North.

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

- (q) `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°).

- (r) `xminimum`
Spectral-only equivalent of `depmin` (f_0 or ω_0).
- (s) `xmaximum`
Spectral-only equivalent of `depmax` (f_{max} or ω_{max}).
- (t) `yminimum`
Spectral-only equivalent of `b`.
- (u) `ymaximum`
Spectral-only equivalent of `e`.

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.

- (a) `delta`
Increment between evenly spaced samples (Δt for timeseries, Δf or $\Delta \omega$ for spectra).
- (b) `b`
First value (*begin*) of independent variable (t_0).
- (c) `e`
Final value (*end*) of independent variable (t_{max}).

- (d) o
Event *origin* time, in seconds relative to the reference time.
- (e) a
Event first *arrival* time, in seconds relative to the reference time.
- (f) t (0--9)
User defined *time* values, in seconds relative to the reference time.
- (g) f
Event end (*fini*) time, in seconds relative to the reference time.
- (h) stla
Station latitude in decimal degrees, N/S--positive/negative.
sacfmt automatically enforces stla $\in [-90,90]$.
- (i) stlo
Station longitude in decimal degrees, E/W--positive/negative.
sacfmt automatically enforces stlo $\in [-180,180]$.
- (j) evla
Event latitude in decimal degrees, N/S--positive/negative.
sacfmt automatically enforces evla $\in [-90,90]$.
- (k) evlo
Event longitude in decimal degrees, E/W--positive/negative.
sacfmt automatically enforces evlo $\in [-180,180]$.
- (l) sb
Original (*saved*) b value.
- (m) sdelta
Original (*saved*) delta value.

3. Integer (26)

32-bit (1 word, 4 bytes)

- (a) nzyear
Reference time GMT year.
- (b) nzjday
Reference time GMT day-of-year (often called **Julian Date**) (1–366).
- (c) nzhour
Reference time GMT hour (00–23).
- (d) nzmin
Reference time GMT minute (0–59).
- (e) nzsec
Reference time GMT second (0–59).
- (f) nzmsec
Reference time GMT Millisecond (0–999).
- (g) nvhdr
SAC-file version.

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

- (h) norid
Origin ID.

- (i) `nevid`
Event ID.
- (j) `npts`
Number of points in data.
- (k) `nsnpts`
Original (*saved*) `npts`.
- (l) `nwfid`
Waveform ID.
- (m) `nxsize`
Spectral-only equivalent of `npts` (length of spectrum).
- (n) `nysize`
Spectral-only, width of spectrum.
- (o) `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.

- (p) `idep`
Dependent variable type.

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

- (q) `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 |

- (r) `iinst`
Recording instrument type.
Not used by SAC—free for other purposes.
- (s) `istreg`
Station geographic region.
Not used by SAC—free for other purposes.
- (t) `ievreg`
Event geographic region.
Not used by SAC—free for other purposes.

(u) `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 |

(v) `igual`

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.(w) `isynth`

Synthetic data flag.

| Value | Type | Description |
|-------|---------|-------------|
| 49 | IRLDATA | Real data |
| XX | * | Synthetic |

*Values and types not listed in the standard.

(x) `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) |

(y) `imagsrc`

Source of magnitude information.

| Value | Type | Description |
|-------|----------|--|
| 58 | INEIC | National Earthquake Information Center |
| 61 | IPDE | Preliminary Determination of Epicenter |
| 62 | IISC | International 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 |

(z) `ibody`

Body/spheroid definition used to calculate distances.

| Value | Type | Name | Semi-major axis (a [m]) | Inverse Flattening (f) |
|--------|----------|---------------------------|-------------------------|------------------------|
| -12345 | UNDEF | Earth (<i>Historic</i>) | 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 (<i>WGS84</i>) | 6378137.0 | 0.0033528106647474805 |
| 102 | IMOON | Moon | 1737400.0 | 0.0 |
| 103 | IMARS | Mars | 3396190.0 | 0.005886007555525457 |

4. Boolean (4)

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

(a) `leven`**REQUIRED**

Evenly-spaced data flag.

If true, then data is evenly spaced.

(b) `lpspol`

Station polarity flag.

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

(c) `lovrok`

File overwrite flag.

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

(d) `lcalda`

Calculate geometry flag.

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

5. String (23)

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

- (a) `kstnm`
Station name.
- (b) `kevnrm*`
Event name.
*This is the **only** four word (16 character) string.
- (c) `khole`
Nuclear—hole identifier.
Other—Location identifier (LOCID).
- (d) `ko`
Text for `o`.
- (e) `ka`
Text for `a`.
- (f) `kt (0--9)`
Text for `t (0--9)`.
- (g) `kf`
Text for `f`.
- (h) `kuser (0--2)`
Text for the first three of `user (0--9)`.
- (i) `kdatrd`
Date the data was read onto a computer.
- (j) `kinst`
Text for `inst`.

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.

- (a) `data1`
The first data vector—**always** present in a SAC-file and begins at word 158.
- (b) `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.