sac-format

0.6.0

Generated by Doxygen 1.9.8

1 Introduction 1

1.1 Why sac-format . . . 1

1.1.1 Safe . . . 1

1.1.2 Fast . . . 1

1.1.3 Easy . . . 2

1.1.4 Small . . . 2

1.1.5 Documented . . . 2

1.1.6 Transparent . . . 2

1.1.7 Trace Class . . . 2

1.1.8 Low-Level I/O . . . 2

2 Installation 3

2.1 Windows . . . 3

2.2 macOS . . . 9

2.2.1 Graphical . . . 9

2.2.2 Command line . . . 12

2.2.2.1 Self-Extracting Archive . . . 12

2.2.2.2 Gzipped Tar Archive . . . 12

2.3 Linux . . . 13

2.3.1 Debian Archive . . . 13

2.3.2 RPM Archive . . . 13

2.3.3 Self-Extrating Archive . . . 13

2.3.4 Gzipped Tar Archive . . . 13

3 Quickstart 15

3.1 Example Programs . . . 15

3.1.1 list_sac . . . 15

3.2 CMake Integration . . . 15

3.3 Example . . . 16

3.3.1 Reading and Writing . . . 16

4 Basic Documentation 17

4.1 Trace class . . . 17

4.1.1 Reading SAC . . . 17

4.1.2 Writing SAC . . . 17

4.1.2.1 v7 files . . . 17

4.1.2.2 v6 files . . . 17

4.1.3 Getters and Setters . . . 18

4.1.3.1 Example Getters . . . 18

4.1.3.2 Example Setters . . . 18

4.1.3.3 Setter rules . . . 18

4.1.4 Convenience Methods . . . 20

4.1.4.1 calc_geometry . . . 20

4.1.4.2 frequency . . . 20
4.1.4.3 date . . . 20
4.1.4.4 time . . . 20
4.1.5 Exceptions . . . 20
4.2 Convenience Functions . . . 20
4.2.1 degrees_to_radians . . . 20
4.2.2 radians_to_degrees . . . 21
4.2.3 gcarc . . . 21
4.2.4 azimuth . . . 21
4.2.5 limit_360 . . . 21
4.2.6 limit_180 . . . 21
4.2.7 limit_90 . . . 21
4.3 Low-Level I/O . . . 21
4.3.1 Binary conversion . . . 21
4.3.1.1 int_to_binary and binary_to_int . . . 21
4.3.1.2 float_to_binary and binary_to_float . . . 22
4.3.1.3 double_to_binary and binary_to_double . . . 22
4.3.1.4 string_to_binary and binary_to_string . . . 22
4.3.1.5 long_string_to_binary and binary_to_long_string . . . 22
4.3.2 Reading/Writing . . . 22
4.3.2.1 read_word, read_two_words, read_four_words, and read_data . . . 22
4.3.2.2 convert_to_word, convert_to_words, and bool_to_word . . . 22
4.3.2.3 write_words . . . 22
4.3.3 Utility . . . 23
4.3.3.1 concat_words . . . 23
4.3.3.2 bits_string and string_bits . . . 23
4.3.3.3 remove_leading_spaces and remove_trailing_spaces . . . 23
4.3.3.4 string_cleaning . . . 23
4.3.3.5 prep_string . . . 23
4.3.3.6 equal_within_tolerance . . . 23
4.4 Testing . . . 23
4.4.1 Errors only . . . 24
4.4.2 Full output . . . 24
4.4.3 Compact output . . . 24
4.4.4 Additional options . . . 24
4.4.5 Using ctest . . . 24
4.5 Benchmarking . . . 24
4.6 Source File List . . . 24
4.6.1 Core . . . 24
4.6.1.1 sac_format.hpp . . . 24
4.6.1.2 sac_format.cpp . . . 24
4.6.2 Testing and Benchmarking . . . 25

4.6.2.1 util.hpp . . . 25
4.6.2.2 utests.cpp . . . 25
4.6.2.3 benchmark.cpp . . . 25
4.6.3 Example programs . . . 25
4.6.3.1 list_sac.cpp . . . 25

5 SAC-file format **27**

5.1 Floating-point (39) . . . 27
5.1.1 depmin . . . 27
5.1.2 depmen . . . 27
5.1.3 depmax . . . 27
5.1.4 odelta . . . 27
5.1.5 resp(0–9) . . . 28
5.1.6 stel . . . 28
5.1.7 stdp . . . 28
5.1.8 evel . . . 28
5.1.9 evdp . . . 28
5.1.10 mag . . . 28
5.1.11 user(0–9) . . . 28
5.1.12 dist . . . 28
5.1.13 az . . . 28
5.1.14 baz . . . 29
5.1.15 gcarc . . . 29
5.1.16 cmpaz . . . 29
5.1.17 cmpinc . . . 29
5.1.18 xminimum . . . 29
5.1.19 xmaximum . . . 29
5.1.20 yminimum . . . 29
5.1.21 ymaximum . . . 30
5.2 Double (22) . . . 30
5.2.1 delta . . . 30
5.2.2 b . . . 30
5.2.3 e . . . 30
5.2.4 o . . . 30
5.2.5 a . . . 30
5.2.6 t(0–9) . . . 30
5.2.7 f . . . 30
5.2.8 stla . . . 31
5.2.9 stlo . . . 31
5.2.10 evla . . . 31
5.2.11 evlo . . . 31
5.2.12 sb . . . 31

5.2.13 sdelta . . . 31
5.3 Integer (26) . . . 31
5.3.1 nzyear . . . 31
5.3.2 nzjday . . . 32
5.3.3 nzhour . . . 32
5.3.4 nzmin . . . 32
5.3.5 nzsec . . . 32
5.3.6 nzmsec . . . 32
5.3.7 nvhdr . . . 32
5.3.8 norid . . . 32
5.3.9 nevid . . . 32
5.3.10 npts . . . 32
5.3.11 nsnpts . . . 33
5.3.12 nwfid . . . 33
5.3.13 nxsize . . . 33
5.3.14 nysize . . . 33
5.3.15 iftype . . . 33
5.3.16 idep . . . 33
5.3.17 iztype . . . 34
5.3.18 iinst . . . 34
5.3.19 istreg . . . 34
5.3.20 ievreg . . . 34
5.3.21 ievtyp . . . 34
5.3.22 iqual . . . 35
5.3.23 isynth . . . 35
5.3.24 imagtyp . . . 35
5.3.25 imagsrc . . . 36
5.3.26 ibody . . . 36
5.4 Boolean (4) . . . 36
5.4.1 leven . . . 36
5.4.2 lpspol . . . 37
5.4.3 lovrok . . . 37
5.4.4 lcalda . . . 37
5.5 String (23) . . . 37
5.5.1 kstnm . . . 37
5.5.2 kevnm . . . 37
5.5.3 khole . . . 37
5.5.4 ko . . . 37
5.5.5 ka . . . 38
5.5.6 kt(0–9) . . . 38
5.5.7 kf . . . 38
5.5.8 kuser(0–2) . . . 38

5.5.9 kdatrd . . . 38
5.5.10 kinst . . . 38
5.6 Data (2) . . . 38
5.6.1 data1 . . . 38
5.6.2 data2 . . . 38

6 Build Instructions **39**
6.1 Dependencies . . . 39
6.1.1 Automatic (CMake) . . . 39
6.1.2 Manual . . . 39
6.1.2.1 macOS and Linux . . . 39
6.2 Building . . . 39
6.2.1 GCC . . . 39
6.2.2 Clang . . . 40
6.2.3 MSVC . . . 40

7 Namespace Index **41**
7.1 Namespace List . . . 41

8 Hierarchical Index **43**
8.1 Class Hierarchy . . . 43

9 Class Index **45**
9.1 Class List . . . 45

10 Namespace Documentation **47**
10.1 sacfmt Namespace Reference . . . 47
10.1.1 Detailed Description . . . 51
10.1.2 Typedef Documentation . . . 51
10.1.2.1 char_bit . . . 51
10.1.2.2 unsigned_int . . . 52
10.1.2.3 word_four . . . 52
10.1.2.4 word_one . . . 52
10.1.2.5 word_two . . . 52
10.1.3 Enumeration Type Documentation . . . 52
10.1.3.1 name . . . 52
10.1.4 Function Documentation . . . 58
10.1.4.1 azimuth() . . . 58
10.1.4.2 binary_to_bool() . . . 60
10.1.4.3 binary_to_double() . . . 61
10.1.4.4 binary_to_float() . . . 62
10.1.4.5 binary_to_int() . . . 62
10.1.4.6 binary_to_long_string() . . . 63
10.1.4.7 binary_to_string() . . . 64

10.1.4.8 bits_string() . . . 65
10.1.4.9 bool_to_binary() . . . 66
10.1.4.10 bool_to_word() . . . 66
10.1.4.11 concat_words() [1/2] . . . 67
10.1.4.12 concat_words() [2/2] . . . 68
10.1.4.13 convert_to_word() [1/4] . . . 68
10.1.4.14 convert_to_word() [2/4] . . . 68
10.1.4.15 convert_to_word() [3/4] . . . 69
10.1.4.16 convert_to_word() [4/4] . . . 69
10.1.4.17 convert_to_words() [1/2] . . . 69
10.1.4.18 convert_to_words() [2/2] . . . 69
10.1.4.19 degrees_to_radians() . . . 70
10.1.4.20 double_to_binary() . . . 71
10.1.4.21 equal_within_tolerance() [1/2] . . . 71
10.1.4.22 equal_within_tolerance() [2/2] . . . 71
10.1.4.23 float_to_binary() . . . 73
10.1.4.24 gcarc() . . . 73
10.1.4.25 int_to_binary() . . . 74
10.1.4.26 limit_180() . . . 75
10.1.4.27 limit_360() . . . 76
10.1.4.28 limit_90() . . . 77
10.1.4.29 long_string_to_binary() . . . 78
10.1.4.30 nwords_after_current() . . . 78
10.1.4.31 prep_string() . . . 79
10.1.4.32 radians_to_degrees() . . . 80
10.1.4.33 read_data() . . . 81
10.1.4.34 read_four_words() . . . 82
10.1.4.35 read_two_words() . . . 83
10.1.4.36 read_word() . . . 84
10.1.4.37 remove_leading_spaces() . . . 85
10.1.4.38 remove_trailing_spaces() . . . 85
10.1.4.39 safe_to_finish_reading() . . . 86
10.1.4.40 safe_to_read_data() . . . 87
10.1.4.41 safe_to_read_footer() . . . 88
10.1.4.42 safe_to_read_header() . . . 89
10.1.4.43 string_bits() . . . 90
10.1.4.44 string_cleaning() . . . 91
10.1.4.45 string_to_binary() . . . 92
10.1.4.46 uint_to_binary() . . . 92
10.1.4.47 word_position() . . . 93
10.1.4.48 write_words() . . . 94
10.1.5 Variable Documentation . . . 94

10.1.5.1 ascii_space . . . 94
10.1.5.2 binary_word_size . . . 94
10.1.5.3 bits_per_byte . . . 94
10.1.5.4 circle_deg . . . 95
10.1.5.5 common_skip_num . . . 95
10.1.5.6 data_word . . . 95
10.1.5.7 deg_per_rad . . . 95
10.1.5.8 earth_radius . . . 95
10.1.5.9 f_eps . . . 95
10.1.5.10 modern_hdr_version . . . 95
10.1.5.11 num_bool . . . 96
10.1.5.12 num_data . . . 96
10.1.5.13 num_double . . . 96
10.1.5.14 num_float . . . 96
10.1.5.15 num_footer . . . 96
10.1.5.16 num_int . . . 96
10.1.5.17 num_string . . . 96
10.1.5.18 old_hdr_version . . . 97
10.1.5.19 rad_per_deg . . . 97
10.1.5.20 sac_map . . . 97
10.1.5.21 unset_bool . . . 98
10.1.5.22 unset_double . . . 99
10.1.5.23 unset_float . . . 99
10.1.5.24 unset_int . . . 99
10.1.5.25 unset_word . . . 99
10.1.5.26 word_length . . . 99
10.2 sacfmt::bitset_type Namespace Reference . . . 99
10.2.1 Detailed Description . . . 100
10.2.2 Variable Documentation . . . 100
10.2.2.1 bytes . . . 100

11 Class Documentation **101**

11.1 sacfmt::io_error Class Reference . . . 101
11.1.1 Detailed Description . . . 102
11.1.2 Constructor & Destructor Documentation . . . 102
11.1.2.1 io_error() . . . 102
11.1.3 Member Function Documentation . . . 102
11.1.3.1 what() . . . 102
11.1.4 Member Data Documentation . . . 103
11.1.4.1 message . . . 103
11.2 sacfmt::read_spec Struct Reference . . . 103
11.2.1 Detailed Description . . . 103

11.2.2 Member Data Documentation . . . 103
11.2.2.1 num_words . . . 103
11.2.2.2 start_word . . . 103
11.3 sacfmt::Trace Class Reference . . . 104
11.3.1 Detailed Description . . . 109
11.3.2 Constructor & Destructor Documentation . . . 109
11.3.2.1 Trace() [1/2] . . . 109
11.3.2.2 Trace() [2/2] . . . 109
11.3.3 Member Function Documentation . . . 112
11.3.3.1 a() [1/2] . . . 112
11.3.3.2 a() [2/2] . . . 113
11.3.3.3 az() [1/2] . . . 113
11.3.3.4 az() [2/2] . . . 113
11.3.3.5 b() [1/2] . . . 113
11.3.3.6 b() [2/2] . . . 114
11.3.3.7 baz() [1/2] . . . 114
11.3.3.8 baz() [2/2] . . . 114
11.3.3.9 calc_az() . . . 114
11.3.3.10 calc_baz() . . . 115
11.3.3.11 calc_dist() . . . 116
11.3.3.12 calc_gcarc() . . . 117
11.3.3.13 calc_geometry() . . . 118
11.3.3.14 cmpaz() [1/2] . . . 119
11.3.3.15 cmpaz() [2/2] . . . 119
11.3.3.16 cmpinc() [1/2] . . . 120
11.3.3.17 cmpinc() [2/2] . . . 120
11.3.3.18 data1() [1/2] . . . 120
11.3.3.19 data1() [2/2] . . . 120
11.3.3.20 data2() [1/2] . . . 121
11.3.3.21 data2() [2/2] . . . 121
11.3.3.22 date() . . . 121
11.3.3.23 delta() [1/2] . . . 122
11.3.3.24 delta() [2/2] . . . 122
11.3.3.25 depmax() [1/2] . . . 122
11.3.3.26 depmax() [2/2] . . . 123
11.3.3.27 depmen() [1/2] . . . 123
11.3.3.28 depmen() [2/2] . . . 123
11.3.3.29 depmin() [1/2] . . . 124
11.3.3.30 depmin() [2/2] . . . 124
11.3.3.31 dist() [1/2] . . . 124
11.3.3.32 dist() [2/2] . . . 124
11.3.3.33 e() [1/2] . . . 125

11.3.3.34 e() [2/2] . . . 125
11.3.3.35 evdp() [1/2] . . . 125
11.3.3.36 evdp() [2/2] . . . 125
11.3.3.37 evel() [1/2] . . . 126
11.3.3.38 evel() [2/2] . . . 126
11.3.3.39 evla() [1/2] . . . 126
11.3.3.40 evla() [2/2] . . . 127
11.3.3.41 evlo() [1/2] . . . 127
11.3.3.42 evlo() [2/2] . . . 127
11.3.3.43 f() [1/2] . . . 128
11.3.3.44 f() [2/2] . . . 128
11.3.3.45 frequency() . . . 128
11.3.3.46 gcarc() [1/2] . . . 129
11.3.3.47 gcarc() [2/2] . . . 129
11.3.3.48 geometry_set() . . . 130
11.3.3.49 ibody() [1/2] . . . 130
11.3.3.50 ibody() [2/2] . . . 131
11.3.3.51 idep() [1/2] . . . 131
11.3.3.52 idep() [2/2] . . . 131
11.3.3.53 ievreg() [1/2] . . . 132
11.3.3.54 ievreg() [2/2] . . . 132
11.3.3.55 ievtyp() [1/2] . . . 132
11.3.3.56 ievtyp() [2/2] . . . 132
11.3.3.57 iftype() [1/2] . . . 133
11.3.3.58 iftype() [2/2] . . . 133
11.3.3.59 iinst() [1/2] . . . 133
11.3.3.60 iinst() [2/2] . . . 133
11.3.3.61 imagsrc() [1/2] . . . 134
11.3.3.62 imagsrc() [2/2] . . . 134
11.3.3.63 imagtyp() [1/2] . . . 134
11.3.3.64 imagtyp() [2/2] . . . 134
11.3.3.65 iqual() [1/2] . . . 135
11.3.3.66 iqual() [2/2] . . . 135
11.3.3.67 istreg() [1/2] . . . 135
11.3.3.68 istreg() [2/2] . . . 135
11.3.3.69 isynth() [1/2] . . . 136
11.3.3.70 isynth() [2/2] . . . 136
11.3.3.71 iztype() [1/2] . . . 136
11.3.3.72 iztype() [2/2] . . . 136
11.3.3.73 ka() [1/2] . . . 137
11.3.3.74 ka() [2/2] . . . 137
11.3.3.75 kcmpnm() [1/2] . . . 137

11.3.3.76 kcmpnm() [2/2] . . . 137
11.3.3.77 kdatrd() [1/2] . . . 138
11.3.3.78 kdatrd() [2/2] . . . 138
11.3.3.79 kevnm() [1/2] . . . 138
11.3.3.80 kevnm() [2/2] . . . 138
11.3.3.81 kf() [1/2] . . . 139
11.3.3.82 kf() [2/2] . . . 139
11.3.3.83 khole() [1/2] . . . 139
11.3.3.84 khole() [2/2] . . . 139
11.3.3.85 kinst() [1/2] . . . 140
11.3.3.86 kinst() [2/2] . . . 140
11.3.3.87 knetwk() [1/2] . . . 140
11.3.3.88 knetwk() [2/2] . . . 140
11.3.3.89 ko() [1/2] . . . 141
11.3.3.90 ko() [2/2] . . . 141
11.3.3.91 kstnm() [1/2] . . . 141
11.3.3.92 kstnm() [2/2] . . . 141
11.3.3.93 kt0() [1/2] . . . 142
11.3.3.94 kt0() [2/2] . . . 142
11.3.3.95 kt1() [1/2] . . . 142
11.3.3.96 kt1() [2/2] . . . 142
11.3.3.97 kt2() [1/2] . . . 143
11.3.3.98 kt2() [2/2] . . . 143
11.3.3.99 kt3() [1/2] . . . 143
11.3.3.100 kt3() [2/2] . . . 143
11.3.3.101 kt4() [1/2] . . . 144
11.3.3.102 kt4() [2/2] . . . 144
11.3.3.103 kt5() [1/2] . . . 144
11.3.3.104 kt5() [2/2] . . . 144
11.3.3.105 kt6() [1/2] . . . 145
11.3.3.106 kt6() [2/2] . . . 145
11.3.3.107 kt7() [1/2] . . . 145
11.3.3.108 kt7() [2/2] . . . 145
11.3.3.109 kt8() [1/2] . . . 146
11.3.3.110 kt8() [2/2] . . . 146
11.3.3.111 kt9() [1/2] . . . 146
11.3.3.112 kt9() [2/2] . . . 146
11.3.3.113 kuser0() [1/2] . . . 147
11.3.3.114 kuser0() [2/2] . . . 147
11.3.3.115 kuser1() [1/2] . . . 147
11.3.3.116 kuser1() [2/2] . . . 147
11.3.3.117 kuser2() [1/2] . . . 148

11.3.3.118 kuser2() [2/2] . . . 148
11.3.3.119 lcalda() [1/2] . . . 148
11.3.3.120 lcalda() [2/2] . . . 148
11.3.3.121 legacy_write() . . . 148
11.3.3.122 leven() [1/2] . . . 150
11.3.3.123 leven() [2/2] . . . 151
11.3.3.124 lovrok() [1/2] . . . 151
11.3.3.125 lovrok() [2/2] . . . 151
11.3.3.126 lpspol() [1/2] . . . 152
11.3.3.127 lpspol() [2/2] . . . 152
11.3.3.128 mag() [1/2] . . . 152
11.3.3.129 mag() [2/2] . . . 152
11.3.3.130 nevid() [1/2] . . . 153
11.3.3.131 nevid() [2/2] . . . 153
11.3.3.132 norid() [1/2] . . . 153
11.3.3.133 norid() [2/2] . . . 153
11.3.3.134 npts() [1/2] . . . 154
11.3.3.135 npts() [2/2] . . . 154
11.3.3.136 nsnpts() [1/2] . . . 154
11.3.3.137 nsnpts() [2/2] . . . 154
11.3.3.138 nvhdr() [1/2] . . . 155
11.3.3.139 nvhdr() [2/2] . . . 155
11.3.3.140 nwfid() [1/2] . . . 155
11.3.3.141 nwfid() [2/2] . . . 155
11.3.3.142 nxsize() [1/2] . . . 156
11.3.3.143 nxsize() [2/2] . . . 156
11.3.3.144 nysize() [1/2] . . . 156
11.3.3.145 nysize() [2/2] . . . 156
11.3.3.146 nzhour() [1/2] . . . 157
11.3.3.147 nzhour() [2/2] . . . 157
11.3.3.148 nzjday() [1/2] . . . 157
11.3.3.149 nzjday() [2/2] . . . 157
11.3.3.150 nzmin() [1/2] . . . 158
11.3.3.151 nzmin() [2/2] . . . 158
11.3.3.152 nzmsec() [1/2] . . . 158
11.3.3.153 nzmsec() [2/2] . . . 158
11.3.3.154 nzsec() [1/2] . . . 159
11.3.3.155 nzsec() [2/2] . . . 159
11.3.3.156 nzyear() [1/2] . . . 159
11.3.3.157 nzyear() [2/2] . . . 159
11.3.3.158 o() [1/2] . . . 160
11.3.3.159 o() [2/2] . . . 160

11.3.3.160 odelta() [1/2] . . . 160
11.3.3.161 odelta() [2/2] . . . 160
11.3.3.162 operator==() . . . 160
11.3.3.163 resize_data() . . . 161
11.3.3.164 resize_data1() . . . 162
11.3.3.165 resize_data2() . . . 162
11.3.3.166 resp0() [1/2] . . . 162
11.3.3.167 resp0() [2/2] . . . 162
11.3.3.168 resp1() [1/2] . . . 163
11.3.3.169 resp1() [2/2] . . . 163
11.3.3.170 resp2() [1/2] . . . 163
11.3.3.171 resp2() [2/2] . . . 163
11.3.3.172 resp3() [1/2] . . . 164
11.3.3.173 resp3() [2/2] . . . 164
11.3.3.174 resp4() [1/2] . . . 164
11.3.3.175 resp4() [2/2] . . . 164
11.3.3.176 resp5() [1/2] . . . 165
11.3.3.177 resp5() [2/2] . . . 165
11.3.3.178 resp6() [1/2] . . . 165
11.3.3.179 resp6() [2/2] . . . 165
11.3.3.180 resp7() [1/2] . . . 166
11.3.3.181 resp7() [2/2] . . . 166
11.3.3.182 resp8() [1/2] . . . 166
11.3.3.183 resp8() [2/2] . . . 166
11.3.3.184 resp9() [1/2] . . . 167
11.3.3.185 resp9() [2/2] . . . 167
11.3.3.186 sb() [1/2] . . . 167
11.3.3.187 sb() [2/2] . . . 167
11.3.3.188 sdelta() [1/2] . . . 168
11.3.3.189 sdelta() [2/2] . . . 168
11.3.3.190 stdp() [1/2] . . . 168
11.3.3.191 stdp() [2/2] . . . 168
11.3.3.192 stel() [1/2] . . . 169
11.3.3.193 stel() [2/2] . . . 169
11.3.3.194 stla() [1/2] . . . 169
11.3.3.195 stla() [2/2] . . . 170
11.3.3.196 stlo() [1/2] . . . 170
11.3.3.197 stlo() [2/2] . . . 170
11.3.3.198 t0() [1/2] . . . 171
11.3.3.199 t0() [2/2] . . . 171
11.3.3.200 t1() [1/2] . . . 171
11.3.3.201 t1() [2/2] . . . 172

11.3.3.202 t2() [1/2] . . . 172
11.3.3.203 t2() [2/2] . . . 172
11.3.3.204 t3() [1/2] . . . 172
11.3.3.205 t3() [2/2] . . . 173
11.3.3.206 t4() [1/2] . . . 173
11.3.3.207 t4() [2/2] . . . 173
11.3.3.208 t5() [1/2] . . . 173
11.3.3.209 t5() [2/2] . . . 174
11.3.3.210 t6() [1/2] . . . 174
11.3.3.211 t6() [2/2] . . . 174
11.3.3.212 t7() [1/2] . . . 174
11.3.3.213 t7() [2/2] . . . 175
11.3.3.214 t8() [1/2] . . . 175
11.3.3.215 t8() [2/2] . . . 175
11.3.3.216 t9() [1/2] . . . 175
11.3.3.217 t9() [2/2] . . . 176
11.3.3.218 time() . . . 176
11.3.3.219 user0() [1/2] . . . 177
11.3.3.220 user0() [2/2] . . . 177
11.3.3.221 user1() [1/2] . . . 177
11.3.3.222 user1() [2/2] . . . 177
11.3.3.223 user2() [1/2] . . . 178
11.3.3.224 user2() [2/2] . . . 178
11.3.3.225 user3() [1/2] . . . 178
11.3.3.226 user3() [2/2] . . . 178
11.3.3.227 user4() [1/2] . . . 179
11.3.3.228 user4() [2/2] . . . 179
11.3.3.229 user5() [1/2] . . . 179
11.3.3.230 user5() [2/2] . . . 179
11.3.3.231 user6() [1/2] . . . 180
11.3.3.232 user6() [2/2] . . . 180
11.3.3.233 user7() [1/2] . . . 180
11.3.3.234 user7() [2/2] . . . 180
11.3.3.235 user8() [1/2] . . . 181
11.3.3.236 user8() [2/2] . . . 181
11.3.3.237 user9() [1/2] . . . 181
11.3.3.238 user9() [2/2] . . . 181
11.3.3.239 write() . . . 181
11.3.3.240 xmaximum() [1/2] . . . 185
11.3.3.241 xmaximum() [2/2] . . . 185
11.3.3.242 xminimum() [1/2] . . . 185
11.3.3.243 xminimum() [2/2] . . . 185

11.3.3.244 ymaximum() [1/2] . . . 186
11.3.3.245 ymaximum() [2/2] . . . 186
11.3.3.246 yminimum() [1/2] . . . 186
11.3.3.247 yminimum() [2/2] . . . 186
11.3.4 Member Data Documentation . . . 187
11.3.4.1 bools . . . 187
11.3.4.2 data . . . 187
11.3.4.3 doubles . . . 187
11.3.4.4 floats . . . 187
11.3.4.5 ints . . . 187
11.3.4.6 strings . . . 187
11.4 sacfmt::bitset_type::uint< nbits > Struct Template Reference . . . 188
11.4.1 Detailed Description . . . 188
11.5 sacfmt::bitset_type::uint< 4 ∗bits_per_byte > Struct Reference . . . 188
11.5.1 Detailed Description . . . 188
11.5.2 Member Typedef Documentation . . . 188
11.5.2.1 type . . . 188
11.6 sacfmt::bitset_type::uint< bytes ∗bits_per_byte > Struct Reference . . . 189
11.6.1 Detailed Description . . . 189
11.6.2 Member Typedef Documentation . . . 189
11.6.2.1 type . . . 189
11.7 sacfmt::word_pair< T > Struct Template Reference . . . 189
11.7.1 Detailed Description . . . 189
11.7.2 Member Data Documentation . . . 190
11.7.2.1 first . . . 190
11.7.2.2 second . . . 190

Index **191**

Chapter 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 `sac-format` from the GitHub release page.

`Download` an offline version of the documentation (PDF).

Get `help` from the community forum.

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 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 are `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.

1.1.2 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.

1.1.3 Easy

sac-format is **easy** single-header makes integration in any project simple. Installation is easy with our automatic installers. 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.

1.1.4 Small

sac-format is **small** in total (header + implementation; excluding comments) the library is under 2100∗ 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.

∗ This value includes only the library, excluding all testing/benchmarking and example codes. Including `utests.cpp`, `benchmark.cpp`, `util.hpp`, the example program (`list_sac`), and sac-format totals just over 5100 lines of code.

1.1.5 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.

1.1.6 Transparent

sac-format is **transparent** all analysis and coverage information is publicly available online.

- `CodeFactor`
- `Codacy`
- `CodeCov`
- `Coverity Scan`

1.1.7 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.

1.1.8 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!

Chapter 2

Installation

This section provides installation instructions.

The easiest way to use sac-format is to install it via the automatic installers. Installers for the latest release are located here. Be sure to check the sha512 checksum of the installer against its correspondingly named `.sha512` file to ensure the file is safe (for example: `sac-format.pkg` corresponds to `sac-format.pkg.sha512`).

2.1 Windows

sac-format provides a graphical installer on Windows (`sac-format.exe`).

Always check the sha512 checksum value of the installer (`sac-format.exe`; more info here) against `sac-format.exe.sha512`.

By default, Microsoft Defender will block the installer with a pop-up like that one below:

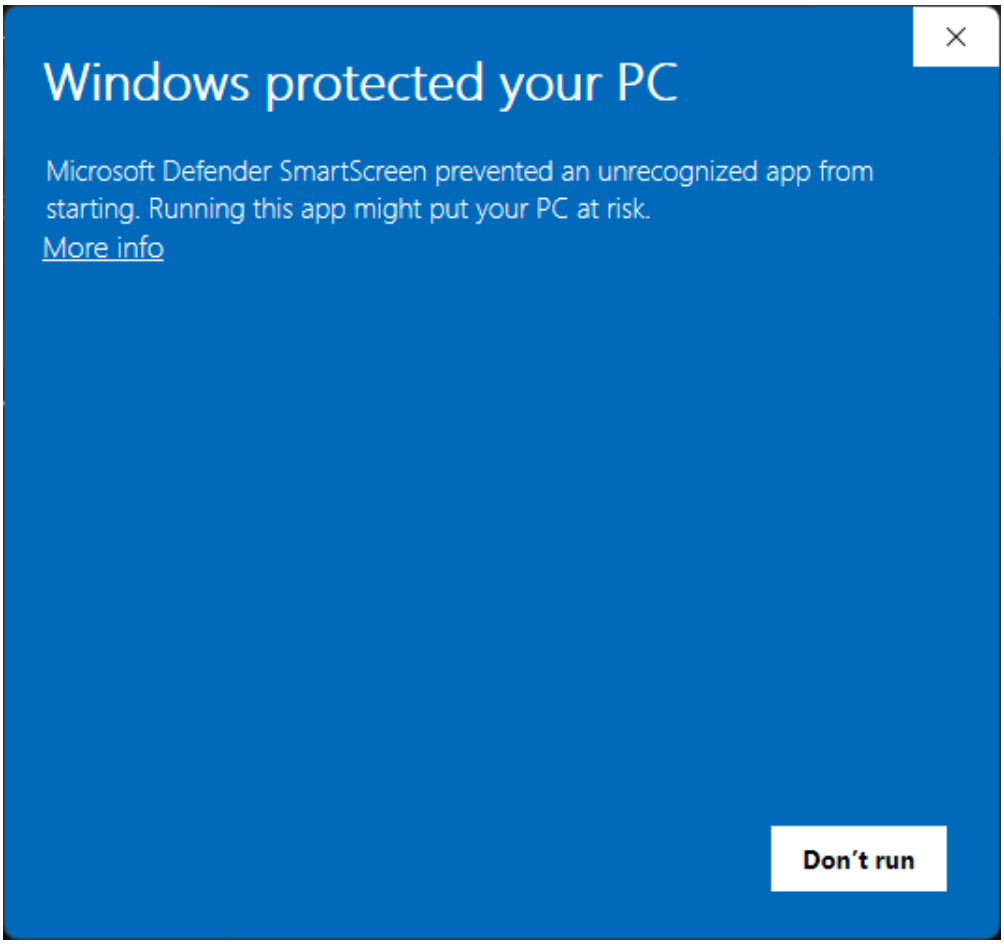


Figure 2.1 Windows Warning 1

To continue the install, click on the "More Info" link and then the "Run anyway" button as seen in the following image:

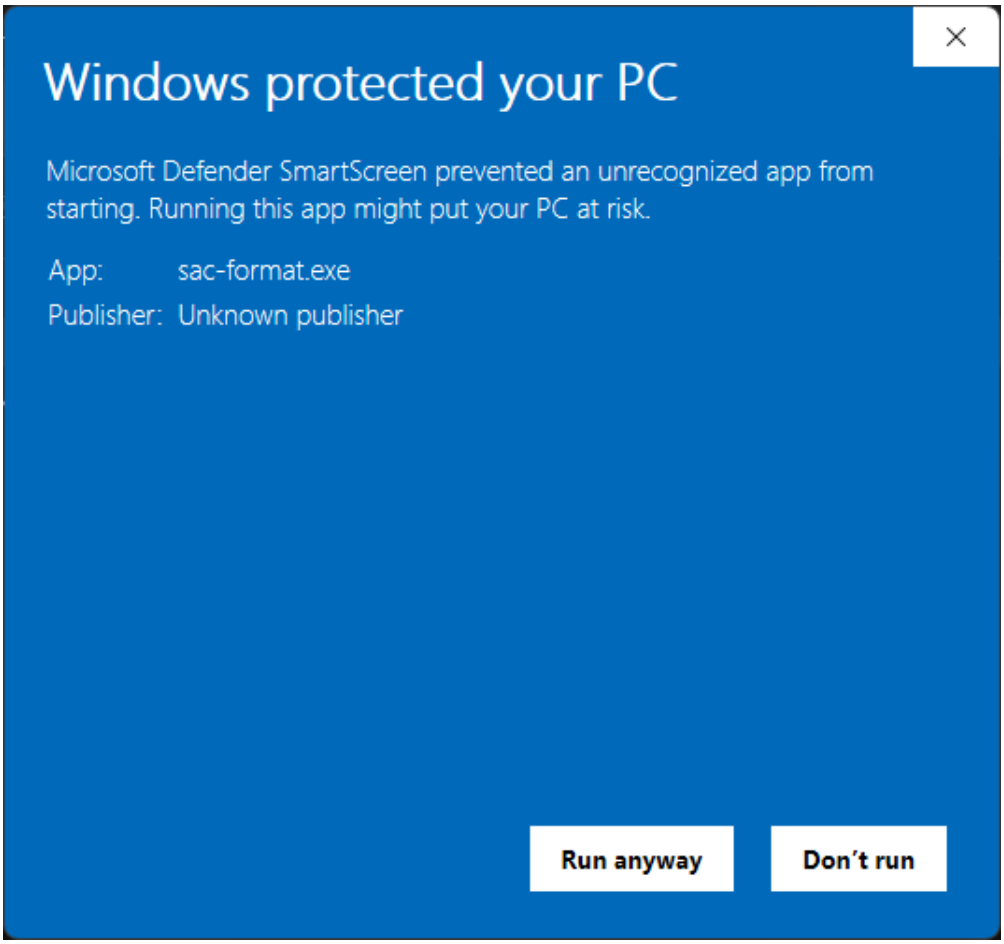


Figure 2.2 Windows Warning 2

Then the installer will open and present you with the welcome screen:

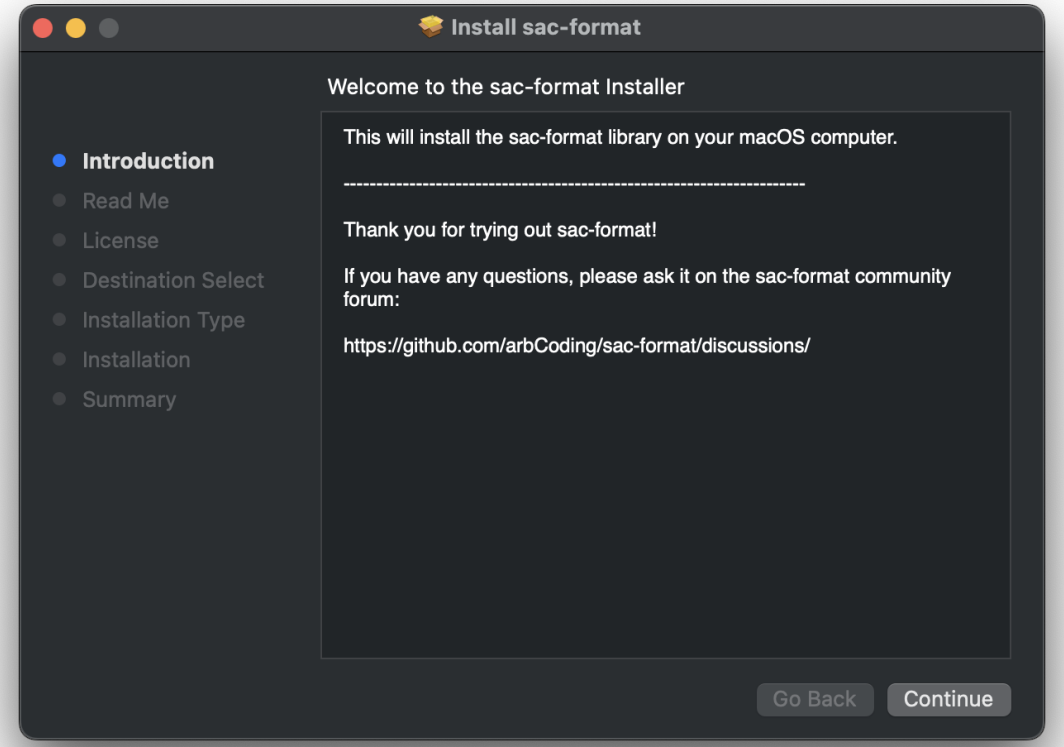


Figure 2.3 Windows Intro Install

By default, sac-format installs in `C:/Program Files/sac-format` as seen in the screen below:

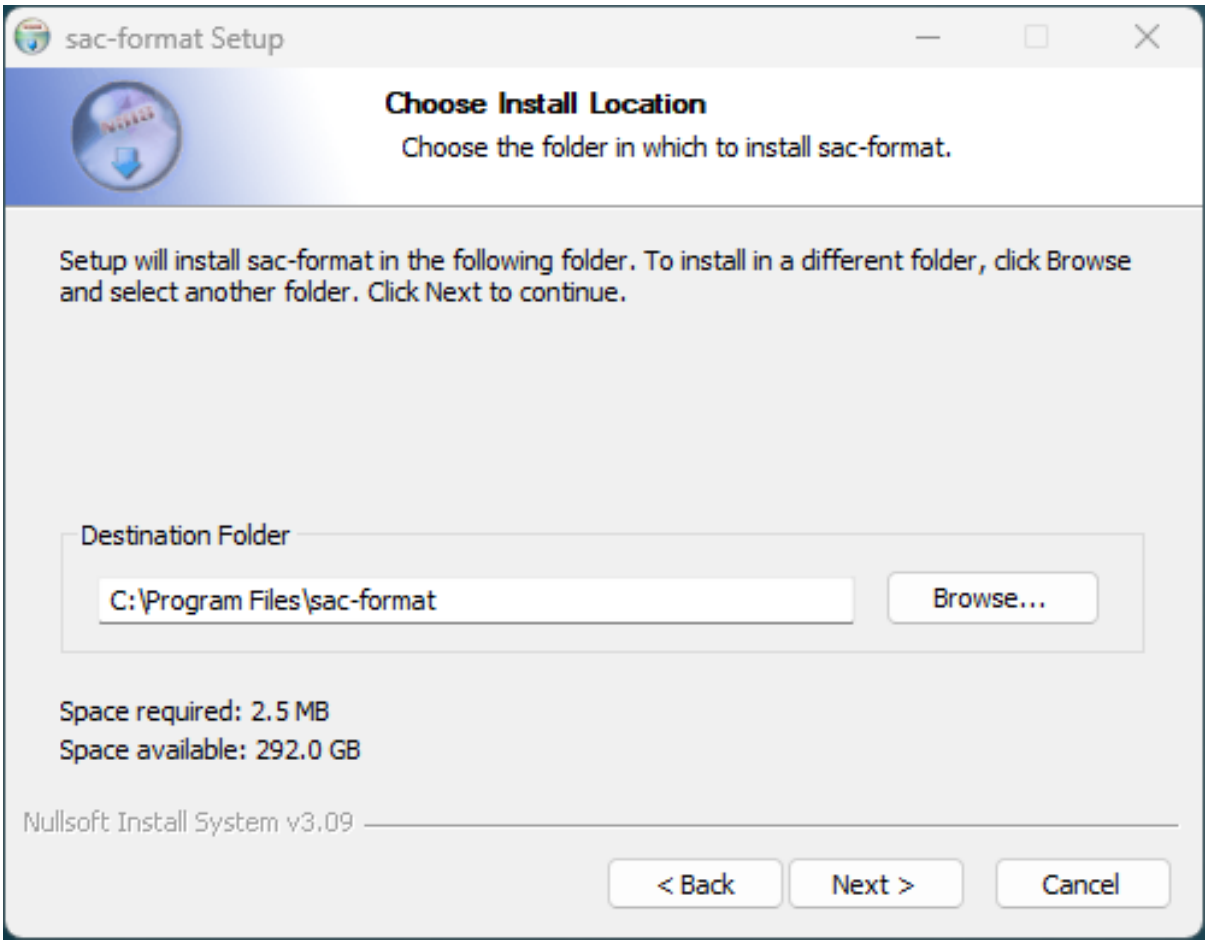


Figure 2.4 Windows Location Install

Because all programs in sac-format are command-line based feel free to disable Start Menu shortcuts:

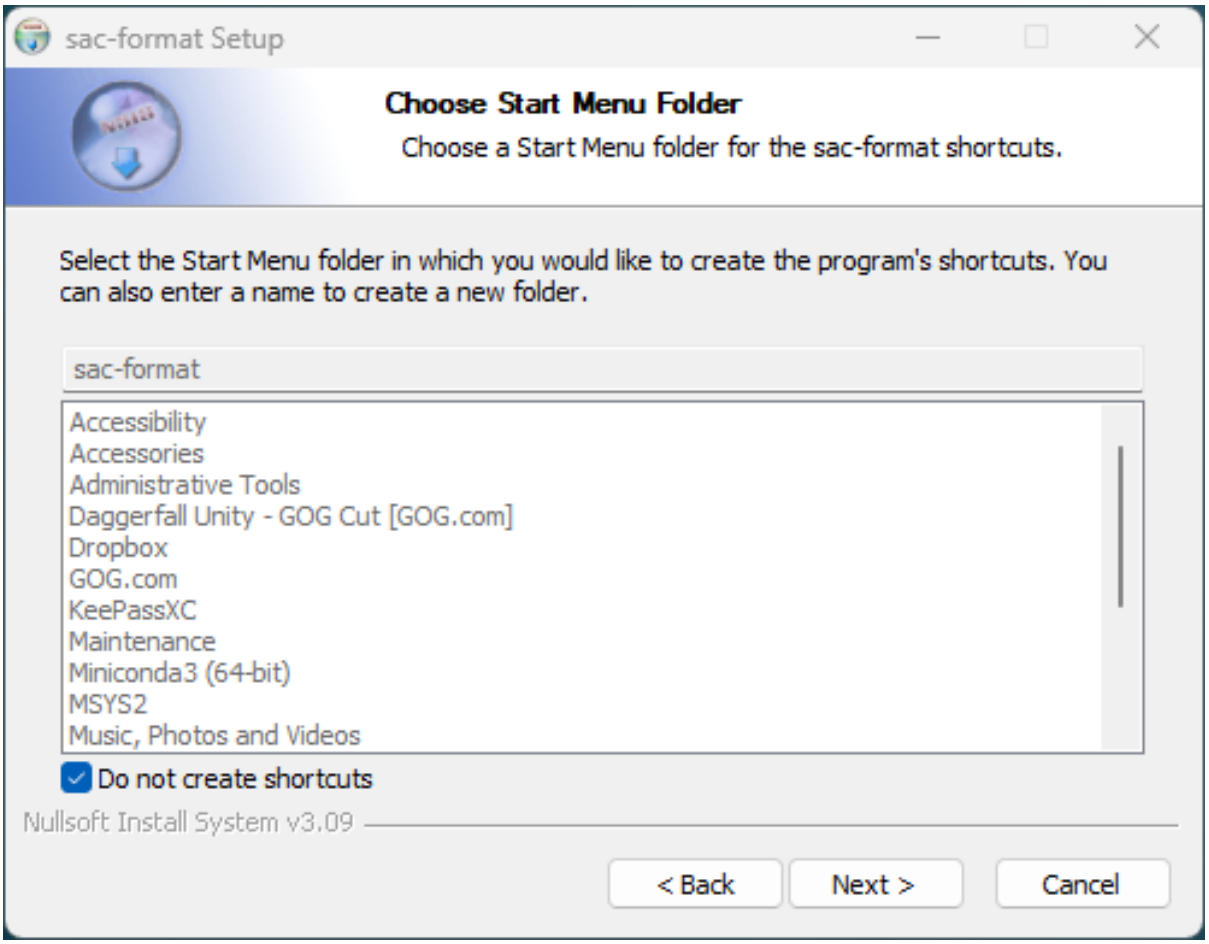


Figure 2.5 Windows No Shortcuts

Upon successful install of sac-format you will see this window:

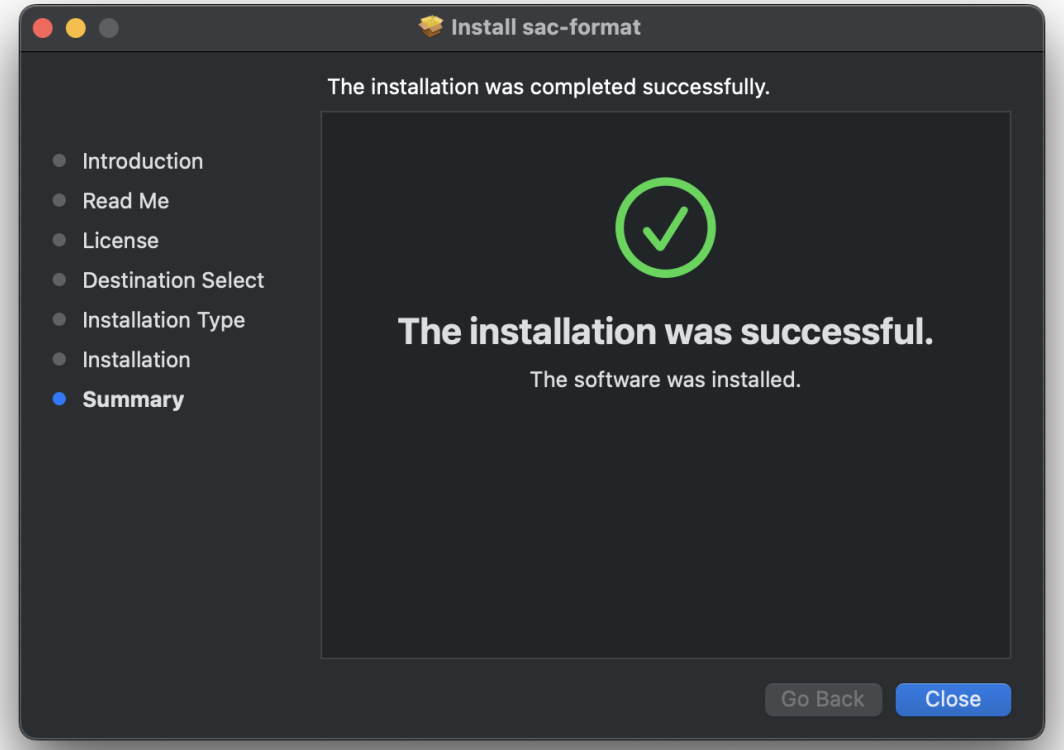


Figure 2.6 Windows Install Success

2.2 macOS

sac-format provides both command line and graphical installers on macOS.

2.2.1 Graphical

The graphical installer is `sac-format.pkg` and will walk you through the installation process. **NOTE**: the default installation location is `/opt/sac-format`.

By default, macOS will block the installer. To install, right-click on `sac-format.pkg` and select open. A warning will pop up that looks like:

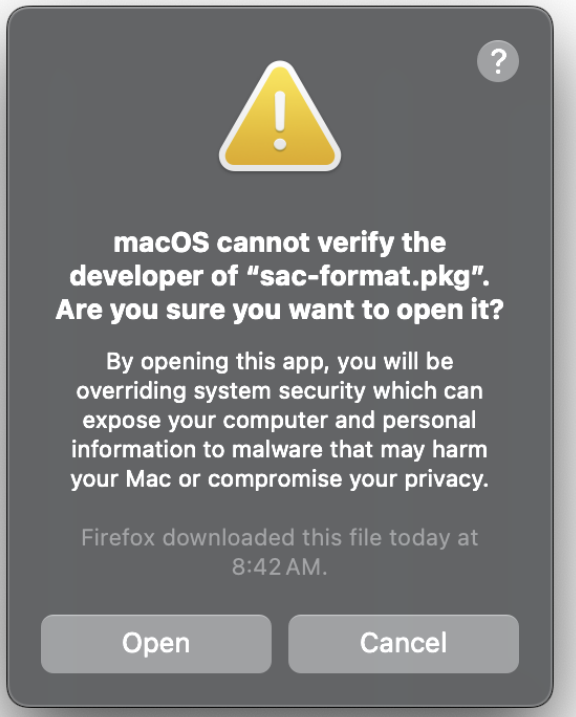


Figure 2.7 macOS Warning

Simply click "Open" and the installer will begin from the first screen:

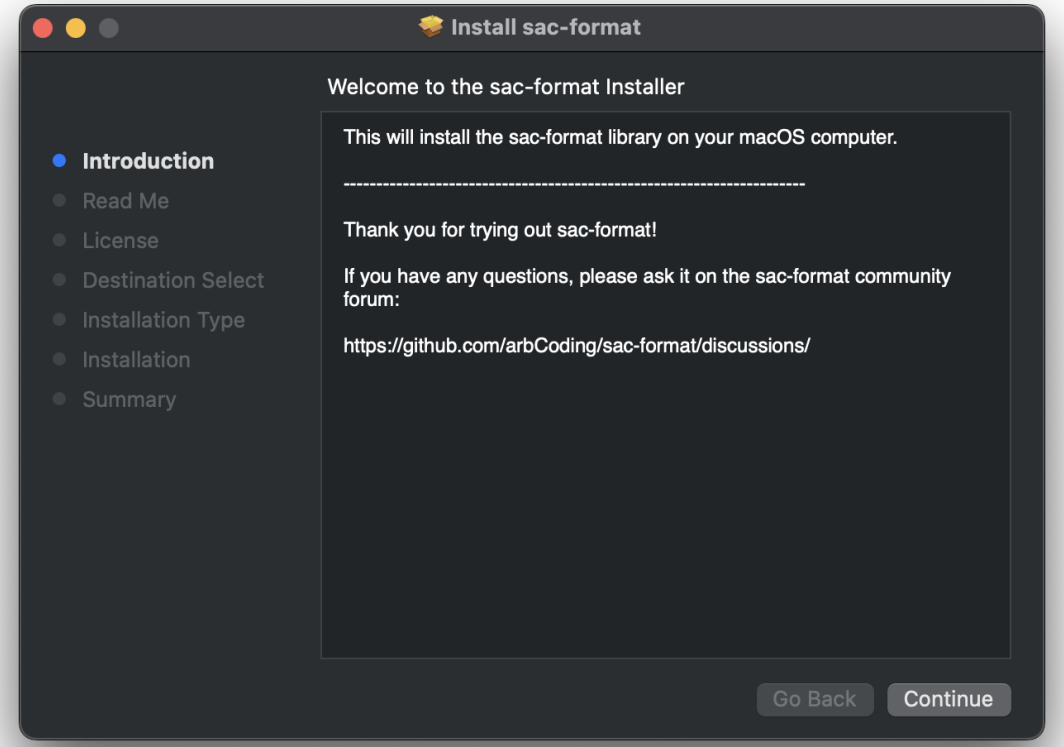


Figure 2.8 macOS Intro Install

Upon successful installation you will see:

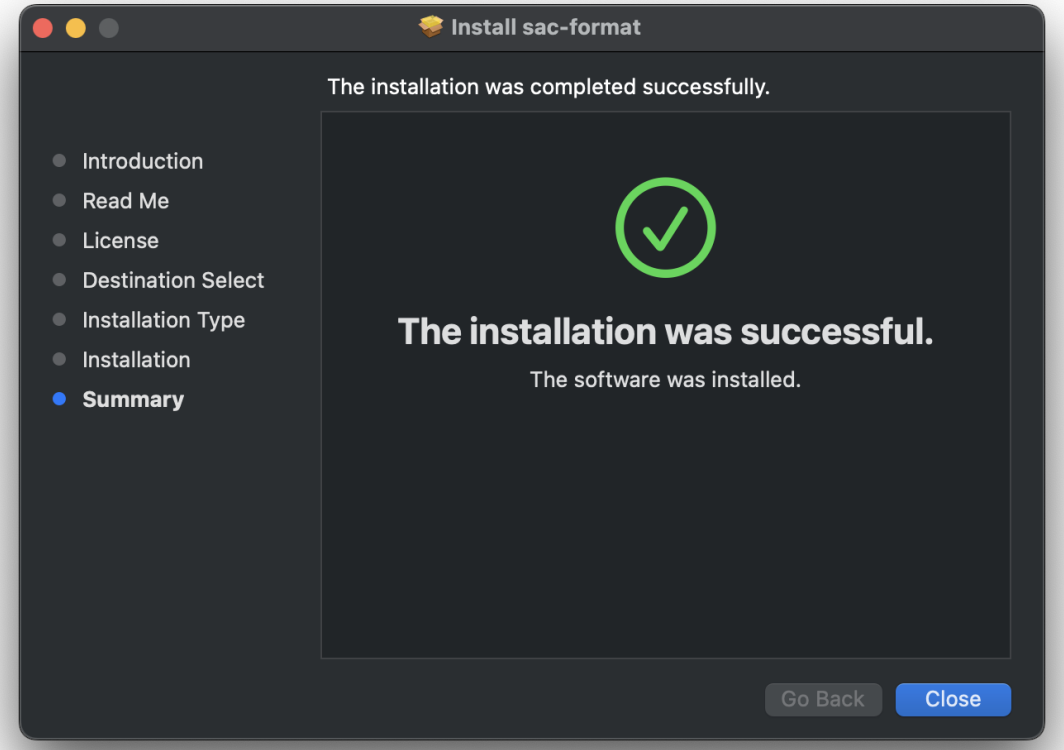


Figure 2.9 macOS Install Success

2.2.2 Command line

Command line installation is performed either using the self-extrating archive or by manually extracting the gzipped tar archive.

2.2.2.1 Self-Extracting Archive

```
# Check the sha512 checksum
sha512sum -c sac-format-<version>-Darwin-<arch>.sh.sha512
# Run self-extracting archive
bash sac-format-<version>-Darwin-<arch>.sh
```

Be sure to replace `<version>` and `<arch>` with the correct versions and architectures, respectively (for example: `sac-format-0.4.0-Darwin-x86_64.sh`).

2.2.2.2 Gzipped Tar Archive

```
# Check the sha512 checksum
sha512sum -c sac-format-<version>-Darwin-<arch>.tar.gz.sha512
# Extract Gzipped tar archive
tar -xzf sac-format-<version>-Darwin-<arch>.tar.gz
```

2.3 Linux

sac-format provides four different command line installation methods on Linux.

`Debian` based distributions (for example: Debian, Ubuntu, Linux Mint) can use the Debian Archive.

`RedHat` based distributions (for example: RedHat, Fedora, CentOS) can use the RPM Archive.

All distributions can use the Self-Extracting Archive.

All distributions can use the Gzipped Tar Archive.

2.3.1 Debian Archive

```
# Check the sha512 checksum
sha512sum -c sac-format.deb.sha512
# Install using apt
sudo apt install ./sac-format.deb
```

2.3.2 RPM Archive

```
# Check the sha512 checksum
sha512sum -c sac-format.rpm.sha512
# Install using rpm
sudo rpm -i sac-format.rpm
```

2.3.3 Self-Extrating Archive

```
# Check the sha512 checksum
sha512sum -c sac-format-<version>-Linux-<arch>.sh.sha512
# Run self-extrating archive
bash sac-format-<version>-Linux-<arch>.sh
```

2.3.4 Gzipped Tar Archive

```
# Check the sha512 checksum
sha512sum -c sac-format-<version>-Linux-<arch>.tar.gz.sha512
# Extract gzipped tar archive
tar -xzf sac-format-<version>-Linux-<arch>.tar.gz
```

Chapter 3

Quickstart

This section provides information to incorporate into a project.

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

3.1 Example Programs

3.1.1 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.

3.2 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)
```

3.3 Example

3.3.1 Reading and Writing

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

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{file};
    // Confirm equality
    std::cout « (trace1 == trace2) « '\n';
    fs::remove(file);
    return EXIT_SUCCESS;
}
```

Chapter 4

Basic Documentation

This section provides a brief overview of functionality and usage.

4.1 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).

4.1.1 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:

```
#include <sac_foramt.hpp>
#include <filesystem>

int main() {
  std::filesystem::path my_file{"/home/user/data/ANMO.SAC"};
  sacfmt::Trace anmo{my_file};
  return EXIT_SUCCESS;
}
```

4.1.2 Writing SAC

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

4.1.2.1 v7 files

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

4.1.2.2 v6 files

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

4.1.3 Getters and Setters

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

4.1.3.1 Example Getters

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

4.1.3.2 Example Setters

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

4.1.3.3 Setter rules

Most of the setters are only constrained by the parameter type (single-precision, double-precision, boolean, etc.). **Some** setters are constrained by additional rules.

Required for sanity

Rules here are required because the sac-format library assumes them (not strictly required by the SAC format standard). For instance, the geometric functions assume certain bounds on latitudes and longitudes.

sac-format automatically imposes these rules.

`stla(input)`

Limited to [-90, 90] degrees, input that is outside that range is reduced using circular symmetry.

`stlo(input)`

Limited to [-180, 180] degrees, input that is outside that range is reduced using circular symmetry.

`evla(input)`

Limited to [-90, 90] degrees, input that is outside that range is reduced using circular symmetry.

evlo(input)

Limited to [-180, 180] degrees, input that is outside that range is reduced using circular symmetry.

Required for safety

Rules here are required by the SAC format standard. sac-format automatically imposes these rules to prevent the creation of corrupt sac-files.

npts(input)

Because `npts` defines the size of the data vectors, changing this value will change the size of `data1` and `data2`∗. Increasing npts resizes the vectors (`std::vector::resize`) by placing zeros at the **end** of the vectors. Reducing npts resizes the vectors down to the **first npts** values.

Therefore, care must be taken to maintain separate copies of `data1` and `data2`∗ if you plan to manipulate the original data **after** resizing.

∗ data2 has `npts` only if it is legal, otherwise it is of size 0.

leven(input)

Changing the value of `leven` potentially changes the legality of `data2`, it also potentially affects the value of `iftype`.

If iftype>1, then leven must be `true` (evenly sampled data). Therefore, if leven is made `false` in this scenario (unevenly sampled data) then iftype becomes unset∗.

If changing leven makes data2 legal∗∗, then data2 is qresized to have `npts` zeros.

∗ The SAC format defines the unset values for all data-types. For integers (like iftype) it is the integer value `-12345`.

∗∗ If data2 was already legal, then it is unaffected.

iftype(input)

Changing the value of `iftype` poentially changes the legality of `data2`, it also potentially affects the value of `leven`.

If leven is `false`, then iftype must be either 1 or unset. Therefore, changing iftype to have a value >1 requires that leven becomes `true` (evenly sampled data).

If changing iftype makes data2 legal∗, then data2 is resized to have `npts` zeros.

∗ If data2 was already legal, then it is unaffected.

data1(input)

If the size of `data1` is changed, then `npts` must change to reflect the new size. If `data2` is legal, this adjusts its size to match as well.

data2(input)

If the size of `data2` is changed to be larger than 0 and it is illegal, it is made legal by setting `iftype(2)` (spectral-data).

When the size of data2 changes, `npts` is updated to the new size and `data1` is resized to match.

If `data2` is made illegal, its size is reduced to 0 while `npts` and `data1` are unaffected.

4.1.4 Convenience Methods

4.1.4.1 calc_geometry

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

```
trace.stla(45.3);
trace.stlo(34.5);
trace.evla(18.5);
trace.evlo(-34);
trace.calc_geometry();
std::cout « "GcArc: " « trace.gcarc() « '\n';
std::cout « "Dist: " « trace.dist() « '\n';
std::cout « "Azimuth: " « trace.az() « '\n';
std::cout « "BAzimuth: " « trace.baz() « '\n';
```

4.1.4.2 frequency

Calculate frequency from `delta`.

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

4.1.4.3 date

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

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

4.1.4.4 time

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

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

4.1.5 Exceptions

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

4.2 Convenience Functions

4.2.1 degrees_to_radians

Convert decimal degrees to radians.

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

4.2.2 radians_to_degrees

Convert radians to decimal degrees.

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

4.2.3 gcarc

Calculate great-circle arc distance (spherical planet).

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

4.2.4 azimuth

Calculate azimuth between two points (spherical planet).

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

4.2.5 limit_360

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

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

4.2.6 limit_180

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

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

4.2.7 limit_90

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

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

4.3 Low-Level I/O

Low-level I/O functions are discussed below.

4.3.1 Binary conversion

4.3.1.1 int_to_binary and binary_to_int

Conversion pair for binary representation of integer values.

```
const int input{10};
// sacfmt::word_one is alias for std::bitset<32> (one word)
sacfmt::word_one binary{sacfmt::int_to_binary(input)};
const int output{sacfmt::binary_to_int(binary)};
std::cout « (input == output) « '\n';
```

4.3.1.2 float_to_binary and binary_to_float

Conversion pair for binary representation of floating-point values.

```
const float input{5F};
sacfmt::word_one binary{sacfmt::float_to_binary(input)};
const float output{sacfmt::binary_to_float(binary)};
std::cout « (input == output) « '\n';
```

4.3.1.3 double_to_binary and binary_to_double

Conversion pair for binary representation of double-precision values.

```
const double input{1e5};
// sacfmt::word_two is alias for std::bitset<64> (two words)
sacfmt::word_two binary{sacfmt::double_to_binary(input)};
const double output{sacfmt::binary_to_double(binary)};
std::cout « (input == output) « '\n';
```

4.3.1.4 string_to_binary and binary_to_string

Conversion pair for binary representation of two-word (regular) string values.

```
const std::string input{"NmlStrng"};
sacfmt::word_two binary{sacfmt::string_to_binary(input)};
const std::string output{sacfmt::binary_to_string(binary)};
std::cout « (input == output) « '\n';
```

4.3.1.5 long_string_to_binary and binary_to_long_string

Conversion pair for binary representation of four-word (only `kstnm` string values.

```
const std::string input{"The Long String"};
// sacfmt::word_four is alias for std::bitset<128> (four words)
sacfmt::word_four binary{sacfmt::long_string_to_binary(input)};
const std::string output{sacfmt::binary_to_long_string(binary)};
std::cout « (input == output) « '\n';
```

4.3.2 Reading/Writing

NOTE that care must be taken when using them to ensure that safe input is provided; the `Trace` class ensures safe I/O, low-level I/O functions do not necessarily ensure safety.

4.3.2.1 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`).

4.3.2.2 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).

4.3.2.3 write_words

Writes input words (as `std::vector<char>`) to a binary SAC-file.

4.3.3 Utility

4.3.3.1 concat_words

Concatenates words taking into account the system endianness.

4.3.3.2 bits_string and string_bits

Template function that performs conversion of binary strings of arbitrary length to an arbitrary number of words.

4.3.3.3 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).

4.3.3.4 string_cleaning

Ensures string does not contain an internal termination character (`\0`) and removes it if present, then removes blank spaces.

4.3.3.5 prep_string

Performs `string_cleaning` followed by string truncation/padding to the necessary length.

4.3.3.6 equal_within_tolerance

Floating-point/double-precision equality within a provided tolerance (default is `f_eps`, defined in `sac_format.hpp`).

4.4 Testing

Unit- and integration-tests (using Catch2) are contained in the `tests` folder. They include:

- `binary_conversions.cpp` confirms that conversion to/from binary functions correctly.
- `constants.cpp` confirms constant values (e.g. SAC magic numbers) are correct.
- `datetime.cpp` confirms date and time functions work correctly.
- `geometry.cpp` confirms that geometric calculations are correct (azimuth, greater-circle arc-length, etc.).
- `trace.cpp` confirms that the trace class is functioning correctly (I/O, exceptions, bounded headers, etc.).

The tests compile to the following programs:

- `basic_tests` (binary conversions and constants).
- `datetime_tests`
- `geometry_tests`
- `trace_tests`

Test coverage details are visible on `CodeCov.io` and `Codacy.com`. All tests can be locally-run to ensure full functionality and compliance.

4.4.1 Errors only

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

4.4.2 Full output

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

4.4.3 Compact output

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

4.4.4 Additional options

To see additional options, run `-?`.

4.4.5 Using ctest

If you have CMake install, you can run the tests using `ctest`.

4.5 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 `becnhmark -?`.

4.6 Source File List

4.6.1 Core

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

4.6.1.1 sac_format.hpp

Interface: function declarations and constants.

4.6.1.2 sac_format.cpp

Implementation: function details.

4.6.2 Testing and Benchmarking

4.6.2.1 util.hpp

Utility functions and constants exclusive to testing and benchmarking. Not split into interface/implementation.

4.6.2.2 utests.cpp

4.6.2.3 benchmark.cpp

4.6.3 Example programs

4.6.3.1 list_sac.cpp

Chapter 5

SAC-file format

This section provides a centralized description of the 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).

5.1 Floating-point (39)

32-bit (1 word, 4 bytes)

5.1.1 depmin

Minimum value of the dependent variable (displacement/velocity/acceleration/volts/counts).

5.1.2 depmen

Mean value of the dependent variable.

5.1.3 depmax

Maximum value of the dependent variable.

5.1.4 odelta

Modified (*observational*) value of `delta`.

5.1.5 resp(0–9)

Instrument response parameters (poles, zeros, and a constant).

Not used by SAC they're free for other purposes.

5.1.6 stel

Station elevation in meters above sea level (*m.a.s.l*).

Not used by SAC free for other purposes.

5.1.7 stdp

Station depth in meters below surface (borehole/buried vault).

Not used by SAC free for other purposes.

5.1.8 evel

Event elevation *m.a.s.l.*

Not used by SAC free for other purposes.

5.1.9 evdp

Event depth in kilometers (*previously meters*) below surface.

5.1.10 mag

Event magnitude.

5.1.11 user(0–9)

Storage for user-defined values.

5.1.12 dist

Station-Event distance in kilometers.

5.1.13 az

Azimuth $(\text{Event} \rightarrow \text{Station})$, decimal degrees from North.

5.1.14 baz

Back-azimuth $(\text{Station} \rightarrow \text{Event})$, decimal degrees from North.

5.1.15 gcarc

Station-Event great circle arc-length, decimal degrees.

5.1.16 cmpaz

Instrument measurement azimuth, decimal degrees from North.

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

5.1.17 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°).

5.1.18 xminimum

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

5.1.19 xmaximum

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

5.1.20 yminimum

Spectral-only equivalent of `b`.

5.1.21 ymaximum

Spectral-only equivalent of `e`.

5.2 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.

5.2.1 delta

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

5.2.2 b

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

5.2.3 e

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

5.2.4 o

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

5.2.5 a

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

5.2.6 t(0–9)

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

5.2.7 f

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

5.2.8 stla

Station latitude in decimal degrees, N/S - positive/negative.

sac-format automatically enforces $\mathrm{stla} \in [-90, 90]$.

5.2.9 stlo

Station longitude in decimal degrees, E/W - positive/negative.

sac-format automatically enforces $\mathrm{stlo} \in [-180, 180]$.

5.2.10 evla

Event latitude in decimal degrees, N/S - positive/negative.

sac-format automatically enforces $\mathrm{evla} \in [-90, 90]$.

5.2.11 evlo

Event longitude in decimal degrees, E/W - positive/negative.

sac-format automatically enforces $\mathrm{evlo} \in [-180, 180]$.

5.2.12 sb

Original (*saved*) `b` value.

5.2.13 sdelta

Original (*saved*) `delta` value.

5.3 Integer (26)

32-bit (1 word, 4 bytes)

5.3.1 nzyear

Reference time GMT year.

5.3.2 nzjday

Reference time GMT day-of-year (often called `Julian Date`) (1–366).

5.3.3 nzhour

Reference time GMT hour (0–23).

5.3.4 nzmin

Reference time GMT minute (0–59).

5.3.5 nzsec

Reference time GMT second (0–59).

5.3.6 nzmsec

Reference time GMT Millisecond (0–999).

5.3.7 nvhdr

SAC-file version.

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

5.3.8 norid

Origin ID.

5.3.9 nevid

Event ID.

5.3.10 npts

Number of points in data.

5.3.11 nsnpts

Original (*saved*) `npts`.

5.3.12 nwfid

Waveform ID.

5.3.13 nxsize

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

5.3.14 nysize

Spectral-only, width of spectrum.

5.3.15 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.

5.3.16 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)

5.3.17 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

5.3.18 iinst

Recording instrument type.

Not used by SAC: free for other purposes.

5.3.19 istreg

Station geographic region.

Not used by SAC: free for other purposes.

5.3.20 ievreg

Event geographic region.

Not used by SAC: free for other purposes.

5.3.21 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

Value	Type	Description
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

5.3.22 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.

5.3.23 isynth

Synthetic data flag.

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

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

5.3.24 imagtyp

Magnitude type.

Value	Type	Description
52	IMB	Body-wave magnitude (M_b)

Value	Type	Description
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)

5.3.25 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

5.3.26 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

5.4 Boolean (4)

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

5.4.1 leven

REQUIRED Evenly-spaced data flag.

If true, then data is evenly spaced.

5.4.2 lpspol

Station polarity flag.

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

5.4.3 lovrok

File overwrite flag.

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

5.4.4 lcalda

Calculate geometry flag.

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

5.5 String (23)

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

5.5.1 kstnm

Station name.

5.5.2 kevnm

Event name.

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

5.5.3 khole

Nuclear: Hole identifier.

Other: Location identifier (LOCID).

5.5.4 ko

Text for `o`.

5.5.5 ka

Text for `a`.

5.5.6 kt(0–9)

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

5.5.7 kf

Text for `f`.

5.5.8 kuser(0–2)

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

5.5.9 kdatrd

Date the data was read onto a computer.

5.5.10 kinst

Text for `iinst`.

5.6 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.

5.6.1 data1

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

5.6.2 data2

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

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

Chapter 6

Build Instructions

This section provides instructions to build from source.

6.1 Dependencies

6.1.1 Automatic (CMake)

`Xoshiro-cpp v1.12.0` (testing and benchmarking).

6.1.2 Manual

`Catch2 v3.4.0` (testing and benchmarking). Note that this is automatic on Windows (not Linux nor macOS).

6.1.2.1 macOS and Linux

```
git clone https://github.com/catchorg/Catch2.git
cd Catch2
git checkout v3.5.2
cmake -Bbuild -S. -DBUILD_TESTING=OFF
sudo cmake --build ./build/ --target install
```

6.2 Building

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

6.2.1 GCC

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

6.2.2 Clang

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

6.2.3 MSVC

```
git clone https://github.com/arbCoding/sac-format.git
cd sac-format
cmake -B ./build -DCMAKE_BUILD_TYPE=Release -DCMAKE_CXX_STANDARD=20 `
-DCMAKE_CXX_STANDARD_REQUIRED=ON -DCMAKE_CXX_EXTENSIONS=OFF `
-DCMAKE_CXX_FLAGS="/O2 /EHsc /Gs /guard:cf"
```

Chapter 7

Namespace Index

7.1 Namespace List

Here is a list of all namespaces with brief descriptions:

sacfmt
Sac-format namespace 47
sacfmt::bitset_type
Bitset type-safety namespace 99

Chapter 8

Hierarchical Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

std::exception
    sacfmt::io_error 101
sacfmt::read_spec 103
sacfmt::Trace 104
sacfmt::bitset_type::uint< nbits > 188
sacfmt::bitset_type::uint< 4 ∗bits_per_byte > 188
sacfmt::bitset_type::uint< bytes ∗bits_per_byte > 189
sacfmt::word_pair< T > 189

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

sacfmt::io_error
: Class for generic I/O exceptions . . . 101

sacfmt::read_spec
: Struct that specifies parameters for reading . . . 103

sacfmt::Trace
: The Trace class . . . 104

sacfmt::bitset_type::uint< nbits >
: Ensure type-safety for conversions between floats/doubles and bitsets . . . 188

sacfmt::bitset_type::uint< 4 *bits_per_byte >
: One-word (floats) . . . 188

sacfmt::bitset_type::uint< bytes *bits_per_byte >
: Two-words (doubles) . . . 189

sacfmt::word_pair< T >
: Struct containing a pair of words . . . 189

Chapter 10

Namespace Documentation

10.1 sacfmt Namespace Reference

sac-format namespace

Namespaces

- namespace bitset_type

  bitset type-safety namespace.

Classes

- class io_error

  Class for generic I/O exceptions.
- struct read_spec

  Struct that specifies parameters for reading.
- class Trace

  The Trace class.
- struct word_pair

  Struct containing a pair of words.

Typedefs

- using char_bit = std::bitset< bits_per_byte >

  One binary character (useful for building strings).
- using word_one = std::bitset< binary_word_size >

  One binary word (useful for non-strings).
- using word_two = std::bitset< static_cast< size_t >(2) ∗binary_word_size >

  Two binary words (useful for strings).
- using word_four = std::bitset< static_cast< size_t >(4) ∗binary_word_size >

  Four binary words (kEvNm only).
- template<class T >
  using unsigned_int = typename bitset_type::uint< sizeof(T) ∗bits_per_byte >::type

  Convert variable to unsigned-integer using type-safe conversions.

Enumerations

- enum class name {
  depmin , depmax , odelta , resp0 ,
  resp1 , resp2 , resp3 , resp4 ,
  resp5 , resp6 , resp7 , resp8 ,
  resp9 , stel , stdp , evel ,
  evdp , mag , user0 , user1 ,
  user2 , user3 , user4 , user5 ,
  user6 , user7 , user8 , user9 ,
  dist , az , baz , gcarc ,
  depmen , cmpaz , cmpinc , xminimum ,
  xmaximum , yminimum , ymaximum , delta ,
  b , e , o , a ,
  t0 , t1 , t2 , t3 ,
  t4 , t5 , t6 , t7 ,
  t8 , t9 , f , stla ,
  stlo , evla , evlo , sb ,
  sdelta , nzyear , nzjday , nzhour ,
  nzmin , nzsec , nzmsec , nvhdr ,
  norid , nevid , npts , nsnpts ,
  nwfid , nxsize , nysize , iftype ,
  idep , iztype , iinst , istreg ,
  ievreg , ievtyp , iqual , isynth ,
  imagtyp , imagsrc , ibody , leven ,
  lpspol , lovrok , lcalda , kstnm ,
  kevnm , khole , ko , ka ,
  kt0 , kt1 , kt2 , kt3 ,
  kt4 , kt5 , kt6 , kt7 ,
  kt8 , kt9 , kf , kuser0 ,
  kuser1 , kuser2 , kcmpnm , knetwk ,
  kdatrd , kinst , data1 , data2 }

  Enumeration of all SAC fields.

Functions

- std::streamoff word_position (const size_t word_number) noexcept

  Calculates position of word in SAC-file.
- word_one uint_to_binary (uint num) noexcept

  Convert unsigned integer to 32-bit (one word) binary bitset.
- word_one int_to_binary (int num) noexcept

  Convert integer to 32-bit (one word) binary bitset.
- int binary_to_int (word_one bin) noexcept

  Convert 32-bit (one word) binary bitset to integer.
- word_one float_to_binary (const float num) noexcept

  Convert floating-point value to 32-bit (one word) binary bitset.
- float binary_to_float (const word_one &bin) noexcept

  Convert 32-bit (one word) binary bitset to a floating-point value.
- word_two double_to_binary (const double num) noexcept

  Convert double-precision value to 64-bit (two words) binary bitset.
- double binary_to_double (const word_two &bin) noexcept

  Convert 64-bit (two words) binary bitset to double-precision value.
- void remove_leading_spaces (std::string *str) noexcept

  Remove all leading spaces from a string.

- void remove_trailing_spaces (std::string *str) noexcept

  Remove all trailing spaces from a string.
- std::string string_cleaning (const std::string &str) noexcept

  Remove leading/trailing spaces and control characters from a string.
- void prep_string (std::string *str, const size_t str_size) noexcept

  Cleans string and then truncates/pads as necessary.
- template<typename T >
  void string_bits (T *bits, const std::string &str, const size_t str_size) noexcept

  Template function to convert string into binary bitset.
- template<typename T >
  std::string bits_string (const T &bits, const size_t num_words) noexcept

  Template function to convert binary bitset to string.
- word_two string_to_binary (std::string str) noexcept

  Convert string to a 64-bit (two word) binary bitset.
- std::string binary_to_string (const word_two &str) noexcept

  Convert a 64-bit (two word) binary bitset to a string.
- word_four long_string_to_binary (std::string str) noexcept

  Convert a string to a 128-bit (four word) binary bitset.
- std::string binary_to_long_string (const word_four &str) noexcept

  Convert a 128-bit (four word) binary bitset to a string.
- word_one bool_to_binary (const bool flag) noexcept

  Convert a boolean to a 32-bit (one word) binary bitset.
- bool binary_to_bool (const word_one &flag) noexcept

  Convert a 32-bit (one word) binary bitset to a boolean.
- word_two concat_words (const word_pair< word_one > &pair_words) noexcept

  Concatenate two word_one binary strings into a single word_two string.
- word_four concat_words (const word_pair< word_two > &pair_words) noexcept

  Concatenate two word_two binary strings into a single word_four string.
- bool nwords_after_current (std::ifstream *sac, const read_spec &spec) noexcept

  Determine if the SAC-file has enough remaining data to read the requested amount of data.
- void safe_to_read_header (std::ifstream *sac)

  Determine if the SAC-file is large enough to contain a complete header.
- void safe_to_read_footer (std::ifstream *sac)

  Determines if the SAC-file has enough space remaining to contain a complete footer.
- void safe_to_read_data (std::ifstream *sac, const size_t n_words, const bool data2)

  Determines if the SAC-file has enough space remaining to contain a complete data vector.
- void safe_to_finish_reading (std::ifstream *sac)

  Determines if the SAC-file is finished.
- word_one read_word (std::ifstream *sac)

  Read one word (32 bits, useful for non-strings) from a binary SAC-File.
- word_two read_two_words (std::ifstream *sac)

  Read two words (64 bits, useful for most strings) from a binary SAC-file.
- word_four read_four_words (std::ifstream *sac)

  Read four words (128 bits, kEvNm only) from a binary SAC-file.
- std::vector< double > read_data (std::ifstream *sac, const read_spec &spec)

  Reader arbitrary number of words (useful for vectors) from a binary SAC-file.
- void write_words (std::ofstream *sac_file, const std::vector< char > &input)

  Write arbitrary number of words (useful for vectors) to a binary SAC-file.
- template<typename T >
  std::vector< char > convert_to_word (const T input) noexcept

  Template function to convert input value into a std::vector<char> for writing.

- std::vector< char > convert_to_word (const double input) noexcept

  Convert double value into a std::vector<char> for writing.
- template<size_t N>
  std::array< char, N > convert_to_words (const std::string &str, int n_words) noexcept

  Template function to convert input string value into a std::array<char> for writing.
- std::vector< char > bool_to_word (const bool flag) noexcept

  Convert boolean to a word for writing.
- bool equal_within_tolerance (const std::vector< double > &vector1, const std::vector< double > &vector2, const double tolerance) noexcept

  Check if two std::vector<double> are equal within a tolerance limit.
- bool equal_within_tolerance (const double val1, const double val2, const double tolerance) noexcept

  Check if two double values are equal within a tolerance limit.
- double degrees_to_radians (const double degrees) noexcept

  Convert decimal degrees to radians.
- double radians_to_degrees (const double radians) noexcept

  Convert radians to decimal degrees.
- double gcarc (const double latitude1, const double longitude1, const double latitude2, const double longitude2) noexcept

  Calculate great circle arc distance in decimal degrees between two points.
- double azimuth (const double latitude1, const double longitude1, const double latitude2, const double longitude2) noexcept

  Calculate azimuth between two points.
- double limit_360 (const double degrees) noexcept

  Takes a decimal degree value and constrains it to full circle using symmetry.
- double limit_180 (const double degrees) noexcept

  Takes a decimal degree value and constrains it to a half circle using symmetry.
- double limit_90 (const double degrees) noexcept

  Takes a decimal degree value and constrains it to a quarter circle using symmetry.
- template std::vector< char > convert_to_word (const float input) noexcept
- template std::vector< char > convert_to_word (const int x) noexcept
- template std::array< char, word_length > convert_to_words (const std::string &str, const int n_words) noexcept

Variables

- constexpr size_t word_length {4}

  Size (bytes) of fundamental data-chunk.
- constexpr size_t bits_per_byte {8}

  Size (bits) of binary character.
- constexpr size_t binary_word_size {word_length * bits_per_byte}

  Size (bits) of funamental data-chunk.
- constexpr std::streamoff data_word {158}

  First word of (first) data-section (stream offset).
- constexpr int unset_int {-12345}

  Integer unset value (SAC Magic).
- constexpr float unset_float {-12345.0F}

  Float-point unset value (SAC Magic).
- constexpr double unset_double {-12345.0}

  Double-precision unset value (SAC Magic).
- constexpr bool unset_bool {false}

  Boolean unset value (SAC Magic).

- const std::string unset_word {"-12345"}

  String unset value (SAC Magic).
- constexpr float f_eps {2.75e-6F}

  Accuracy precision expected of SAC floating-point values.
- constexpr int ascii_space {32}

  ASCII-code of 'space' character.
- constexpr int num_float {39}

  Number of float-poing header values in SAC format.
- constexpr int num_double {22}

  Number of double-precision header values in SAC format.
- constexpr int num_int {26}

  Number of integer header values in SAC format.
- constexpr int num_bool {4}

  Number of boolean header values in SAC format.
- constexpr int num_string {23}

  Number of string header values in SAC format.
- constexpr int num_data {2}

  Number of data arrays in SAC format.
- constexpr int num_footer {22}

  Number of double-precision footer values in SAC format (version 7).
- constexpr int modern_hdr_version {7}

  nVHdr value for newest SAC format (2020+).
- constexpr int old_hdr_version {6}

  nVHdr value for historic SAC format (pre-2020).
- constexpr int common_skip_num {7}

  Extremely common number of 'internal use' headers in SAC format.
- constexpr double rad_per_deg {std::numbers::pi_v<double> / 180.0}

  Radians per degree.
- constexpr double deg_per_rad {1.0 / rad_per_deg}

  Degrees per radian.
- constexpr double circle_deg {360.0}

  Degrees in a circle.
- constexpr double earth_radius {6378.14}

  Average radius of Earth (kilometers).
- const std::unordered_map< name, const size_t > sac_map

  Lookup table for variable locations.

10.1.1 Detailed Description

sac-format namespace

10.1.2 Typedef Documentation

10.1.2.1 char_bit

```
using sacfmt::char_bit = typedef std::bitset<bits_per_byte>
```

One binary character (useful for building strings).

10.1.2.2 unsigned_int

```
template<class T >
using sacfmt::unsigned_int = typedef typename bitset_type::uint<sizeof(T) * bits_per_byte>↩
::type
```

Convert variable to unsigned-integer using type-safe conversions.

10.1.2.3 word_four

```
using sacfmt::word_four = typedef std::bitset<static_cast<size_t>(4) * binary_word_size>
```

Four binary words (kEvNm only).

10.1.2.4 word_one

```
using sacfmt::word_one = typedef std::bitset<binary_word_size>
```

One binary word (useful for non-strings).

10.1.2.5 word_two

```
using sacfmt::word_two = typedef std::bitset<static_cast<size_t>(2) * binary_word_size>
```

Two binary words (useful for strings).

10.1.3 Enumeration Type Documentation

10.1.3.1 name

```
enum class sacfmt::name  [strong]
```

Enumeration of all SAC fields.

Additional information can be found at SAC-file format

Enumerator

depmin	Float Minimum value of the dependent variable (displacement/velocity/acceleration/volts/counts).
depmax	Float Maximum value of the dependent variable.
odelta	Float Modified (observational) value of delta.
resp0	Float Instrument response parameter (poles, zeros, and a constant). Not used by SAC - free for other purposes.
resp1	See resp0.
resp2	See resp0.

Enumerator

resp3	See resp0.
resp4	See resp0.
resp5	See resp0.
resp6	See resp0.
resp7	See resp0.
resp8	See resp0.
resp9	See resp0.
stel	Float Station elevation in meters above sea level (m.a.s.l.). Not used by SAC - free for other purposes.
stdp	Float Station depth in meters below surface (borehole/buried vault). Not used by SAC - free for other purposes.
evel	Float Event elevation m.a.s.l. Not used by SAC - free for other purposes.
evdp	Float Event depth in kilometers (previous meters) below surface.
mag	Float Event magnitude.
user0	Float Storage for user-defined values.
user1	See user0.
user2	See user0.
user3	See user0.
user4	See user0.
user5	See user0.
user6	See user0.
user7	See user0.
user8	See user0.
user9	See user0.
dist	Float Station-Event distance in kilometers.
az	Float Azimuth $Station \rightarrow Event$ in decimal degrees from North.
baz	Float Back-Azimuth $Event \rightarrow Station$ in decimal degrees from North.
gcarc	Float Great-circle arc-distance between station and event in decimal degrees.
depmen	Float Mean value of dependent variable.
cmpaz	Float Instrument measurement azimuth, decimal degrees from North.
cmpinc	Float Instrument measurement incidence angle, decimal degrees from upward vertical (incident 0 = dip -90). Note: SEED/MINISEED use dip angle, decimal degrees from horizontal (dip 0 = incident 90).
xminimum	Float Spectral-only equivalent of depmin (f_0 or ω_0).
xmaximum	Float Spectral-only equivalent of depman (f_{max} or ω_{max}).

Enumerator

yminimum	Float Spectral-only equivalent of b.
ymaximum	Float Spectral-only equivalent of e.
delta	Double Increment between evenly-spaced samples (Δt for timeseries, Δf or $\Delta\omega$ for spectral).
b	Double First value (beginning) of independent variable (t_0).
e	Double Final value (ending) of the independent variable (t_{max}).
o	Double Event origin time, in seconds relative to the reference time.
a	Double Event first arrival time, in seconds relative to the reference time.
t0	Double User defined time value, in seconds relative to the reference time.
t1	See t0.
t2	See t0.
t3	See t0.
t4	See t0.
t5	See t0.
t6	See t0.
t7	See t0.
t8	See t0.
t9	See t0.
f	Double Event end (fini) time, in seconds relative to the reference time.
stla	Double Station latitude in decimal degrees, N/S is positive/negative. sac-format automatically enforces $\phi \in [-90, 90]$.
stlo	Double Station longitude in decimal degrees, E/W is positive/negative. sac-format automaticall enforces $\lambda \in [-180, 180]$.
evla	Double Event latitude in decimal degrees, N/S is positive/negative. sac-format automatically enforces $\phi \in [-90, 90]$.
evlo	Double Event longitude in decimal degrees, E/W is positive/negative. sac-format automatically enforces $\lambda \in [-180, 180]$.
sb	Double Original (saved) value of b (beginning).
sdelta	Double Original (saved) value of delta (sample-spacing).
nzyear	Integer Reference time GMT year.
nzjday	Integer Reference time GMT day-of-year (often called Julian Date). 1-366 Not enforced.
nzhour	Integer Reference time GMT hour. 00-23 Not enforced.

Enumerator

nzmin	Integer Reference time GMT minute. 00-59 Not enforced.
nzsec	Integer Reference time GMT second. 00-59 Not enforced.
nzmsec	Integer Reference time GMT millisecond. 0-999 not enforced.
nvhdr	Integer SAC-file version. 7 = 2020+, sac 102.0+, has a Footer. 6 = pre-2020, sac 101.6a-, no Footer.
norid	Integer Origin ID.
nevid	Integer Event ID.
npts	Integer Number of points in data.
nsnpts	Integer Original (saved) npts.
nwfid	Integer Waveform ID.
nxsize	Integer Spectral-only equivalent of npts (length of spectrum).
nysize	Integer Spectral-only; width of spectrum.
iftype	Integer File type.
idep	Integer Dependent variable type.
iztype	Integer Reference time equivalent.
iinst	Integer Recording instrument type. Not used by SAC - free for other purposes.
istreg	Integer Station geographic region. Not used by SAC - free for other purposes.
ievreg	Integer Event geographic region. Not used by SAC - free for other purposes.
ievtyp	Integer Event type. Not used by SAC - free for other purposes.
iqual	Integer Quality of data. Not used by SAC - free for other purposes.
isynth	Integer Synthetic data flag. Not used by SAC - free for other purposes.
imagtyp	Integer Magnitude type.

Enumerator

imagsrc	Integer Magnitude information source.
ibody	Integer Body/spheroid definition used to calculate distances. Not currently-used by sac-format (SAC does used it).
leven	Boolean REQUIRED Evenly-spaced data flag. True = even.
lpspol	Boolean Station polarity flag. True = positive (left-handed, e.g. North-East-Up).
lovrok	Boolean File overwrite flag. If true, okay to overwrite file. Not used by sac-format.
lcalda	Boolean Calculate geometry flag. Not used by sac-format.
kstnm	String (2 words) Station name.
kevnm	String (4 words) Event name.
khole	String (2 words) Nuclear-Hole identifier. Other-Location identifier (LOCID).
ko	String (2 words) Text for o.
ka	String (2 words) Text for a.
kt0	String (2 words) Text for t0
kt1	See kt0.
kt2	See kt0.
kt3	See kt0.
kt4	See kt0.
kt5	See kt0.
kt6	See kt0.
kt7	See kt0.
kt8	See kt0.
kt9	See kt0.
kf	String (2 words) Text for f.
kuser0	String (2 words) Text for user0.
kuser1	See kuser0.
kuser2	See kuser0.
kcmpnm	String (2 words) Component name.
knetwk	String (2 words) Network name.
kdatrd	String (2 words) Date the data was read onto a computer.

Enumerator

kinst	String (2 words) Instrument name.
data1	std::vector<double> First data vector. ALWAYS present, ALWAYS begins at word 158.
data2	std::vector<double> Second data vector. CONDITIONAL present. IF PRESENT, begins at end of data1. Required if leven is false (uneven sampling), or if iftype is spectral/XY/XYZ.

```
00280                {
00281    // Floats
00288    depmin,
00294    depmax,
00300    odelta,
00308    resp0,
00310    resp1,
00312    resp2,
00314    resp3,
00316    resp4,
00318    resp5,
00320    resp6,
00322    resp7,
00324    resp8,
00326    resp9,
00334    stel,
00342    stdp,
00350    evel,
00356    evdp,
00362    mag,
00368    user0,
00370    user1,
00372    user2,
00374    user3,
00376    user4,
00378    user5,
00380    user6,
00382    user7,
00384    user8,
00386    user9,
00392    dist,
00399    az,
00406    baz,
00412    gcarc,
00418    depmen,
00424    cmpaz,
00434    cmpinc,
00441    xminimum,
00448    xmaximum,
00454    yminimum,
00460    ymaximum,
00461    // Doubles
00470    delta,
00476    b,
00483    e,
00489    o,
00495    a,
00501    t0,
00503    t1,
00505    t2,
00507    t3,
00509    t4,
00511    t5,
00513    t6,
00515    t7,
00517    t8,
00519    t9,
00525    f,
00533    stla,
00541    stlo,
00549    evla,
00557    evlo,
00563    sb,
00569    sdelta,
00570    // Ints
00576    nzyear,
00584    nzjday,
00592    nzhour,
00600    nzmin,
00608    nzsec,
00616    nzmsec,
00625    nvhdr,
```

```
00631   norid,
00637   nevid,
00643   npts,
00649   nsnpts,
00655   nwfid,
00661   nxsize,
00667   nysize,
00673   iftype,
00679   idep,
00685   iztype,
00693   iinst,
00701   istreg,
00709   ievreg,
00717   ievtyp,
00725   iqual,
00733   isynth,
00739   imagtyp,
00745   imagsrc,
00753   ibody,
00754   // Bools
00762   leven,
00770   lpspol,
00780   lovrok,
00788   lcalda,
00789   // Strings
00795   kstnm,
00801   kevnm,
00809   khole,
00815   ko,
00821   ka,
00827   kt0,
00829   kt1,
00831   kt2,
00833   kt3,
00835   kt4,
00837   kt5,
00839   kt6,
00841   kt7,
00843   kt8,
00845   kt9,
00851   kf,
00857   kuser0,
00859   kuser1,
00861   kuser2,
00867   kcmpnm,  // missing in org documentation
00873   knetwk,  // missing in org documentation
00879   kdatrd,
00885   kinst,
00886   // Data
00892   data1,
00901   data2
00902 };
```

10.1.4 Function Documentation

10.1.4.1 azimuth()

```
double sacfmt::azimuth (
            const double latitude1,
            const double longitude1,
            const double latitude2,
            const double longitude2 )  [noexcept]
```

Calculate azimuth between two points.

Assumes spherical Earth (in future may update to solve on a more general body).

ϕ is latitude. λ is longitude. θ is azimuth.

$$\theta = tan^{-1}\left(\frac{sin(\delta\lambda)cos(\phi_2)}{cos(\phi_1)sin(\phi_2) - sin(\phi_1)cos(\phi_2)cos(\delta\lambda)}\right)$$

Parameters

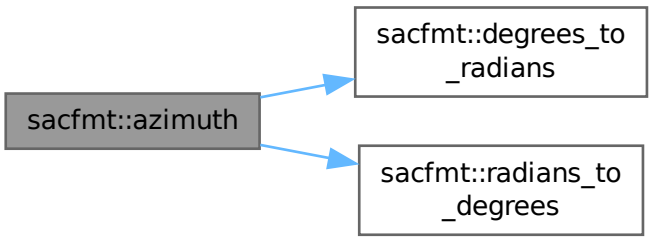
in	*latitude1*	Latitude of first location in decimal degrees.
in	*longitude1*	Longitude of first location in decimal degrees.
in	*latitude2*	Latitude of second location in decimal degrees.
in	*longitude2*	Longitude of second location in decimal degrees.

Returns

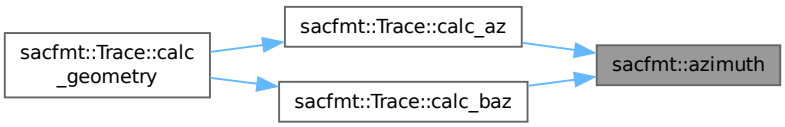
double The azimuth from the first location to the second location.

```
00738                                                                                {
00739   const double lat1{degrees_to_radians(latitude1)};
00740   const double lon1{degrees_to_radians(longitude1)};
00741   const double lat2{degrees_to_radians(latitude2)};
00742   const double lon2{degrees_to_radians(longitude2)};
00743   const double dlon{lon2 - lon1};
00744   const double numerator{std::sin(dlon) * std::cos(lat2)};
00745   const double denominator{(std::cos(lat1) * std::sin(lat2)) -
00746                            (std::sin(lat1) * std::cos(lat2) * std::cos(dlon))};
00747   double result{radians_to_degrees(std::atan2(numerator, denominator))};
00748   while (result < 0.0) {
00749     result += circle_deg;
00750   }
00751   return result;
00752 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.2 binary_to_bool()

```
bool sacfmt::binary_to_bool (
            const word_one & flag )  [noexcept]
```

Convert a 32-bit (one word) binary bitset to a boolean.

Parameters

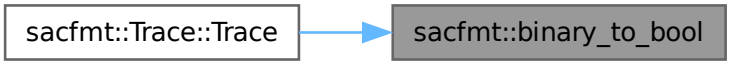
in	*flag*	word_one binary bitset to be converted (takes zeroth element).

Returns

boolean Converted boolean value.

```
00357 { return flag[0]; }
```

Here is the caller graph for this function:



10.1.4.3 binary_to_double()

```
double sacfmt::binary_to_double (
            const word_two & bin )  [noexcept]
```

Convert 64-bit (two words) binary bitset to double-precision value.

Converts bitset to unsigned long long then to double.

Parameters

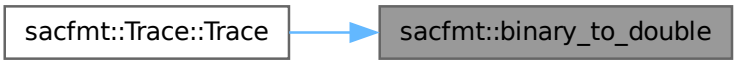
in	*bin*	word_two Binary value to be converted.

Returns

double Converted value.

```
00159                                                    {
00160   const auto val = bin.to_ullong();
00161   double result{};
00162   // flawfinder: ignore
00163   memcpy(&result, &val, sizeof(double));
00164   return result;
00165 }
```

Here is the caller graph for this function:



10.1.4.4 binary_to_float()

```
float sacfmt::binary_to_float (
            const word_one & bin )  [noexcept]
```

Convert 32-bit (one word) binary bitset to a floating-point value.

Converts bitset to unsigned long then to float.

Parameters

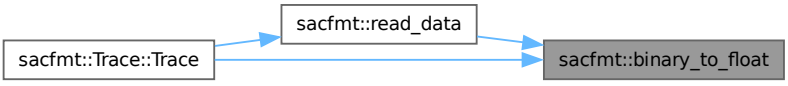
in	*bin*	word_one Binary value to be converted.

Returns

float Converted value.

```
00127                                                   {
00128   const auto val = bin.to_ulong();
00129   float result{};
00130   // flawfinder: ignore
00131   memcpy(&result, &val, sizeof(float));
00132   return result;
00133 }
```

Here is the caller graph for this function:



10.1.4.5 binary_to_int()

```
int sacfmt::binary_to_int (
            word_one bin )  [noexcept]
```

Convert 32-bit (one word) binary bitset to integer.

Uses two's complement to convert a binary value into an integer.

Parameters

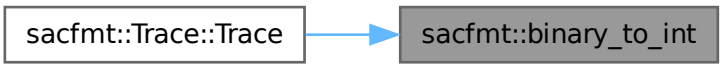
in	*bin*	Binary value to be converted.

Returns

int Converted value.

```
00088                                           {
00089   int result{};
00090   if (bin.test(binary_word_size - 1)) {
00091     // Complement
00092     bin.flip();
00093     result = static_cast<int>(bin.to_ulong());
00094     result += 1;
00095     // Change sign to make it negative
00096     result *= -1;
00097   } else {
00098     result = static_cast<int>(bin.to_ulong());
00099   }
00100   return result;
00101 }
```

Here is the caller graph for this function:



10.1.4.6 binary_to_long_string()

```
std::string sacfmt::binary_to_long_string (
            const word_four & str )  [noexcept]
```

Convert a 128-bit (four word) binary bitset to a string.

Exclusively used to work with the kEvNm header.

Parameters

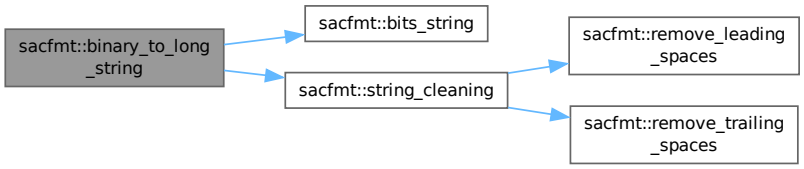
in	*str*	word_four to be converted to a string.

Returns

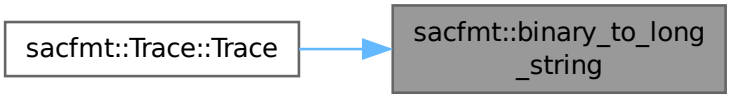
std::string Converted string.

```
00332                                                                   {
00333   std::string result{bits_string(str, 4)};
00334   return string_cleaning(result);
00335 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.7 binary_to_string()

```
std::string sacfmt::binary_to_string (
            const word_two & str )  [noexcept]
```

Convert a 64-bit (two word) binary bitset to a string.

Parameters

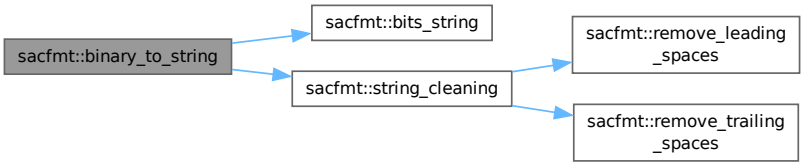
in	*str*	word_two to be converted to a string.

Returns

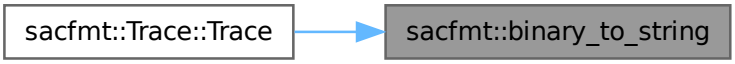
std::string Converted string.

```
00298                                                                {
00299   std::string result{bits_string(str, 2)};
00300   return string_cleaning(result);
00301 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.8 bits_string()

```
template<typename T >
std::string sacfmt::bits_string (
            const T & bits,
            const size_t num_words )  [noexcept]
```

Template function to convert binary bitset to string.

Parameters

in	*bits*	Source bitset for the string.
in	*num_words*	Length of string in words (4 chars = 1 word)

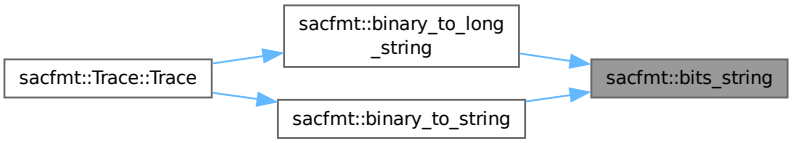
Returns

std::string String converted from bitset.

```
00258                                                                {
00259   std::string result{};
00260   result.reserve(num_words * word_length);
00261   constexpr size_t char_size{bits_per_byte};
```

```
00262   char_bit byte{};
00263   for (size_t i{0}; i < num_words * binary_word_size; i += char_size) {
00264     for (size_t j{0}; j < char_size; ++j) [[likely]] {
00265       byte[j] = bits[i + j];
00266     }
00267     result.push_back(static_cast<char>(byte.to_ulong()));
00268   }
00269   return result;
00270 }
```

Here is the caller graph for this function:



10.1.4.9 bool_to_binary()

```
word_one sacfmt::bool_to_binary (
            const bool flag )  [noexcept]
```

Convert a boolean to a 32-bit (one word) binary bitset.

Parameters

in	*flag*	Boolean value to be converted to a bitset (sets zeroth element).

Returns

word_one Converted binary bitset.

```
00344                                                {
00345   word_one result{};
00346   result[0] = flag;
00347   return result;
00348 }
```

10.1.4.10 bool_to_word()

```
std::vector< char > sacfmt::bool_to_word (
            const bool flag )  [noexcept]
```

Convert boolean to a word for writing.

Parameters

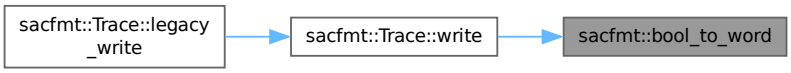
in	*flag*	Boolean to be converted.

Returns

std::vector<char> Prepared value for writing.

```
00598                                                          {
00599   std::vector<char> result;
00600   result.resize(word_length);
00601   result[0] = static_cast<char>(flag ? 1 : 0);
00602   for (size_t i{1}; i < word_length; ++i) {
00603     result[i] = 0;
00604   }
00605   return result;
00606 }
```

Here is the caller graph for this function:



10.1.4.11 concat_words() [1/2]

```
word_two sacfmt::concat_words (
            const word_pair< word_one > & pair_words )  [noexcept]
```

Concatenate two word_one binary strings into a single word_two string.

Useful for reading strings from SAC-files.

Parameters

in	*pair_words*	word_pair Words to be concatenated.

Returns

word_two Concatenated words.

```
00368                                                                  {
00369   word_two result{};
00370   for (size_t i{0}; i < binary_word_size; ++i) [[likely]] {
00371     result[i] = pair_words.first[i];
00372     result[i + binary_word_size] = pair_words.second[i];
00373   }
00374   return result;
00375 }
```

Here is the caller graph for this function:

10.1.4.12 concat_words() [2/2]

```
word_four sacfmt::concat_words (
            const word_pair< word_two > & pair_words )  [noexcept]
```

Concatenate two word_two binary strings into a single word_four string.

Exclusively used to read kEvNm header from SAC-file.

Parameters

in	*pair_words*	word_pair Words to be concatenated.

Returns

word_four Concatenated words.

```
00386                                                                    {
00387   word_four result{};
00388   constexpr size_t two_words{2 * binary_word_size};
00389   for (size_t i{0}; i < two_words; ++i) [[likely]] {
00390     result[i] = pair_words.first[i];
00391     result[i + two_words] = pair_words.second[i];
00392   }
00393   return result;
00394 }
```

10.1.4.13 convert_to_word() [1/4]

```
std::vector< char > sacfmt::convert_to_word (
            const double input )  [noexcept]
```

Convert double value into a std::vector<char> for writing.

Parameters

in	*input*	Input value to convert (double).

Returns

std::vector<char> Prepared for writing to binary SAC-file.

```
00550                                                                {
00551   std::array<char, static_cast<size_t>(2) * word_length> tmp{};
00552   // Copy bytes from input into the tmp array
00553   // flawfinder: ignore
00554   std::memcpy(tmp.data(), &input, static_cast<size_t>(2) * word_length);
00555   std::vector<char> word{};
00556   word.resize(static_cast<size_t>(2) * word_length);
00557   for (size_t i{0}; i < 2 * word_length; ++i) {
00558     word[i] = tmp[i];
00559   }
00560   return word;
00561 }
```

10.1.4.14 convert_to_word() [2/4]

```
template std::vector< char > sacfmt::convert_to_word (
            const float input )  [noexcept]
```

10.1.4.15 convert_to_word() [3/4]

```
template std::vector< char > sacfmt::convert_to_word (
            const int x )  [noexcept]
```

10.1.4.16 convert_to_word() [4/4]

```
template<typename T >
std::vector< char > sacfmt::convert_to_word (
            const T input )  [noexcept]
```

Template function to convert input value into a std::vector<char> for writing.

Parameters

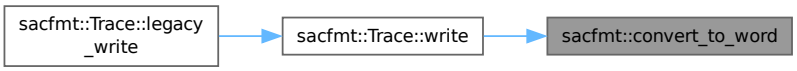
in	*input*	Input value (float or int) to convert.

Returns

std::vector<char> Prepared for writing to binary SAC-file.

```
00527                                                          {
00528   std::array<char, word_length> tmp{};
00529   // Copy bytes from input into the tmp array
00530   // flawfinder: ignore
00531   std::memcpy(tmp.data(), &input, word_length);
00532   std::vector<char> word{};
00533   word.resize(word_length);
00534   for (size_t i{0}; i < word_length; ++i) [[likely]] {
00535     word[i] = tmp[i];
00536   }
00537   return word;
00538 }
```

Here is the caller graph for this function:



10.1.4.17 convert_to_words() [1/2]

```
template std::array< char, word_length > sacfmt::convert_to_words (
            const std::string & str,
            const int n_words )  [noexcept]
```

10.1.4.18 convert_to_words() [2/2]

```
template<size_t N>
template std::array< char, 4 *word_length > sacfmt::convert_to_words (
            const std::string & str,
            int n_words )  [noexcept]
```

Template function to convert input string value into a std::array<char> for writing.

Parameters

in	*str*	Input string to convert.
in	*n_words*	Number of words

Returns

std::array<char, N> Prepared for writing to a binary SAC-file.

```
00574                                                                {
00575   std::array<char, N> all_words{};
00576   // String to null-terminated character array
00577   const char *c_str = str.c_str();
00578   for (size_t i{0}; i < static_cast<size_t>(n_words) * word_length; ++i) {
00579     all_words[i] = c_str[i];
00580   }
00581   return all_words;
00582 }
```

10.1.4.19 degrees_to_radians()

```
double sacfmt::degrees_to_radians (
            const double degrees )  [noexcept]
```

Convert decimal degrees to radians.

$$r = d \cdot \frac{\pi}{180^\circ}$$

Parameters

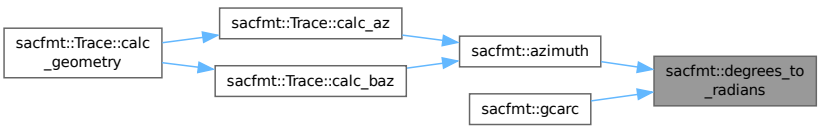
in	*degrees*	Angle in decimal degrees to be converted.

Returns

double Angle in radians.

```
00661                                                    {
00662   return rad_per_deg * degrees;
00663 }
```

Here is the caller graph for this function:



10.1.4.20 double_to_binary()

```
word_two sacfmt::double_to_binary (
            const double num )  [noexcept]
```

Convert double-precision value to 64-bit (two words) binary bitset.

Converts double to unsigned-integer of same size for storage in bitset.

Parameters

in	*num*	Double value to be converted.

Returns

word_two Converted value.

```
00143                                                    {
00144   unsigned_int<double> num_as_uint{0};
00145   // flawfinder: ignore
00146   std::memcpy(&num_as_uint, &num, sizeof(double));
00147   word_two result{num_as_uint};
00148   return result;
00149 }
```

10.1.4.21 equal_within_tolerance() [1/2]

```
bool sacfmt::equal_within_tolerance (
            const double val1,
            const double val2,
            const double tolerance )  [noexcept]
```

Check if two double values are equal within a tolerance limit.

Default tolerance is f_eps.

Parameters

in	*val1*	First double in comparison.
in	*val2*	Second double in comparison.
in	*tolerance*	Numerical equality tolerance (default f_eps).

Returns

bool Boolean equality value.

```
00647                                                                {
00648   return std::abs(val1 - val2) < tolerance;
00649 }
```

10.1.4.22 equal_within_tolerance() [2/2]

```
bool sacfmt::equal_within_tolerance (
            const std::vector< double > & vector1,
```

```
        const std::vector< double > & vector2,
        const double tolerance )  [noexcept]
```

Check if two std::vector<double> are equal within a tolerance limit.

Default tolerance is f_eps.

Parameters

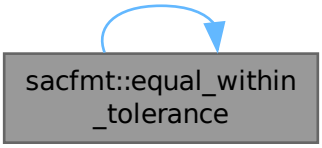
in	*vector1*	First data vector in comparison.
in	*vector2*	Second data vector in comparison.
in	*tolerance*	Numerical equality tolerance (default f_eps).

Returns

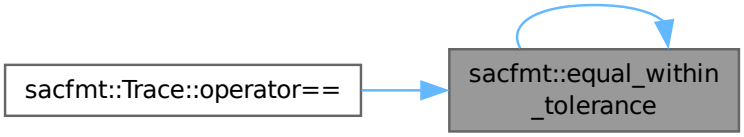
bool Boolean equality value.

```
00624                                                                    {
00625   if (vector1.size() != vector2.size()) {
00626     return false;
00627   }
00628   for (size_t i{0}; i < vector1.size(); ++i) [[likely]] {
00629     if (!equal_within_tolerance(vector1[i], vector2[i], tolerance)) {
00630       return false;
00631     }
00632   }
00633   return true;
00634 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.23 float_to_binary()

```
word_one sacfmt::float_to_binary (
            const float num )  [noexcept]
```

Convert floating-point value to 32-bit (one word) binary bitset.

Converts float to unsigned-integer of same size for storage in bitset.

Parameters

in	*num*	Float value to be converted.

Returns

word_one Converted value.

```
00111                                                {
00112   unsigned_int<float> num_as_uint{0};
00113   // flawfinder: ignore
00114   std::memcpy(&num_as_uint, &num, sizeof(float));
00115   word_one result{num_as_uint};
00116   return result;
00117 }
```

10.1.4.24 gcarc()

```
double sacfmt::gcarc (
            const double latitude1,
            const double longitude1,
            const double latitude2,
            const double longitude2 )  [noexcept]
```

Calculate great circle arc distance in decimal degrees between two points.

Assumes spherical Earth (in future will include flatenning and adjustable radius for other bodies/greater accuracy).

ϕ is latitude. λ is longitude. Δ is great circle arc distance (gcarc).

$$\Delta = cos^{-1}\left(sin(\phi_1)sin(\phi_2) + cos(\phi_1)cos(\phi_2)cos(\lambda_2 - \lambda_1)\right)$$

Parameters

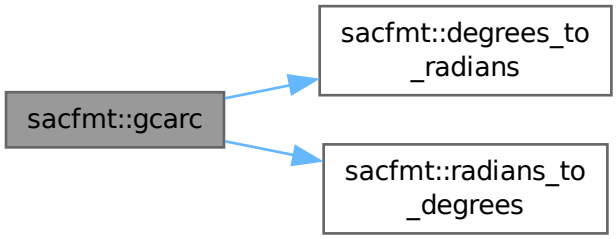
in	*latitude1*	Latitude of first location in decimal degrees.
in	*longitude1*	Longitude of first location in decimal degrees.
in	*latitude2*	Latitude of second location in decimal degrees.
in	*longitude2*	Longitude of second location in decimal degrees.

Returns

double The great circle arc distance in decimal degrees.

```
00704                                                                    {
00705   const double lat1{degrees_to_radians(latitude1)};
00706   const double lon1{degrees_to_radians(longitude1)};
00707   const double lat2{degrees_to_radians(latitude2)};
00708   const double lon2{degrees_to_radians(longitude2)};
00709   return radians_to_degrees(
00710       std::acos(std::sin(lat1) * std::sin(lat2) +
00711                 std::cos(lat1) * std::cos(lat2) * std::cos(lon2 - lon1)));
00712 }
```

Here is the call graph for this function:



10.1.4.25 int_to_binary()

```
word_one sacfmt::int_to_binary (
             int num )  [noexcept]
```

Convert integer to 32-bit (one word) binary bitset.

Uses two's complement to convert an integer into a binary value.

Parameters

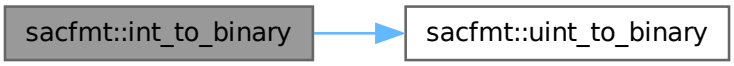
in	*num*	Number to be converted.

Returns

word_one Converted value.

```
00067                                          {
00068   word_one bits{};
00069   if (num >= 0) {
00070     bits = uint_to_binary(static_cast<uint>(num));
00071   } else {
00072     bits = uint_to_binary(static_cast<uint>(-num));
00073     // Complement
00074     bits.flip();
00075     bits = bits.to_ulong() + 1;
00076   }
00077   return bits;
00078 }
```

Here is the call graph for this function:



10.1.4.26 limit_180()

```
double sacfmt::limit_180 (
            const double degrees )  [noexcept]
```

Takes a decimal degree value and constrains it to a half circle using symmetry.

$$[-\infty, \infty] \rightarrow (-180, 180]$$

Parameters

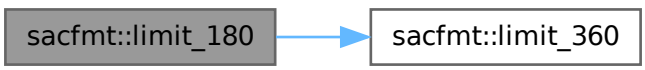
in	*degrees*	Decimal degrees to be constrained.

Returns

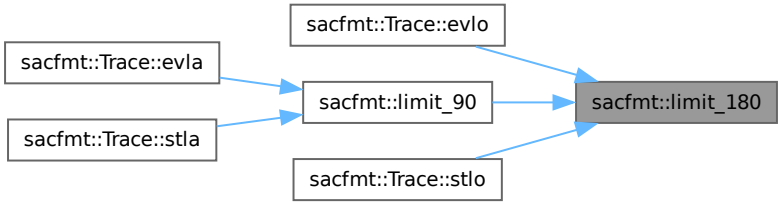
double Value within limits.

```
00791                                                {
00792   double result{limit_360(degrees)};
00793   constexpr double hemi{180.0};
00794   if (result > hemi) {
00795     result = result - circle_deg;
00796   }
00797   return result;
00798 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.27 limit_360()

```
double sacfmt::limit_360 (
            const double degrees )  [noexcept]
```

Takes a decimal degree value and constrains it to full circle using symmetry.

$$[-\infty, \infty] \rightarrow [0, 360]$$

Parameters

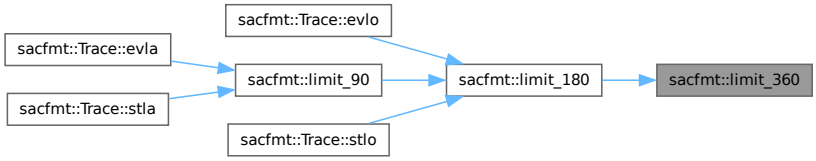
in	*degrees*	Decimal degrees to be constrained.

Returns

double Value within limits.

```
00765                                                {
00766   double result{degrees};
00767   while (std::abs(result) > circle_deg) {
00768     if (result > circle_deg) {
00769       result -= circle_deg;
00770     } else {
00771       result += circle_deg;
00772     }
00773   }
00774   if (result < 0) {
00775     result += circle_deg;
00776   }
00777   return result;
00778 }
```

Here is the caller graph for this function:



10.1.4.28 limit_90()

```
double sacfmt::limit_90 (
            const double degrees )  [noexcept]
```

Takes a decimal degree value and constrains it to a quarter circle using symmetry.

$$[-\infty, \infty] \rightarrow [-90, 90]$$

Parameters

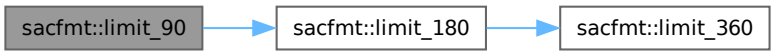
in	*degrees*	Decimal degrees to be constrained.

Returns

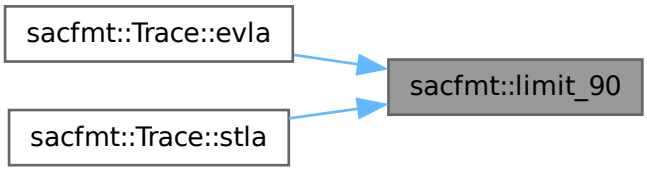
double Value within limits.

```
00811                                                {
00812   double result{limit_180(degrees)};
00813   constexpr double quarter{90.0};
00814   if (result > quarter) {
00815     result = (2 * quarter) - result;
00816   } else if (result < -quarter) {
00817     result = (-2 * quarter) - result;
00818   }
00819   return result;
00820 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.29 long_string_to_binary()

```
word_four sacfmt::long_string_to_binary (
            std::string str )  [noexcept]
```

Convert a string to a 128-bit (four word) binary bitset.

If the string is longer than 16 characters, then only the first 16 characters are kept. If the string is less than 16 characters long, it is right-padded with spaces.

Exclusively used to work with the kEvNm header.

Parameters

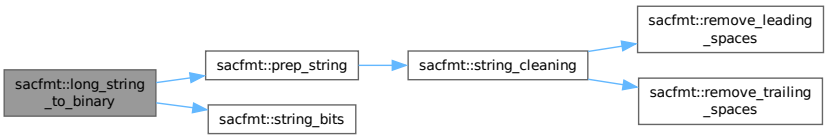
in	*str*	String to be converted to a bitset.

Returns

word_four Converted binary bitset.

```
00315                                                          {
00316   constexpr size_t string_size{4 * word_length};
00317   prep_string(&str, string_size);
00318   // Four words (16 characters)
00319   word_four bits{};
00320   string_bits(&bits, str, string_size);
00321   return bits;
00322 }
```

Here is the call graph for this function:



10.1.4.30 nwords_after_current()

```
bool sacfmt::nwords_after_current (
            std::ifstream * sac,
            const read_spec & spec )  [noexcept]
```

Determine if the SAC-file has enough remaining data to read the requested amount of data.

Parameters

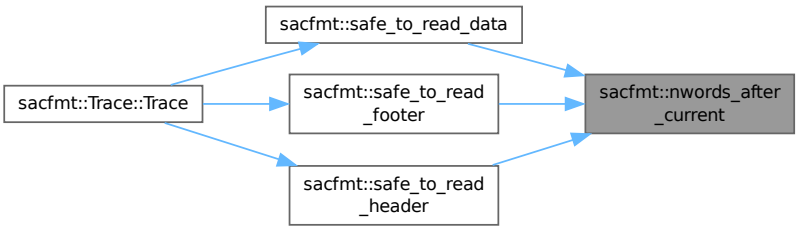
in	*sac*	std::ifstream* SAC-file to read.
in	*spec*	read_spec reading specification.

Returns

bool Truth value (true = safe to read).

```
01640                                                                    {
01641   bool result{false};
01642   if (sac->good()) {
01643     sac->seekg(0, std::ios::end);
01644     const std::size_t final_pos{static_cast<size_t>(sac->tellg())};
01645     // Doesn't like size_t since it wants to allow
01646     // the possibility of negative offsets (not how I use it)
01647     sac->seekg(static_cast<std::streamoff>(spec.start_word));
01648     const std::size_t diff{final_pos - spec.start_word};
01649     result = (diff >= (spec.num_words * word_length));
01650   }
01651   return result;
01652 }
```

Here is the caller graph for this function:



10.1.4.31 prep_string()

```
void sacfmt::prep_string (
            std::string * str,
            const size_t str_size )  [noexcept]
```

Cleans string and then truncates/pads as necessary.

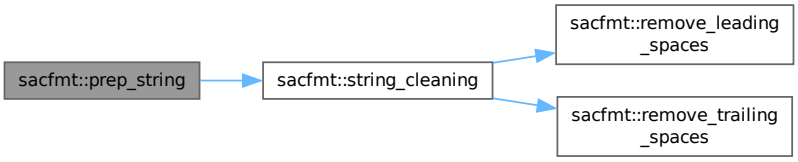
This edits the string in-place.

Parameters

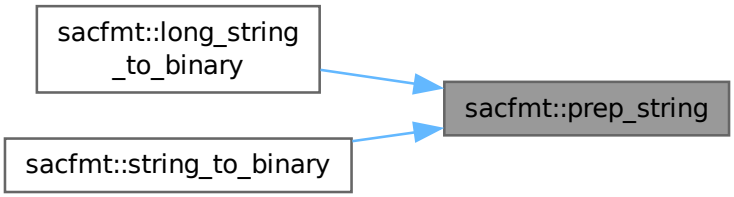
in,out	*str*	std::string* String to be prepared.
in	*str_size*	Desired string length.

```
00218                                                                    {
00219   *str = string_cleaning(*str);
00220   if (str->length() > str_size) {
00221     str->resize(str_size);
00222   } else if (str->length() < str_size) {
00223     *str = str->append(str_size - str->length(), ' ');
00224   }
00225 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.32 radians_to_degrees()

```
double sacfmt::radians_to_degrees (
            const double radians )  [noexcept]
```

Convert radians to decimal degrees.

$$d = r \cdot \frac{180^\circ}{\pi}$$

Parameters

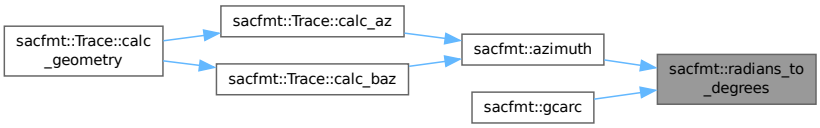
in	*radians*	Angle in radians to be converted.

Returns

double Angle in decimal degrees.

```
00675                                                   {
00676     return deg_per_rad * radians;
00677 }
```

Here is the caller graph for this function:



10.1.4.33 read_data()

```
std::vector< double > sacfmt::read_data (
            std::ifstream * sac,
            const read_spec & spec )
```

Reader arbitrary number of words (useful for vectors) from a binary SAC-file.

Note that this modifies the position of the reader within the stream (to the end of the read words).

Parameters

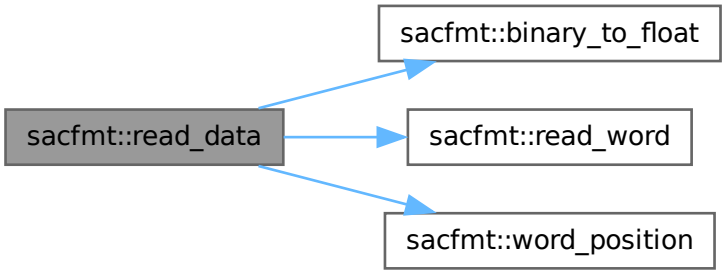
in,out	*sac*	std::ifstream* Input binary SAC-file.
in	*spec*	read_spec Reading specification.

Returns

std::vector<double> Data vector read in.

```
00487                                                                    {
00488   sac->seekg(word_position(spec.start_word));
00489   std::vector<double> result{};
00490   result.resize(spec.num_words);
00491   for (size_t i{0}; i < spec.num_words; ++i) [[likely]] {
00492     result[i] = static_cast<double>(binary_to_float(read_word(sac)));
00493   }
00494   return result;
00495 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

10.1.4.34 read_four_words()

```
word_four sacfmt::read_four_words (
            std::ifstream * sac )
```

Read four words (128 bits, kEvNm only) from a binary SAC-file.

Note that this modifies the position of the reader within the stream (to the end of the read words).

Parameters

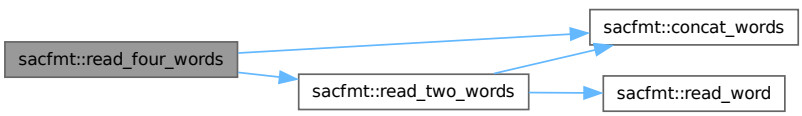
in,out	sac	std::ifstream* Input binary SAC-file.

Returns

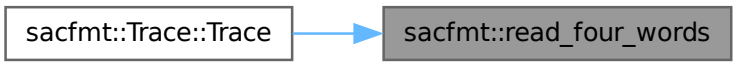
word_four Binary bitset representation of four words.

```
00462                                           {
00463   const word_two first_words{read_two_words(sac)};
00464   const word_two second_words{read_two_words(sac)};
00465   word_pair<word_two> pair_words{};
00466   if constexpr (std::endian::native == std::endian::little) {
00467     pair_words.first = first_words;
00468     pair_words.second = second_words;
00469   } else {
00470     pair_words.first = second_words;
00471     pair_words.second = first_words;
00472   }
00473   return concat_words(pair_words);
00474 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.35 read_two_words()

```
word_two sacfmt::read_two_words (
            std::ifstream * sac )
```

Read two words (64 bits, useful for most strings) from a binary SAC-file.

Note that this modifies the position of the reader within the stream (to the end of the read words).

Parameters

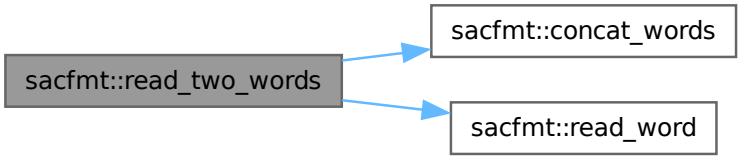
in,out	*sac*	std::ifstream* Input binary SAC-file.

Returns

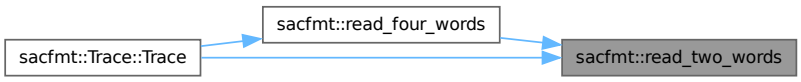
word_two Binary bitset representation of two words.

```
00439                                           {
00440   const word_one first_word{read_word(sac)};
00441   const word_one second_word{read_word(sac)};
00442   word_pair<word_one> pair_words{};
00443   if constexpr (std::endian::native == std::endian::little) {
00444     pair_words.first = first_word;
00445     pair_words.second = second_word;
00446   } else {
00447     pair_words.first = second_word;
00448     pair_words.second = first_word;
00449   }
00450   return concat_words(pair_words);
00451 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.36 read_word()

```
word_one sacfmt::read_word (
            std::ifstream * sac )
```

Read one word (32 bits, useful for non-strings) from a binary SAC-File.

Note that this modifies the position of the reader within the stream (to the end of the read word).

Parameters

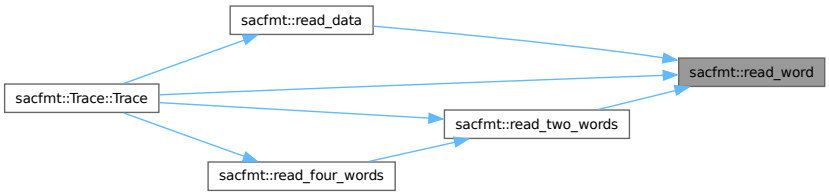
in,out	*sac*	std::ifstream* Input binary SAC-file.

Returns

word_one Binary bitset representation of single word.

```
00407                                   {
00408   word_one bits{};
00409   constexpr size_t char_size{bits_per_byte};
00410   // Where we will store the characters
00411   std::array<char, word_length> word{};
00412   // Read to our character array
00413   // This can always hold the source due to careful typing/sizing
00414   // flawfinder: ignore
00415   if (sac->read(word.data(), word_length)) {
00416     // Take each character
00417     for (size_t i{0}; i < word_length; ++i) [[likely]] {
00418       uint character{static_cast<uint>(word[i])};
00419       char_bit byte{character};
00420       // bit-by-bit
00421       for (size_t j{0}; j < char_size; ++j) [[likely]] {
00422         bits[(i * char_size) + j] = byte[j];
00423       }
00424     }
00425   }
00426   return bits;
00427 }
```

Here is the caller graph for this function:



10.1.4.37 remove_leading_spaces()

```
void sacfmt::remove_leading_spaces (
            std::string * str )  [noexcept]
```

Remove all leading spaces from a string.

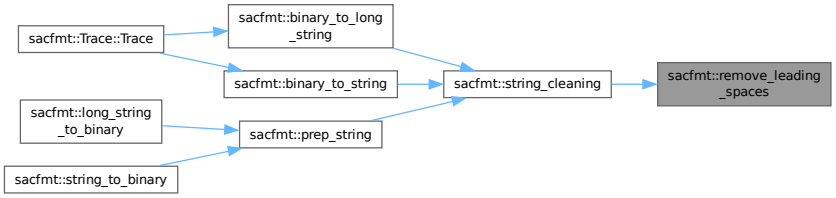
This edits the string in-place.

Parameters

in,out	*str*	std::string* String to have spaces removed.

```
00174                                                   {
00175   while ((static_cast<int>(str->front()) <= ascii_space) && (!str->empty())) {
00176     str->erase(0, 1);
00177   }
00178 }
```

Here is the caller graph for this function:



10.1.4.38 remove_trailing_spaces()

```
void sacfmt::remove_trailing_spaces (
            std::string * str )  [noexcept]
```

Remove all trailing spaces from a string.

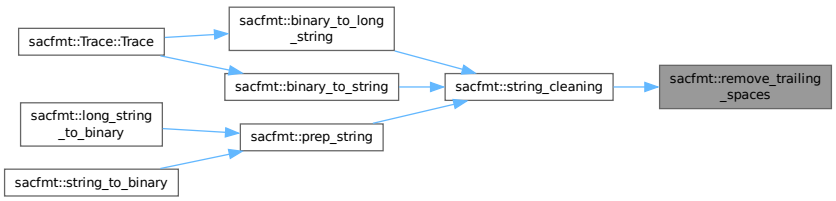
This edits the string in-place.

Parameters

in,out	*str*	std::string* String to have spaces removed.

```
00187                                                    {
00188   while ((static_cast<int>(str->back()) <= ascii_space) && (!str->empty())) {
00189     str->pop_back();
00190   }
00191 }
```

Here is the caller graph for this function:



10.1.4.39 safe_to_finish_reading()

```
void sacfmt::safe_to_finish_reading (
            std::ifstream * sac )
```

Determines if the SAC-file is finished.

This must run after reading the header, data vector(s), and footer (if applicable). This checks to ensure there is no additional data in the SAC-file (there shouldn't be, and out of safety it throws an io_error to inform the user if there are shenanigans).

Parameters

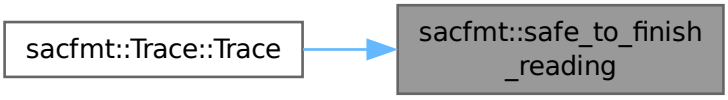
in	*sac*	std::ifstream* SAC-file to be checked.

Exceptions

io_error	If the file is not finished.

```
01720                                                {
01721   const std::streamoff current_pos{sac->tellg()};
01722   sac->seekg(0, std::ios::end);
01723   const std::streamoff end_pos{sac->tellg()};
01724   sac->seekg(current_pos, std::ios::beg);
01725   // How far are we from the end of the file?
01726   const std::streamoff diff{end_pos - current_pos};
01727   // If there is more, something weird happened...
01728   if (diff != 0) {
01729     std::ostringstream oss{};
01730     oss « "Filesize exceeds data specification with ";
01731     oss « diff;
01732     oss « " bytes excess. Data corruption suspected.";
01733     throw io_error(oss.str());
01734   }
01735 }
```

Here is the caller graph for this function:



10.1.4.40 safe_to_read_data()

```
void sacfmt::safe_to_read_data (
            std::ifstream * sac,
            const size_t n_words,
            const bool data2 )
```

Determines if the SAC-file has enough space remaining to contain a complete data vector.

This must be run after reading the header (and first data vector if applicable) and before the footer (if applicable).

Parameters

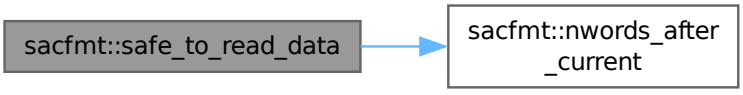
in	*sac*	std::ifstream* SAC-file to read.
in	*n_words*	Number of values in data vector.
in	*data2*	bool True if reading data2, false (default) if reading data1.

Exceptions

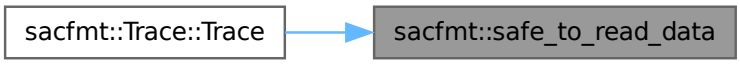
io_error	If unsafe to read.

```
01701                                       {
01702   const std::string data{data2 ? "data2" : "data1"};
01703   const read_spec spec{n_words, static_cast<size_t>(sac->tellg())};
01704   if (!nwords_after_current(sac, spec)) {
01705     throw io_error("Insufficient filesize for " + data + '.');
01706   }
01707 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.41 safe_to_read_footer()

```
void sacfmt::safe_to_read_footer (
            std::ifstream * sac )
```

Determines if the SAC-file has enough space remaining to contain a complete footer.

This must be run after reading the header and data vector(s), not before.

Parameters

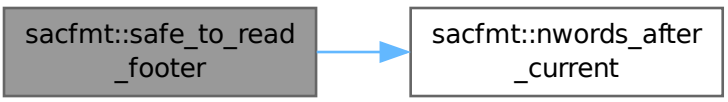
in	*sac*	std::ifstream* SAC-file to read.

Exceptions

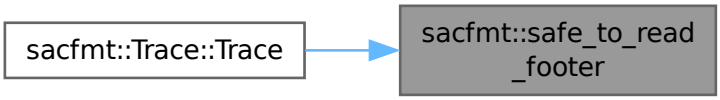
io_error	If unsafe to read.

```
01679                                          {
01680   // doubles are two words long
01681   const read_spec spec{static_cast<size_t>(num_footer) * 2,
01682                        static_cast<size_t>(sac->tellg())};
01683   if (!nwords_after_current(sac, spec)) {
01684     throw io_error("Insufficient filesize for footer.");
01685   }
01686 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.42 safe_to_read_header()

```
void sacfmt::safe_to_read_header (
            std::ifstream * sac )
```

Determine if the SAC-file is large enough to contain a complete header.

This must be run prior to reading the data vector(s) and footer (if applicable), not after.

Parameters

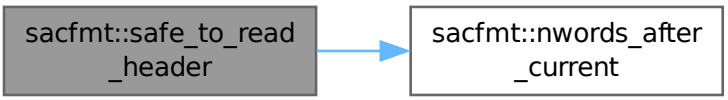
in	*sac*	std::ifstream* SAC-file to read.

Exceptions

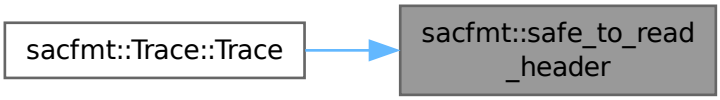
io_error	If unsafe to read.

```
01663                                          {
01664   const read_spec spec{data_word, 0};
01665   if (!nwords_after_current(sac, spec)) {
01666     throw io_error("Insufficient filesize for header.");
01667   }
01668 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.43 string_bits()

```
template<typename T >
void sacfmt::string_bits (
            T * bits,
            const std::string & str,
            const size_t str_size )  [noexcept]
```

Template function to convert string into binary bitset.

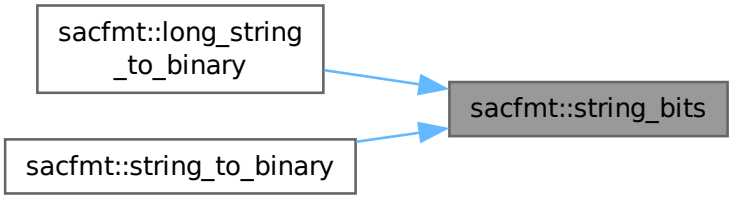
Note that this edits the bitset in place.

Parameters

out	*bits*	Destintation bitset for the string (result).
in	*str*	String to undergo conversion.
in	*str_size*	Desired string size in words (4 chars = 1 word).

```
00238                                                {
00239   constexpr size_t char_size{bits_per_byte};
00240   char_bit byte{};
00241   for (size_t i{0}; i < str_size; ++i) {
00242     size_t character{static_cast<size_t>(str[i])};
00243     byte = char_bit(character);
00244     for (size_t j{0}; j < char_size; ++j) {
00245       (*bits)[(i * char_size) + j] = byte[j];
00246     }
00247   }
00248 }
```

Here is the caller graph for this function:



10.1.4.44 string_cleaning()

```
std::string sacfmt::string_cleaning (
            const std::string & str )  [noexcept]
```

Remove leading/trailing spaces and control characters from a string.

Parameters

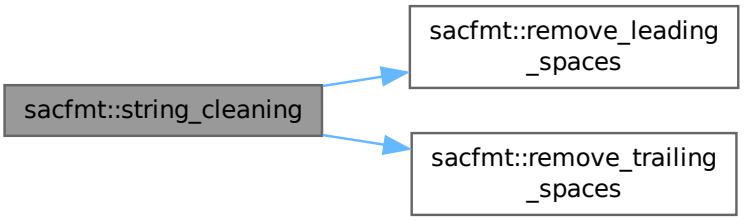
in	*str*	std::string String to be cleaned.

Returns

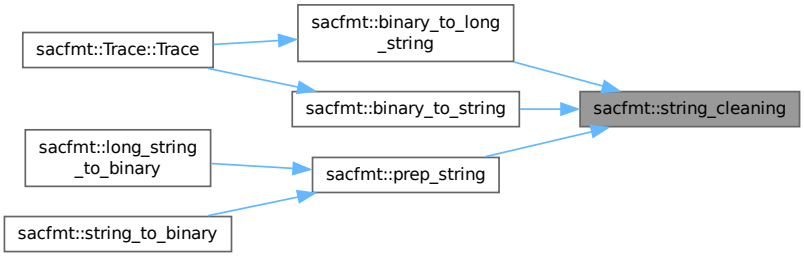
std::string Cleaned string.

```
00199                                                          {
00200   std::string result{str};
00201   size_t null_position{str.find('\0')};
00202   if (null_position != std::string::npos) {
00203     result.erase(null_position);
00204   }
00205   remove_leading_spaces(&result);
00206   remove_trailing_spaces(&result);
00207   return result;
00208 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.4.45 string_to_binary()

```
word_two sacfmt::string_to_binary (
            std::string str )  [noexcept]
```

Convert string to a 64-bit (two word) binary bitset.

If the string is longer than 8 characters, then only the first 8 characters are kept. If the string is less than 8 characters long, it is right-padded with spaces.

Parameters

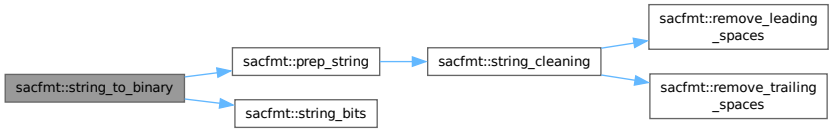
in	*str*	String to be converted to a bitset.

Returns

word_two Converted binary bitset.

```
00282                                                {
00283   constexpr size_t string_size{2 * word_length};
00284   // 1 byte per character
00285   prep_string(&str, string_size);
00286   // Two words (8 characters)
00287   word_two bits{};
00288   string_bits(&bits, str, string_size);
00289   return bits;
00290 }
```

Here is the call graph for this function:



10.1.4.46 uint_to_binary()

```
word_one sacfmt::uint_to_binary (
            uint num )  [noexcept]
```

Convert unsigned integer to 32-bit (one word) binary bitset.

This sets the current bit using bitwise and, updates the bit to manipulate and performs a right-shift (division by 2) until the number is zero.

Parameters

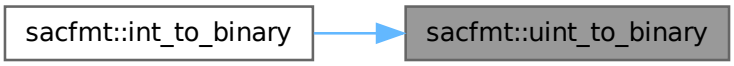
in	*num*	Number to be converted.

Returns

word_one Converted value.

```
00044                                           {
00045   word_one bits{};
00046   for (size_t pos{0}; pos < bits.size(); ++pos) {
00047     if (num > 0) {
00048       // Bitwise and to set flag.
00049       bits.set(pos, static_cast<bool>(num & 1));
00050       // Right-shift bits by 1, same as division by 2
00051       num >>= 1;
00052     } else {
00053       break;
00054     }
00055   }
00056   return bits;
00057 }
```

Here is the caller graph for this function:



10.1.4.47 word_position()

```
std::streamoff sacfmt::word_position (
            const size_t word_number )  [noexcept]
```

Calculates position of word in SAC-file.

Multiplies given word number by the word-length in bytes (defined by the SAC format.)

Parameters

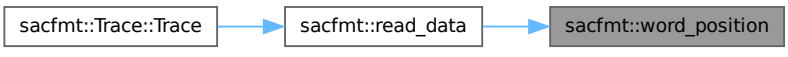
in	*word_number*	Number of desired word in file stream.

Returns

std::streamoff Position in SAC-file of desired word (in bytes).

```
00031                                                                {
00032   return static_cast<std::streamoff>(word_number * word_length);
00033 }
```

Here is the caller graph for this function:



10.1.4.48 write_words()

```
void sacfmt::write_words (
            std::ofstream * sac_file,
            const std::vector< char > & input )
```

Write arbitrary number of words (useful for vectors) to a binary SAC-file.

Note that this modifies the position of the writer within the stream (to the end of the written words).

Parameters

in,out	*sac_file*	std::ofstream* Output binary SAC-file.
in	*input*	std::vector<char> Character vector representation of data for writing.

```
00510                                                                   {
00511   std::ofstream &sac = *sac_file;
00512   if (sac.is_open()) {
00513     for (char character : input) [[likely]] {
00514       sac.write(&character, sizeof(char));
00515     }
00516   }
00517 }
```

Here is the caller graph for this function:

10.1.5 Variable Documentation

10.1.5.1 ascii_space

```
constexpr int sacfmt::ascii_space {32}  [constexpr]
```

ASCII-code of 'space' character.

```
00090 {32};
```

10.1.5.2 binary_word_size

```
constexpr size_t sacfmt::binary_word_size {word_length * bits_per_byte}  [constexpr]
```

Size (bits) of funamental data-chunk.

```
00066 {word_length * bits_per_byte};
```

10.1.5.3 bits_per_byte

```
constexpr size_t sacfmt::bits_per_byte {8}  [constexpr]
```

Size (bits) of binary character.

```
00064 {8};
```

10.1.5.4 circle_deg

```
constexpr double sacfmt::circle_deg {360.0}  [constexpr]
```

Degrees in a circle.
00116 {360.0};

10.1.5.5 common_skip_num

```
constexpr int sacfmt::common_skip_num {7}  [constexpr]
```

Extremely common number of 'internal use' headers in SAC format.
00110 {7};

10.1.5.6 data_word

```
constexpr std::streamoff sacfmt::data_word {158}  [constexpr]
```

First word of (first) data-section (stream offset).
00068 {158};

10.1.5.7 deg_per_rad

```
constexpr double sacfmt::deg_per_rad {1.0 / rad_per_deg}  [constexpr]
```

Degrees per radian.
00114 {1.0 / rad_per_deg};

10.1.5.8 earth_radius

```
constexpr double sacfmt::earth_radius {6378.14}  [constexpr]
```

Average radius of Earth (kilometers).
00118 {6378.14};

10.1.5.9 f_eps

```
constexpr float sacfmt::f_eps {2.75e-6F}  [constexpr]
```

Accuracy precision expected of SAC floating-point values.
00080 {2.75e-6F};

10.1.5.10 modern_hdr_version

```
constexpr int sacfmt::modern_hdr_version {7}  [constexpr]
```

nVHdr value for newest SAC format (2020+).
00106 {7};

10.1.5.11 num_bool

```
constexpr int sacfmt::num_bool {4}  [constexpr]
```

Number of boolean header values in SAC format.
```
00098 {4};
```

10.1.5.12 num_data

```
constexpr int sacfmt::num_data {2}  [constexpr]
```

Number of data arrays in SAC format.
```
00102 {2};
```

10.1.5.13 num_double

```
constexpr int sacfmt::num_double {22}  [constexpr]
```

Number of double-precision header values in SAC format.
```
00094 {22};
```

10.1.5.14 num_float

```
constexpr int sacfmt::num_float {39}  [constexpr]
```

Number of float-poing header values in SAC format.
```
00092 {39};
```

10.1.5.15 num_footer

```
constexpr int sacfmt::num_footer {22}  [constexpr]
```

Number of double-precision footer values in SAC format (version 7).
```
00104 {22};
```

10.1.5.16 num_int

```
constexpr int sacfmt::num_int {26}  [constexpr]
```

Number of integer header values in SAC format.
```
00096 {26};
```

10.1.5.17 num_string

```
constexpr int sacfmt::num_string {23}  [constexpr]
```

Number of string header values in SAC format.
```
00100 {23};
```

10.1.5.18 old_hdr_version

```
constexpr int sacfmt::old_hdr_version {6}  [constexpr]
```

nVHdr value for historic SAC format (pre-2020).

```
00108 {6};
```

10.1.5.19 rad_per_deg

```
constexpr double sacfmt::rad_per_deg {std::numbers::pi_v<double> / 180.0}  [constexpr]
```

Radians per degree.

```
00112 {std::numbers::pi_v<double> / 180.0};
```

10.1.5.20 sac_map

```
const std::unordered_map<name, const size_t> sacfmt::sac_map
```

Lookup table for variable locations.

Maps SAC variables (headers and data) to their internal locations in the Trace class.

```
00910                                                 {
00911     // Floats
00912     {name::depmin, 0},
00913     {name::depmax, 1},
00914     {name::odelta, 2},
00915     {name::resp0, 3},
00916     {name::resp1, 4},
00917     {name::resp2, 5},
00918     {name::resp3, 6},
00919     {name::resp4, 7},
00920     {name::resp5, 8},
00921     {name::resp6, 9},
00922     {name::resp7, 10},
00923     {name::resp8, 11},
00924     {name::resp9, 12},
00925     {name::stel, 13},
00926     {name::stdp, 14},
00927     {name::evel, 15},
00928     {name::evdp, 16},
00929     {name::mag, 17},
00930     {name::user0, 18},
00931     {name::user1, 19},
00932     {name::user2, 20},
00933     {name::user3, 21},
00934     {name::user4, 22},
00935     {name::user5, 23},
00936     {name::user6, 24},
00937     {name::user7, 25},
00938     {name::user8, 26},
00939     {name::user9, 27},
00940     {name::dist, 28},
00941     {name::az, 29},
00942     {name::baz, 30},
00943     {name::gcarc, 31},
00944     {name::depmen, 32},
00945     {name::cmpaz, 33},
00946     {name::cmpinc, 34},
00947     {name::xminimum, 35},
00948     {name::xmaximum, 36},
00949     {name::yminimum, 37},
00950     {name::ymaximum, 38},
00951     // Doubles
00952     {name::delta, 0},
00953     {name::b, 1},
00954     {name::e, 2},
00955     {name::o, 3},
00956     {name::a, 4},
00957     {name::t0, 5},
00958     {name::t1, 6},
00959     {name::t2, 7},
00960     {name::t3, 8},
```

```
00961     {name::t4, 9},
00962     {name::t5, 10},
00963     {name::t6, 11},
00964     {name::t7, 12},
00965     {name::t8, 13},
00966     {name::t9, 14},
00967     {name::f, 15},
00968     {name::stla, 16},
00969     {name::stlo, 17},
00970     {name::evla, 18},
00971     {name::evlo, 19},
00972     {name::sb, 20},
00973     {name::sdelta, 21},
00974     // Ints
00975     {name::nzyear, 0},
00976     {name::nzjday, 1},
00977     {name::nzhour, 2},
00978     {name::nzmin, 3},
00979     {name::nzsec, 4},
00980     {name::nzmsec, 5},
00981     {name::nvhdr, 6},
00982     {name::norid, 7},
00983     {name::nevid, 8},
00984     {name::npts, 9},
00985     {name::nsnpts, 10},
00986     {name::nwfid, 11},
00987     {name::nxsize, 12},
00988     {name::nysize, 13},
00989     {name::iftype, 14},
00990     {name::idep, 15},
00991     {name::iztype, 16},
00992     {name::iinst, 17},
00993     {name::istreg, 18},
00994     {name::ievreg, 19},
00995     {name::ievtyp, 20},
00996     {name::iqual, 21},
00997     {name::isynth, 22},
00998     {name::imagtyp, 23},
00999     {name::imagsrc, 24},
01000     {name::ibody, 25},
01001     // Bools
01002     {name::leven, 0},
01003     {name::lpspol, 1},
01004     {name::lovrok, 2},
01005     {name::lcalda, 3},
01006     // Strings
01007     {name::kstnm, 0},
01008     {name::kevnm, 1},
01009     {name::khole, 2},
01010     {name::ko, 3},
01011     {name::ka, 4},
01012     {name::kt0, 5},
01013     {name::kt1, 6},
01014     {name::kt2, 7},
01015     {name::kt3, 8},
01016     {name::kt4, 9},
01017     {name::kt5, 10},
01018     {name::kt6, 11},
01019     {name::kt7, 12},
01020     {name::kt8, 13},
01021     {name::kt9, 14},
01022     {name::kf, 15},
01023     {name::kuser0, 16},
01024     {name::kuser1, 17},
01025     {name::kuser2, 18},
01026     {name::kcmpnm, 19},
01027     {name::knetwk, 20},
01028     {name::kdatrd, 21},
01029     {name::kinst, 22},
01030     // Data
01031     {name::data1, 0},
01032     {name::data2, 1}};
```

10.1.5.21 unset_bool

```
constexpr bool sacfmt::unset_bool {false}  [constexpr]
```

Boolean unset value (SAC Magic).

```
00076 {false};
```

10.1.5.22 unset_double

```
constexpr double sacfmt::unset_double {-12345.0}  [constexpr]
```

Double-precision unset value (SAC Magic).
00074 {-12345.0};

10.1.5.23 unset_float

```
constexpr float sacfmt::unset_float {-12345.0F}  [constexpr]
```

Float-point unset value (SAC Magic).
00072 {-12345.0F};

10.1.5.24 unset_int

```
constexpr int sacfmt::unset_int {-12345}  [constexpr]
```

Integer unset value (SAC Magic).
00070 {-12345};

10.1.5.25 unset_word

```
const std::string sacfmt::unset_word {"-12345"}
```

String unset value (SAC Magic).
00078 {"-12345"};

10.1.5.26 word_length

```
constexpr size_t sacfmt::word_length {4}  [constexpr]
```

Size (bytes) of fundamental data-chunk.
00062 {4};

10.2 sacfmt::bitset_type Namespace Reference

bitset type-safety namespace.

Classes

- struct uint

  Ensure type-safety for conversions between floats/doubles and bitsets.
- struct uint< 4 ∗bits_per_byte >

  One-word (floats).
- struct uint< bytes ∗bits_per_byte >

  Two-words (doubles)

Variables

- constexpr int bytes {8}

10.2.1 Detailed Description

bitset type-safety namespace.

10.2.2 Variable Documentation

10.2.2.1 bytes

```
constexpr int sacfmt::bitset_type::bytes {8}  [constexpr]
00138 {8};
```

Chapter 11

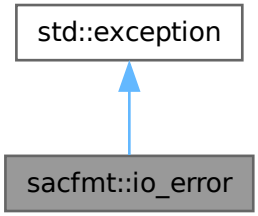
Class Documentation

11.1 sacfmt::io_error Class Reference

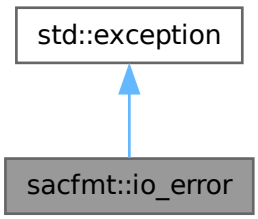
Class for generic I/O exceptions.

`#include <sac_format.hpp>`

Inheritance diagram for sacfmt::io_error:



Collaboration diagram for sacfmt::io_error:



Public Member Functions

- io_error (std::string msg)
  io_error Constructor
- const char * what () const noexcept override
  Error message delivery.

Private Attributes

- const std::string message {}
  Error message.

11.1.1 Detailed Description

Class for generic I/O exceptions.

These errors occur due to bad path, bad permissions, or otherwise corrupt SAC-files.

I/O operations may raise other exceptions (disk failure, out of space, etc.), but those are difficult to emulate for testing purposes (therefore I am unable to reliably cover them); they also arise due to conditions that would render how sac-format handles them moot.

11.1.2 Constructor & Destructor Documentation

11.1.2.1 io_error()

```
sacfmt::io_error::io_error (
            std::string msg )  [inline], [explicit]
```

io_error Constructor

Parameters

in	*msg*	std::string Error message.

```
01352 : message(std::move(msg)) {}
```

11.1.3 Member Function Documentation

11.1.3.1 what()

```
const char * sacfmt::io_error::what ( ) const  [inline], [override], [noexcept]
```

Error message delivery.

Returns

what char* Error message.

```
01358                                                {
01359     return message.c_str();
01360   }
```

11.1.4 Member Data Documentation

11.1.4.1 message

```
const std::string sacfmt::io_error::message {}  [private]
```

Error message.
01344 {};

The documentation for this class was generated from the following file:

- include/sac-format/sac_format.hpp

11.2 sacfmt::read_spec Struct Reference

Struct that specifies parameters for reading.

`#include <sac_format.hpp>`

Public Attributes

- size_t num_words {}
  Number of words to read.
- size_t start_word {}
  Word to start reading from.

11.2.1 Detailed Description

Struct that specifies parameters for reading.

Prevents bug-prone number-swapping in functions that use a reading specification.

11.2.2 Member Data Documentation

11.2.2.1 num_words

```
size_t sacfmt::read_spec::num_words {}
```

Number of words to read.
00211 {};

11.2.2.2 start_word

```
size_t sacfmt::read_spec::start_word {}
```

Word to start reading from.
00213 {};

The documentation for this struct was generated from the following file:

- include/sac-format/sac_format.hpp

11.3 sacfmt::Trace Class Reference

The Trace class.

`#include <sac_format.hpp>`

Public Member Functions

- Trace () noexcept
  Trace default constructor.
- Trace (const std::filesystem::path &path)
  Binary SAC-file reader.
- void write (const std::filesystem::path &path, bool legacy=false) const
  Binary SAC-file writer.
- void legacy_write (const std::filesystem::path &path) const
  Binary SAC-file legacy-write convenience function.
- bool operator== (const Trace &other) const noexcept
  Trace equality operator.
- void calc_geometry () noexcept
  Calculates gcarc, dist, az, and baz from stla, stlo, evla, and evlo.
- double frequency () const noexcept
  Calculate frequency from delta.
- std::string date () const noexcept
  Get date string.
- std::string time () const noexcept
  Get time string.
- float depmin () const noexcept
- float depmax () const noexcept
- float odelta () const noexcept
- float resp0 () const noexcept
- float resp1 () const noexcept
- float resp2 () const noexcept
- float resp3 () const noexcept
- float resp4 () const noexcept
- float resp5 () const noexcept
- float resp6 () const noexcept
- float resp7 () const noexcept
- float resp8 () const noexcept
- float resp9 () const noexcept
- float stel () const noexcept
- float stdp () const noexcept
- float evel () const noexcept
- float evdp () const noexcept
- float mag () const noexcept
- float user0 () const noexcept
- float user1 () const noexcept
- float user2 () const noexcept
- float user3 () const noexcept
- float user4 () const noexcept
- float user5 () const noexcept
- float user6 () const noexcept
- float user7 () const noexcept

- float user8 () const noexcept
- float user9 () const noexcept
- float dist () const noexcept
- float az () const noexcept
- float baz () const noexcept
- float gcarc () const noexcept
- float depmen () const noexcept
- float cmpaz () const noexcept
- float cmpinc () const noexcept
- float xminimum () const noexcept
- float xmaximum () const noexcept
- float yminimum () const noexcept
- float ymaximum () const noexcept
- double delta () const noexcept
- double b () const noexcept
- double e () const noexcept
- double o () const noexcept
- double a () const noexcept
- double t0 () const noexcept
- double t1 () const noexcept
- double t2 () const noexcept
- double t3 () const noexcept
- double t4 () const noexcept
- double t5 () const noexcept
- double t6 () const noexcept
- double t7 () const noexcept
- double t8 () const noexcept
- double t9 () const noexcept
- double f () const noexcept
- double stla () const noexcept
- double stlo () const noexcept
- double evla () const noexcept
- double evlo () const noexcept
- double sb () const noexcept
- double sdelta () const noexcept
- int nzyear () const noexcept
- int nzjday () const noexcept
- int nzhour () const noexcept
- int nzmin () const noexcept
- int nzsec () const noexcept
- int nzmsec () const noexcept
- int nvhdr () const noexcept
- int norid () const noexcept
- int nevid () const noexcept
- int npts () const noexcept
- int nsnpts () const noexcept
- int nwfid () const noexcept
- int nxsize () const noexcept
- int nysize () const noexcept
- int iftype () const noexcept
- int idep () const noexcept
- int iztype () const noexcept
- int iinst () const noexcept
- int istreg () const noexcept
- int ievreg () const noexcept

- int ievtyp () const noexcept
- int iqual () const noexcept
- int isynth () const noexcept
- int imagtyp () const noexcept
- int imagsrc () const noexcept
- int ibody () const noexcept
- bool leven () const noexcept
- bool lpspol () const noexcept
- bool lovrok () const noexcept
- bool lcalda () const noexcept
- std::string kstnm () const noexcept
- std::string kevnm () const noexcept
- std::string khole () const noexcept
- std::string ko () const noexcept
- std::string ka () const noexcept
- std::string kt0 () const noexcept
- std::string kt1 () const noexcept
- std::string kt2 () const noexcept
- std::string kt3 () const noexcept
- std::string kt4 () const noexcept
- std::string kt5 () const noexcept
- std::string kt6 () const noexcept
- std::string kt7 () const noexcept
- std::string kt8 () const noexcept
- std::string kt9 () const noexcept
- std::string kf () const noexcept
- std::string kuser0 () const noexcept
- std::string kuser1 () const noexcept
- std::string kuser2 () const noexcept
- std::string kcmpnm () const noexcept
- std::string knetwk () const noexcept
- std::string kdatrd () const noexcept
- std::string kinst () const noexcept
- std::vector< double > data1 () const noexcept
- std::vector< double > data2 () const noexcept
- void depmin (float input) noexcept
- void depmax (float input) noexcept
- void odelta (float input) noexcept
- void resp0 (float input) noexcept
- void resp1 (float input) noexcept
- void resp2 (float input) noexcept
- void resp3 (float input) noexcept
- void resp4 (float input) noexcept
- void resp5 (float input) noexcept
- void resp6 (float input) noexcept
- void resp7 (float input) noexcept
- void resp8 (float input) noexcept
- void resp9 (float input) noexcept
- void stel (float input) noexcept
- void stdp (float input) noexcept
- void evel (float input) noexcept
- void evdp (float input) noexcept
- void mag (float input) noexcept
- void user0 (float input) noexcept
- void user1 (float input) noexcept

- void user2 (float input) noexcept
- void user3 (float input) noexcept
- void user4 (float input) noexcept
- void user5 (float input) noexcept
- void user6 (float input) noexcept
- void user7 (float input) noexcept
- void user8 (float input) noexcept
- void user9 (float input) noexcept
- void dist (float input) noexcept
- void az (float input) noexcept
- void baz (float input) noexcept
- void gcarc (float input) noexcept
- void depmen (float input) noexcept
- void cmpaz (float input) noexcept
- void cmpinc (float input) noexcept
- void xminimum (float input) noexcept
- void xmaximum (float input) noexcept
- void yminimum (float input) noexcept
- void ymaximum (float input) noexcept
- void delta (double input) noexcept
- void b (double input) noexcept
- void e (double input) noexcept
- void o (double input) noexcept
- void a (double input) noexcept
- void t0 (double input) noexcept
- void t1 (double input) noexcept
- void t2 (double input) noexcept
- void t3 (double input) noexcept
- void t4 (double input) noexcept
- void t5 (double input) noexcept
- void t6 (double input) noexcept
- void t7 (double input) noexcept
- void t8 (double input) noexcept
- void t9 (double input) noexcept
- void f (double input) noexcept
- void stla (double input) noexcept
- void stlo (double input) noexcept
- void evla (double input) noexcept
- void evlo (double input) noexcept
- void sb (double input) noexcept
- void sdelta (double input) noexcept
- void nzyear (int input) noexcept
- void nzjday (int input) noexcept
- void nzhour (int input) noexcept
- void nzmin (int input) noexcept
- void nzsec (int input) noexcept
- void nzmsec (int input) noexcept
- void nvhdr (int input) noexcept
- void norid (int input) noexcept
- void nevid (int input) noexcept
- void npts (int input) noexcept
- void nsnpts (int input) noexcept
- void nwfid (int input) noexcept
- void nxsize (int input) noexcept
- void nysize (int input) noexcept

- void iftype (int input) noexcept
- void idep (int input) noexcept
- void iztype (int input) noexcept
- void iinst (int input) noexcept
- void istreg (int input) noexcept
- void ievreg (int input) noexcept
- void ievtyp (int input) noexcept
- void iqual (int input) noexcept
- void isynth (int input) noexcept
- void imagtyp (int input) noexcept
- void imagsrc (int input) noexcept
- void ibody (int input) noexcept
- void leven (bool input) noexcept
- void lpspol (bool input) noexcept
- void lovrok (bool input) noexcept
- void lcalda (bool input) noexcept
- void kstnm (const std::string &input) noexcept
- void kevnm (const std::string &input) noexcept
- void khole (const std::string &input) noexcept
- void ko (const std::string &input) noexcept
- void ka (const std::string &input) noexcept
- void kt0 (const std::string &input) noexcept
- void kt1 (const std::string &input) noexcept
- void kt2 (const std::string &input) noexcept
- void kt3 (const std::string &input) noexcept
- void kt4 (const std::string &input) noexcept
- void kt5 (const std::string &input) noexcept
- void kt6 (const std::string &input) noexcept
- void kt7 (const std::string &input) noexcept
- void kt8 (const std::string &input) noexcept
- void kt9 (const std::string &input) noexcept
- void kf (const std::string &input) noexcept
- void kuser0 (const std::string &input) noexcept
- void kuser1 (const std::string &input) noexcept
- void kuser2 (const std::string &input) noexcept
- void kcmpnm (const std::string &input) noexcept
- void knetwk (const std::string &input) noexcept
- void kdatrd (const std::string &input) noexcept
- void kinst (const std::string &input) noexcept
- void data1 (const std::vector< double > &input) noexcept
- void data2 (const std::vector< double > &input) noexcept

Private Member Functions

- void calc_gcarc () noexcept

  Calculate great-circle arc-distance (gcarc).
- void calc_dist () noexcept

  Calculate distance (using gcarc).
- void calc_az () noexcept

  Calculate azimuth.
- void calc_baz () noexcept

  Calculate back-azimuth.
- bool geometry_set () const noexcept

Determine if locations are set for geometry calculation.

- void resize_data1 (size_t size) noexcept
- void resize_data2 (size_t size) noexcept
- void resize_data (size_t size) noexcept

  Resize data vectors (only if eligible).

Private Attributes

- std::array< float, num_float > floats {}

  Float storage array.
- std::array< double, num_double > doubles {}

  Double storage array.
- std::array< int, num_int > ints {}

  Integer storage array.
- std::array< bool, num_bool > bools {}

  Boolean storage array.
- std::array< std::string, num_string > strings {}

  String storage array.
- std::array< std::vector< double >, num_data > data {}

  std::vector<double> storage array.

11.3.1 Detailed Description

The Trace class.

This class is the recommended way for reading/writing SAC-files.

It safely reads all data, provides automatic write support based upon the nVHdr header value (determine if a footer should be included or not).

It provides getters and setters for all SAC headers and the data.

11.3.2 Constructor & Destructor Documentation

11.3.2.1 Trace() [1/2]

```
sacfmt::Trace::Trace ( )  [noexcept]
```

Trace default constructor.

Fills all values with their default (unset) values. Data vectors are of size zero.

Returns

Default created Trace object.

```
00832                       {
00833   std::ranges::fill(floats.begin(), floats.end(), unset_float);
00834   std::ranges::fill(doubles.begin(), doubles.end(), unset_double);
00835   std::ranges::fill(ints.begin(), ints.end(), unset_int);
00836   std::ranges::fill(bools.begin(), bools.end(), unset_bool);
00837   std::ranges::fill(strings.begin(), strings.end(), unset_word);
00838 }
```

11.3.2.2 Trace() [2/2]

```
sacfmt::Trace::Trace (
            const std::filesystem::path & path )  [explicit]
```

Binary SAC-file reader.

Parameters

in	*path*	std::filesystem::path SAC-file to be read.

Returns

Trace read in-file.

Exceptions

io_error	If the file is not safe to read for whatever reason.
std::exception	(disk failure).

```
01745                                           {
01746   std::ifstream file(path, std::ifstream::binary);
01747   if (!file) {
01748     throw io_error(path.string() + " cannot be opened to read.");
01749   }
01750   safe_to_read_header(&file);  // throws io_error if not safe
01751   //--------------------------------------------------------------------------
01752   // Header
01753   delta(binary_to_float(read_word(&file)));
01754   depmin(binary_to_float(read_word(&file)));
01755   depmax(binary_to_float(read_word(&file)));
01756   // Skip 'unused'
01757   read_word(&file);
01758   odelta(binary_to_float(read_word(&file)));
01759   b(binary_to_float(read_word(&file)));
01760   e(binary_to_float(read_word(&file)));
01761   o(binary_to_float(read_word(&file)));
01762   a(binary_to_float(read_word(&file)));
01763   // Skip 'internal'
01764   read_word(&file);
01765   // T# pick headers
01766   t0(binary_to_float(read_word(&file)));
01767   t1(binary_to_float(read_word(&file)));
01768   t2(binary_to_float(read_word(&file)));
01769   t3(binary_to_float(read_word(&file)));
01770   t4(binary_to_float(read_word(&file)));
01771   t5(binary_to_float(read_word(&file)));
01772   t6(binary_to_float(read_word(&file)));
01773   t7(binary_to_float(read_word(&file)));
01774   t8(binary_to_float(read_word(&file)));
01775   t9(binary_to_float(read_word(&file)));
01776   f(binary_to_float(read_word(&file)));
01777   // Response headers
01778   resp0(binary_to_float(read_word(&file)));
01779   resp1(binary_to_float(read_word(&file)));
01780   resp2(binary_to_float(read_word(&file)));
01781   resp3(binary_to_float(read_word(&file)));
01782   resp4(binary_to_float(read_word(&file)));
01783   resp5(binary_to_float(read_word(&file)));
01784   resp6(binary_to_float(read_word(&file)));
01785   resp7(binary_to_float(read_word(&file)));
01786   resp8(binary_to_float(read_word(&file)));
01787   resp9(binary_to_float(read_word(&file)));
01788   // Station headers
01789   stla(binary_to_float(read_word(&file)));
01790   stlo(binary_to_float(read_word(&file)));
01791   stel(binary_to_float(read_word(&file)));
01792   stdp(binary_to_float(read_word(&file)));
01793   // Event headers
01794   evla(binary_to_float(read_word(&file)));
01795   evlo(binary_to_float(read_word(&file)));
01796   evel(binary_to_float(read_word(&file)));
01797   evdp(binary_to_float(read_word(&file)));
01798   mag(binary_to_float(read_word(&file)));
01799   // User misc headers
01800   user0(binary_to_float(read_word(&file)));
01801   user1(binary_to_float(read_word(&file)));
01802   user2(binary_to_float(read_word(&file)));
01803   user3(binary_to_float(read_word(&file)));
01804   user4(binary_to_float(read_word(&file)));
01805   user5(binary_to_float(read_word(&file)));
01806   user6(binary_to_float(read_word(&file)));
01807   user7(binary_to_float(read_word(&file)));
01808   user8(binary_to_float(read_word(&file)));
01809   user9(binary_to_float(read_word(&file)));
```

```
01810   // Geometry headers
01811   dist(binary_to_float(read_word(&file)));
01812   az(binary_to_float(read_word(&file)));
01813   baz(binary_to_float(read_word(&file)));
01814   gcarc(binary_to_float(read_word(&file)));
01815   // Metadata headers
01816   sb(binary_to_float(read_word(&file)));
01817   sdelta(binary_to_float(read_word(&file)));
01818   depmen(binary_to_float(read_word(&file)));
01819   cmpaz(binary_to_float(read_word(&file)));
01820   cmpinc(binary_to_float(read_word(&file)));
01821   xminimum(binary_to_float(read_word(&file)));
01822   xmaximum(binary_to_float(read_word(&file)));
01823   yminimum(binary_to_float(read_word(&file)));
01824   ymaximum(binary_to_float(read_word(&file)));
01825   // Skip 'unused' (xcommon_skip_num)
01826   for (int i{0}; i < common_skip_num; ++i) {
01827     read_word(&file);
01828   }
01829   // Date/time headers
01830   nzyear(binary_to_int(read_word(&file)));
01831   nzjday(binary_to_int(read_word(&file)));
01832   nzhour(binary_to_int(read_word(&file)));
01833   nzmin(binary_to_int(read_word(&file)));
01834   nzsec(binary_to_int(read_word(&file)));
01835   nzmsec(binary_to_int(read_word(&file)));
01836   // More metadata headers
01837   nvhdr(binary_to_int(read_word(&file)));
01838   norid(binary_to_int(read_word(&file)));
01839   nevid(binary_to_int(read_word(&file)));
01840   npts(binary_to_int(read_word(&file)));
01841   nsnpts(binary_to_int(read_word(&file)));
01842   nwfid(binary_to_int(read_word(&file)));
01843   nxsize(binary_to_int(read_word(&file)));
01844   nysize(binary_to_int(read_word(&file)));
01845   // Skip 'unused'
01846   read_word(&file);
01847   iftype(binary_to_int(read_word(&file)));
01848   idep(binary_to_int(read_word(&file)));
01849   iztype(binary_to_int(read_word(&file)));
01850   // Skip 'unused'
01851   read_word(&file);
01852   iinst(binary_to_int(read_word(&file)));
01853   istreg(binary_to_int(read_word(&file)));
01854   ievreg(binary_to_int(read_word(&file)));
01855   ievtyp(binary_to_int(read_word(&file)));
01856   iqual(binary_to_int(read_word(&file)));
01857   isynth(binary_to_int(read_word(&file)));
01858   imagtyp(binary_to_int(read_word(&file)));
01859   imagsrc(binary_to_int(read_word(&file)));
01860   ibody(binary_to_int(read_word(&file)));
01861   // Skip 'unused' (xcommon_skip_num)
01862   for (int i{0}; i < common_skip_num; ++i) {
01863     read_word(&file);
01864   }
01865   // Logical headers
01866   leven(binary_to_bool(read_word(&file)));
01867   lpspol(binary_to_bool(read_word(&file)));
01868   lovrok(binary_to_bool(read_word(&file)));
01869   lcalda(binary_to_bool(read_word(&file)));
01870   // Skip 'unused'
01871   read_word(&file);
01872   // KSTNM is 2 words (normal)
01873   kstnm(binary_to_string(read_two_words(&file)));
01874   // KEVNM is 4 words long (unique!)
01875   kevnm(binary_to_long_string(read_four_words(&file)));
01876   // All other 'K' headers are 2 words
01877   khole(binary_to_string(read_two_words(&file)));
01878   ko(binary_to_string(read_two_words(&file)));
01879   ka(binary_to_string(read_two_words(&file)));
01880   kt0(binary_to_string(read_two_words(&file)));
01881   kt1(binary_to_string(read_two_words(&file)));
01882   kt2(binary_to_string(read_two_words(&file)));
01883   kt3(binary_to_string(read_two_words(&file)));
01884   kt4(binary_to_string(read_two_words(&file)));
01885   kt5(binary_to_string(read_two_words(&file)));
01886   kt6(binary_to_string(read_two_words(&file)));
01887   kt7(binary_to_string(read_two_words(&file)));
01888   kt8(binary_to_string(read_two_words(&file)));
01889   kt9(binary_to_string(read_two_words(&file)));
01890   kf(binary_to_string(read_two_words(&file)));
01891   kuser0(binary_to_string(read_two_words(&file)));
01892   kuser1(binary_to_string(read_two_words(&file)));
01893   kuser2(binary_to_string(read_two_words(&file)));
01894   kcmpnm(binary_to_string(read_two_words(&file)));
01895   knetwk(binary_to_string(read_two_words(&file)));
01896   kdatrd(binary_to_string(read_two_words(&file)));
```

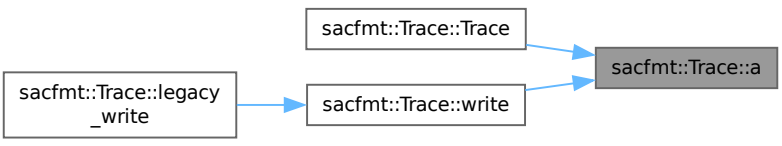
```
01897   kinst(binary_to_string(read_two_words(&file)));
01898   //------------------------------------------------------------------------
01899   // DATA
01900   const bool is_data{npts() != unset_int};
01901   // data1
01902   const size_t n_words{static_cast<size_t>(npts())};
01903   if (is_data) {
01904     // false flags for data1
01905     safe_to_read_data(&file, n_words, false);  // throws io_error if unsafe
01906     const read_spec spec{n_words, data_word};
01907     // Originally floats, read as doubles
01908     data1(read_data(&file, spec));
01909   }
01910   // data2 (uneven or spectral data)
01911   if (is_data && (!leven() || (iftype() > 1))) {
01912     // true flags for data2
01913     safe_to_read_data(&file, n_words, true);  // throws io_error if unsafe
01914     const read_spec spec{n_words, data_word + static_cast<size_t>(npts())};
01915     data2(read_data(&file, spec));
01916   }
01917   //------------------------------------------------------------------------
01918   // Footer
01919   if (nvhdr() == modern_hdr_version) {
01920     safe_to_read_footer(&file);  // throws io_error if not safe
01921     delta(binary_to_double(read_two_words(&file)));
01922     b(binary_to_double(read_two_words(&file)));
01923     e(binary_to_double(read_two_words(&file)));
01924     o(binary_to_double(read_two_words(&file)));
01925     a(binary_to_double(read_two_words(&file)));
01926     t0(binary_to_double(read_two_words(&file)));
01927     t1(binary_to_double(read_two_words(&file)));
01928     t2(binary_to_double(read_two_words(&file)));
01929     t3(binary_to_double(read_two_words(&file)));
01930     t4(binary_to_double(read_two_words(&file)));
01931     t5(binary_to_double(read_two_words(&file)));
01932     t6(binary_to_double(read_two_words(&file)));
01933     t7(binary_to_double(read_two_words(&file)));
01934     t8(binary_to_double(read_two_words(&file)));
01935     t9(binary_to_double(read_two_words(&file)));
01936     f(binary_to_double(read_two_words(&file)));
01937     evlo(binary_to_double(read_two_words(&file)));
01938     evla(binary_to_double(read_two_words(&file)));
01939     stlo(binary_to_double(read_two_words(&file)));
01940     stla(binary_to_double(read_two_words(&file)));
01941     sb(binary_to_double(read_two_words(&file)));
01942     sdelta(binary_to_double(read_two_words(&file)));
01943   }
01944   safe_to_finish_reading(&file);  // throws io_error if the file isn't finished
01945   file.close();
01946 }
```

11.3.3 Member Function Documentation

11.3.3.1 a() [1/2]

```
double sacfmt::Trace::a ( ) const  [noexcept]
01062 { return doubles[sac_map.at(name::a)]; }
```

Here is the caller graph for this function:



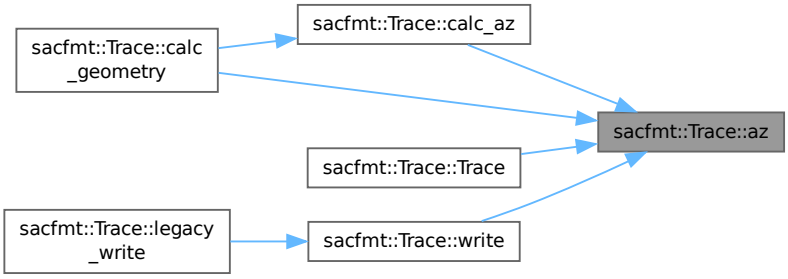
11.3.3.2 a() [2/2]

```
void sacfmt::Trace::a (
            double input )  [noexcept]
01317                                       {
01318   doubles[sac_map.at(name::a)] = input;
01319 }
```

11.3.3.3 az() [1/2]

```
float sacfmt::Trace::az ( ) const  [noexcept]
01033 { return floats[sac_map.at(name::az)]; }
```

Here is the caller graph for this function:



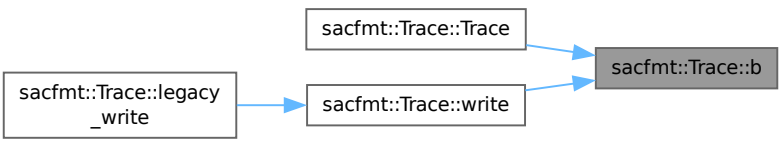
11.3.3.4 az() [2/2]

```
void sacfmt::Trace::az (
            float input )  [noexcept]
01274                                       {
01275   floats[sac_map.at(name::az)] = input;
01276 }
```

11.3.3.5 b() [1/2]

```
double sacfmt::Trace::b ( ) const  [noexcept]
01059 { return doubles[sac_map.at(name::b)]; }
```

Here is the caller graph for this function:



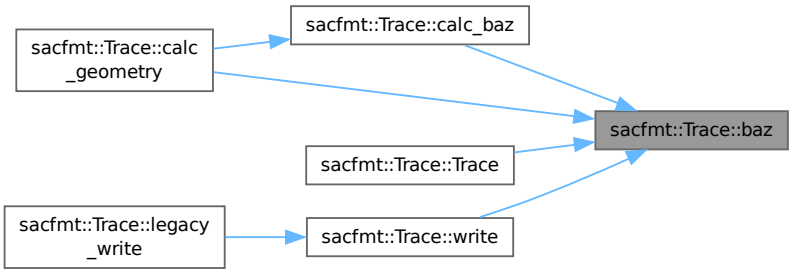
11.3.3.6 b() [2/2]

```
void sacfmt::Trace::b (
            double input )  [noexcept]
01308                                       {
01309   doubles[sac_map.at(name::b)] = input;
01310 }
```

11.3.3.7 baz() [1/2]

```
float sacfmt::Trace::baz ( ) const  [noexcept]
01034 { return floats[sac_map.at(name::baz)]; }
```

Here is the caller graph for this function:



11.3.3.8 baz() [2/2]

```
void sacfmt::Trace::baz (
            float input )  [noexcept]
01277                                       {
01278   floats[sac_map.at(name::baz)] = input;
01279 }
```

11.3.3.9 calc_az()

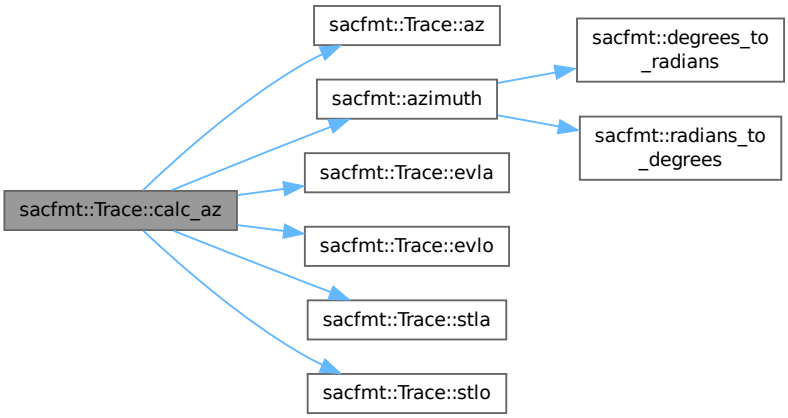
```
void sacfmt::Trace::calc_az ( )  [private], [noexcept]
```

Calculate azimuth.

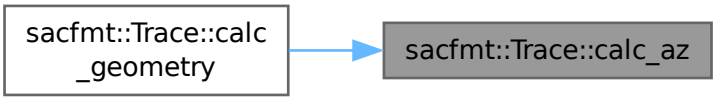
$$Station \rightarrow Event$$

```
00942                                {
00943   az(static_cast<float>(azimuth(evla(), evlo(), stla(), stlo())));
00944 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.3.3.10 calc_baz()

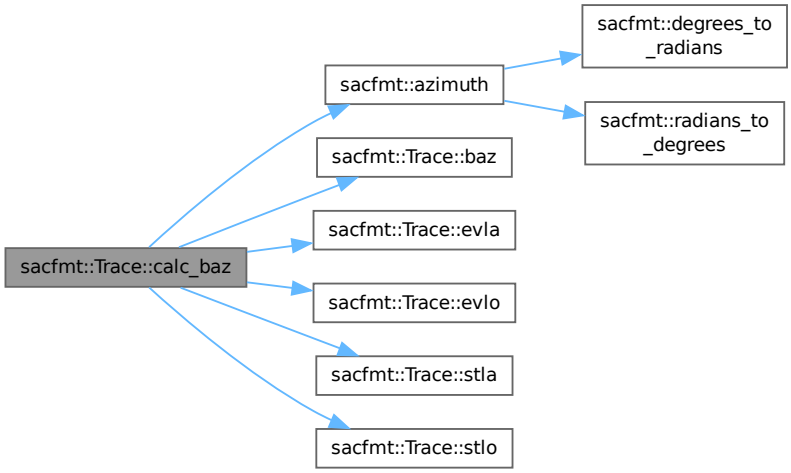
```
void sacfmt::Trace::calc_baz ( )  [private], [noexcept]
```

Calculate back-azimuth.

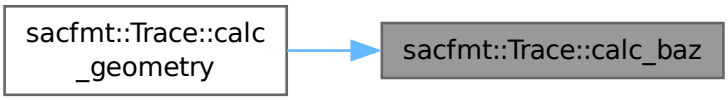
$$Event \rightarrow Station$$

```
00953                                {
00954   baz(static_cast<float>(azimuth(stla(), stlo(), evla(), evlo())));
00955 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.3.3.11 calc_dist()

```
void sacfmt::Trace::calc_dist ( )  [private], [noexcept]
```

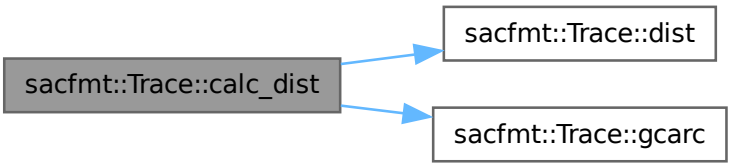
Calculate distance (using gcarc).

Assumes spherical Earth (in future may update to include flattening and different planteray bodies).

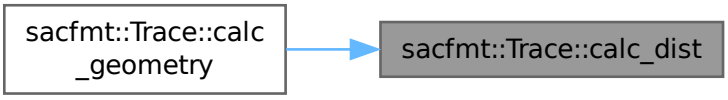
$$d = r_E \cdot \Delta$$

```
00931                                {
00932   dist(static_cast<float>(earth_radius * rad_per_deg * gcarc()));
00933 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



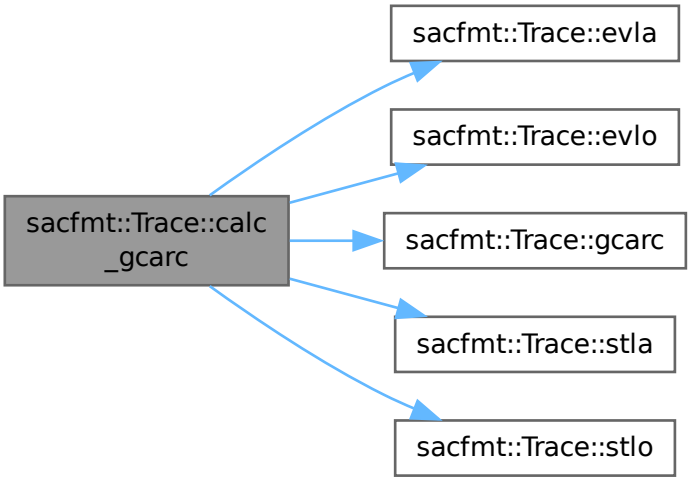
11.3.3.12 calc_gcarc()

```
void sacfmt::Trace::calc_gcarc ( )  [private], [noexcept]
```

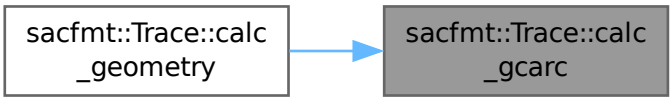
Calculate great-circle arc-distance (gcarc).

```
00916                                {
00917   Trace::gcarc(
00918       static_cast<float>(sacfmt::gcarc(stla(), stlo(), evla(), evlo())));
00919 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



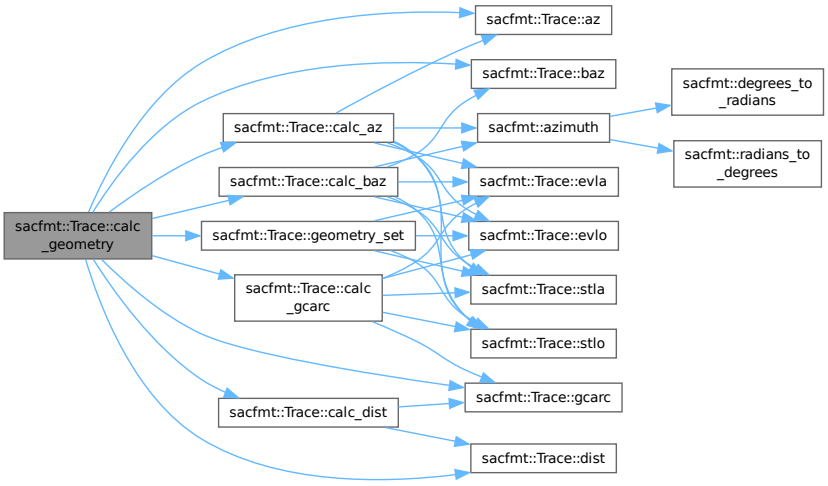
11.3.3.13 calc_geometry()

```
void sacfmt::Trace::calc_geometry ( )  [noexcept]
```

Calculates gcarc, dist, az, and baz from stla, stlo, evla, and evlo.

```
00872                                    {
00873   if (geometry_set()) {
00874     calc_gcarc();
00875     calc_dist();
00876     calc_az();
00877     calc_baz();
00878   } else {
00879     gcarc(unset_double);
00880     dist(unset_double);
00881     az(unset_double);
00882     baz(unset_double);
00883   }
00884 }
```

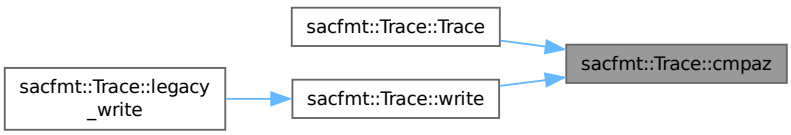
Here is the call graph for this function:



11.3.3.14 cmpaz() [1/2]

```
float sacfmt::Trace::cmpaz ( ) const  [noexcept]
01039 { return floats[sac_map.at(name::cmpaz)]; }
```

Here is the caller graph for this function:



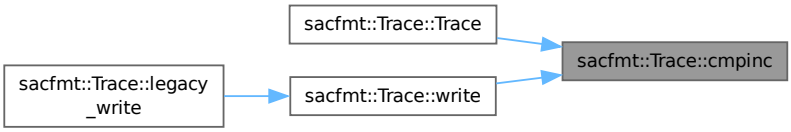
11.3.3.15 cmpaz() [2/2]

```
void sacfmt::Trace::cmpaz (
            float input )  [noexcept]
01286                                       {
01287   floats[sac_map.at(name::cmpaz)] = input;
01288 }
```

11.3.3.16 cmpinc() [1/2]

```
float sacfmt::Trace::cmpinc ( ) const  [noexcept]
01040                                  {
01041   return floats[sac_map.at(name::cmpinc)];
01042 }
```

Here is the caller graph for this function:



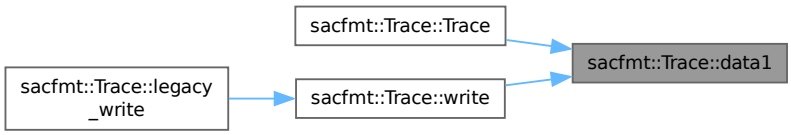
11.3.3.17 cmpinc() [2/2]

```
void sacfmt::Trace::cmpinc (
            float input )  [noexcept]
01289                                  {
01290   floats[sac_map.at(name::cmpinc)] = input;
01291 }
```

11.3.3.18 data1() [1/2]

```
std::vector< double > sacfmt::Trace::data1 ( ) const  [noexcept]
01179                                          {
01180   return data[sac_map.at(name::data1)];
01181 }
```

Here is the caller graph for this function:



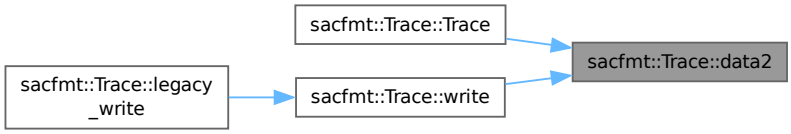
11.3.3.19 data1() [2/2]

```
void sacfmt::Trace::data1 (
            const std::vector< double > & input )  [noexcept]
01567                                              {
01568   data[sac_map.at(name::data1)] = input;
01569   // Propagate change as needed
01570   int size{static_cast<int>(data1().size())};
01571   size = (((size == 0) && (npts() == unset_int)) ? unset_int : size);
01572   if (size != npts()) {
01573     npts(size);
01574   }
01575 }
```

11.3.3.20 data2() [1/2]

```
std::vector< double > sacfmt::Trace::data2 ( ) const  [noexcept]
01182                                                 {
01183   return data[sac_map.at(name::data2)];
01184 }
```

Here is the caller graph for this function:



11.3.3.21 data2() [2/2]

```
void sacfmt::Trace::data2 (
            const std::vector< double > & input )  [noexcept]
01577                                               {
01578   data[sac_map.at(name::data2)] = input;
01579   // Proagate change as needed
01580   int size{static_cast<int>(data2().size())};
01581   size = (((size == 0) && (npts() == unset_int)) ? unset_int : size);
01582   // Need to make sure this is legal
01583   // If positive size and not-legal, make spectral
01584   if (size > 0) {
01585     // If not legal, make spectral
01586     if (leven() && (iftype() <= 1)) {
01587       iftype(2);
01588     }
01589     // If legal and different from npts, update npts
01590     if ((!leven() || (iftype() > 1)) && (size != npts())) {
01591       npts(size);
01592     }
01593   }
01594 }
```

11.3.3.22 date()

```
std::string sacfmt::Trace::date ( ) const  [noexcept]
```

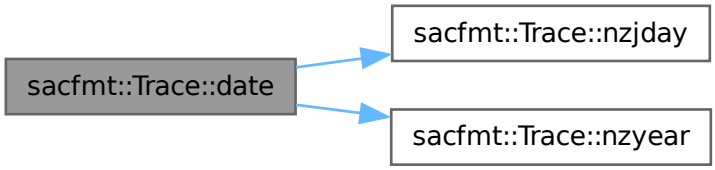
Get date string.

Returns

std::string Date (YYYY-JJJ).

```
00962                                  {
00963   // Require all to be set
00964   if ((nzyear() == unset_int) || (nzjday() == unset_int)) {
00965     return unset_word;
00966   }
00967   std::ostringstream oss{};
00968   oss « nzyear();
00969   oss « '-';
00970   oss « nzjday();
00971   return oss.str();
00972 }
```

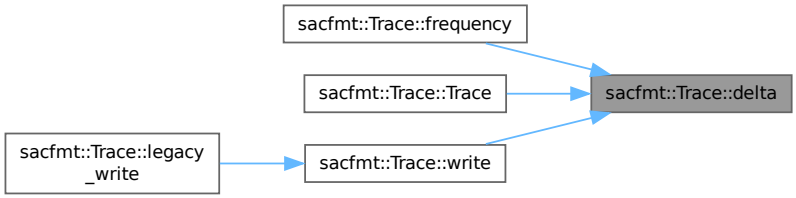
Here is the call graph for this function:



11.3.3.23 delta() [1/2]

```
double sacfmt::Trace::delta ( ) const  [noexcept]
01056                                  {
01057   return doubles[sac_map.at(name::delta)];
01058 }
```

Here is the caller graph for this function:



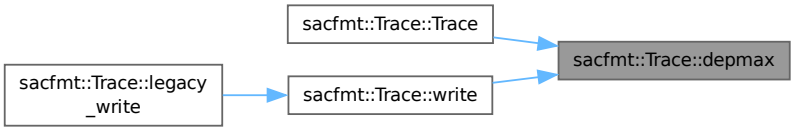
11.3.3.24 delta() [2/2]

```
void sacfmt::Trace::delta (
            double input )  [noexcept]
01305                                      {
01306   doubles[sac_map.at(name::delta)] = input;
01307 }
```

11.3.3.25 depmax() [1/2]

```
float sacfmt::Trace::depmax ( ) const  [noexcept]
01001                                  {
01002   return floats[sac_map.at(name::depmax)];
01003 }
```

Here is the caller graph for this function:



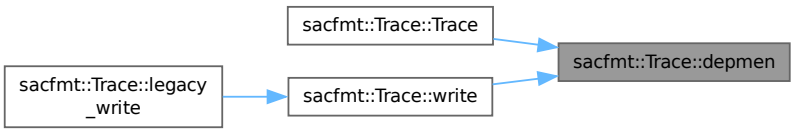
11.3.3.26 depmax() [2/2]

```
void sacfmt::Trace::depmax (
            float input )  [noexcept]
01190                                       {
01191   floats[sac_map.at(name::depmax)] = input;
01192 }
```

11.3.3.27 depmen() [1/2]

```
float sacfmt::Trace::depmen ( ) const  [noexcept]
01036                                  {
01037   return floats[sac_map.at(name::depmen)];
01038 }
```

Here is the caller graph for this function:



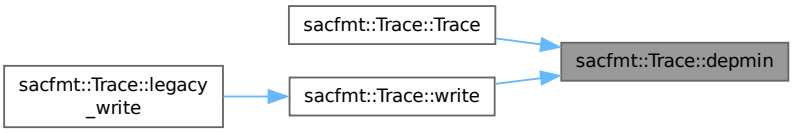
11.3.3.28 depmen() [2/2]

```
void sacfmt::Trace::depmen (
            float input )  [noexcept]
01283                                       {
01284   floats[sac_map.at(name::depmen)] = input;
01285 }
```

11.3.3.29 depmin() [1/2]

```
float sacfmt::Trace::depmin ( ) const  [noexcept]
00998                                  {
00999   return floats[sac_map.at(name::depmin)];
01000 }
```

Here is the caller graph for this function:



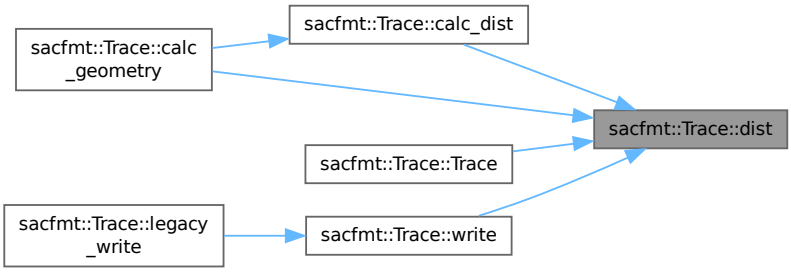
11.3.3.30 depmin() [2/2]

```
void sacfmt::Trace::depmin (
            float input )  [noexcept]
01187                                  {
01188   floats[sac_map.at(name::depmin)] = input;
01189 }
```

11.3.3.31 dist() [1/2]

```
float sacfmt::Trace::dist ( ) const  [noexcept]
01032 { return floats[sac_map.at(name::dist)]; }
```

Here is the caller graph for this function:



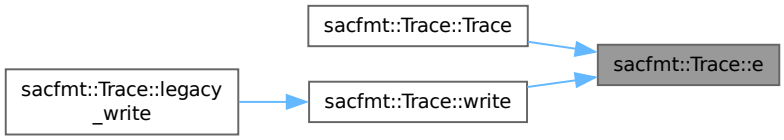
11.3.3.32 dist() [2/2]

```
void sacfmt::Trace::dist (
            float input )  [noexcept]
01271                                  {
01272   floats[sac_map.at(name::dist)] = input;
01273 }
```

11.3.3.33 e() [1/2]

```
double sacfmt::Trace::e ( ) const  [noexcept]
01060 { return doubles[sac_map.at(name::e)]; }
```

Here is the caller graph for this function:



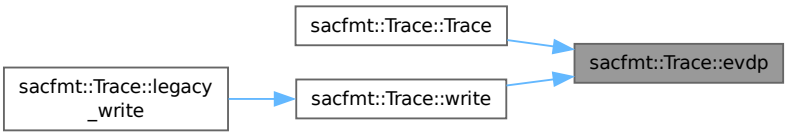
11.3.3.34 e() [2/2]

```
void sacfmt::Trace::e (
            double input )  [noexcept]
01311                                  {
01312   doubles[sac_map.at(name::e)] = input;
01313 }
```

11.3.3.35 evdp() [1/2]

```
float sacfmt::Trace::evdp ( ) const  [noexcept]
01020 { return floats[sac_map.at(name::evdp)]; }
```

Here is the caller graph for this function:



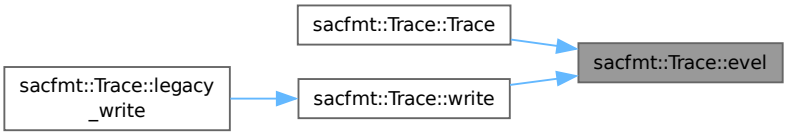
11.3.3.36 evdp() [2/2]

```
void sacfmt::Trace::evdp (
            float input )  [noexcept]
01235                                  {
01236   floats[sac_map.at(name::evdp)] = input;
01237 }
```

11.3.3.37 evel() [1/2]

```
float sacfmt::Trace::evel ( ) const  [noexcept]
01019 { return floats[sac_map.at(name::evel)]; }
```

Here is the caller graph for this function:



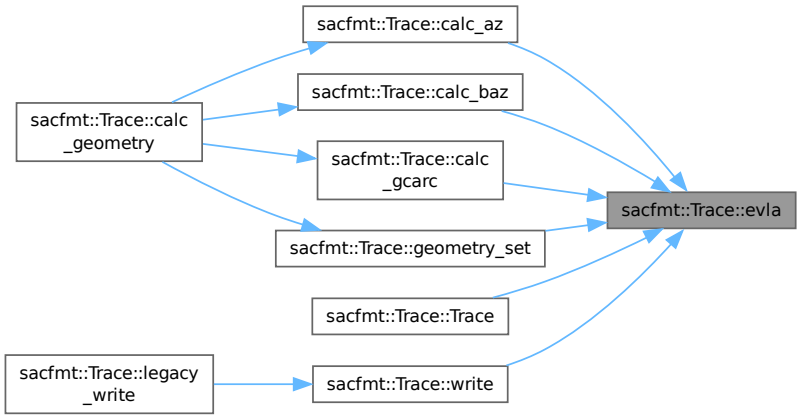
11.3.3.38 evel() [2/2]

```
void sacfmt::Trace::evel (
            float input )  [noexcept]
01232                                       {
01233   floats[sac_map.at(name::evel)] = input;
01234 }
```

11.3.3.39 evla() [1/2]

```
double sacfmt::Trace::evla ( ) const  [noexcept]
01076 { return doubles[sac_map.at(name::evla)]; }
```

Here is the caller graph for this function:



11.3.3.40 evla() [2/2]

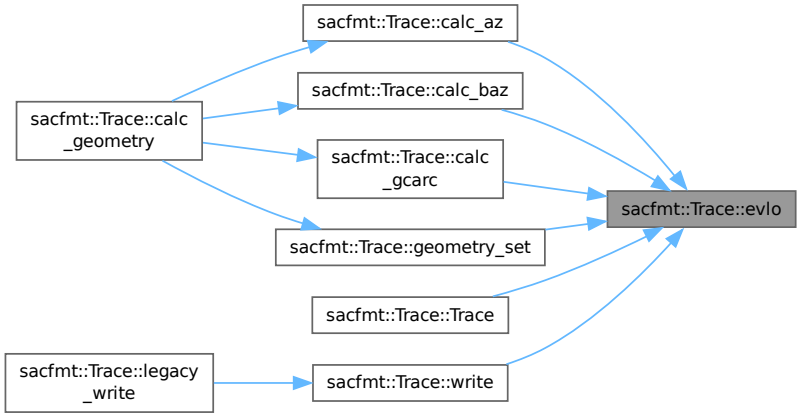
```
void sacfmt::Trace::evla (
            double input )  [noexcept]
01367                                       {
01368   double clean_input{input};
01369   if (clean_input != unset_double) {
01370     clean_input = limit_90(clean_input);
01371   }
01372   doubles[sac_map.at(name::evla)] = clean_input;
01373 }
```

Here is the call graph for this function:

11.3.3.41 evlo() [1/2]

```
double sacfmt::Trace::evlo ( ) const  [noexcept]
01077 { return doubles[sac_map.at(name::evlo)]; }
```

Here is the caller graph for this function:



11.3.3.42 evlo() [2/2]

```
void sacfmt::Trace::evlo (
            double input )  [noexcept]
01374                                       {
01375   double clean_input{input};
01376   if (clean_input != unset_double) {
01377     clean_input = limit_180(clean_input);
01378   }
01379   doubles[sac_map.at(name::evlo)] = clean_input;
```

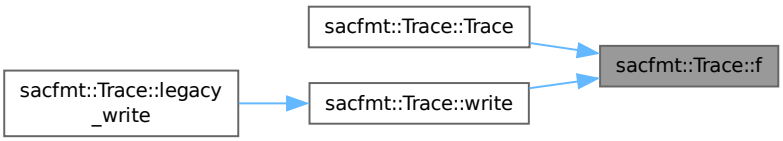
```
01380 }
```

Here is the call graph for this function:

11.3.3.43 f() [1/2]

```
double sacfmt::Trace::f ( ) const  [noexcept]
01073 { return doubles[sac_map.at(name::f)]; }
```

Here is the caller graph for this function:



11.3.3.44 f() [2/2]

```
void sacfmt::Trace::f (
            double input )  [noexcept]
01350                                   {
01351   doubles[sac_map.at(name::f)] = input;
01352 }
```

11.3.3.45 frequency()

```
double sacfmt::Trace::frequency ( ) const  [noexcept]
```

Calculate frequency from delta.

$$f = \frac{1}{\delta}$$

Returns

double Frequency.

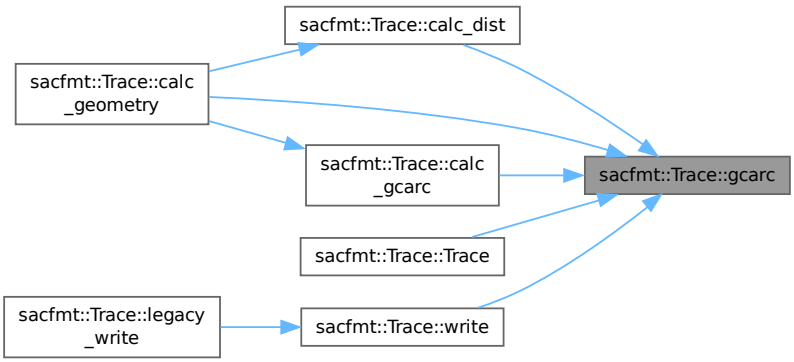
```
00895                                       {
00896   const double delta_val{delta()};
00897   if ((delta_val == unset_double) || (delta_val <= 0)) {
00898     return unset_double;
00899   }
00900   return 1.0 / delta_val;
00901 }
```

Here is the call graph for this function:

11.3.3.46 gcarc() [1/2]

```
float sacfmt::Trace::gcarc ( ) const  [noexcept]
01035 { return floats[sac_map.at(name::gcarc)]; }
```

Here is the caller graph for this function:



11.3.3.47 gcarc() [2/2]

```
void sacfmt::Trace::gcarc (
            float input )  [noexcept]
01280                                       {
01281   floats[sac_map.at(name::gcarc)] = input;
01282 }
```

11.3.3.48 geometry_set()

```
bool sacfmt::Trace::geometry_set ( ) const  [private], [noexcept]
```

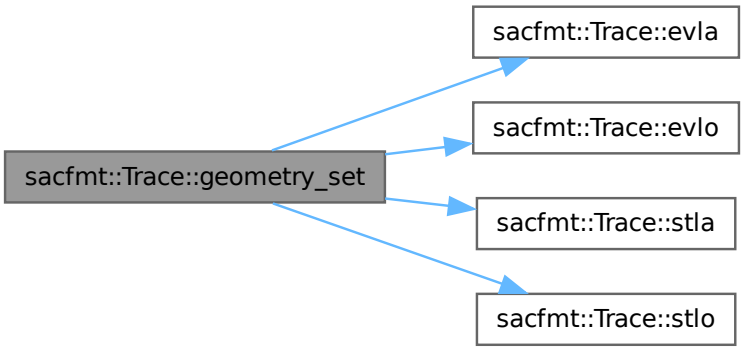
Determine if locations are set for geometry calculation.

Returns

bool True if able to calculate geometry.

```
00908                                       {
00909   return (stla() != unset_double) && (stlo() != unset_double) &&
00910          (evla() != unset_double) && (evlo() != unset_double);
00911 }
```

Here is the call graph for this function:

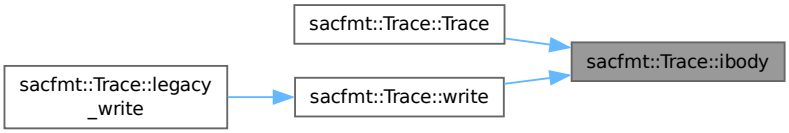


Here is the caller graph for this function:

11.3.3.49 ibody() [1/2]

```
int sacfmt::Trace::ibody ( ) const  [noexcept]
01108 { return ints[sac_map.at(name::ibody)]; }
```

Here is the caller graph for this function:



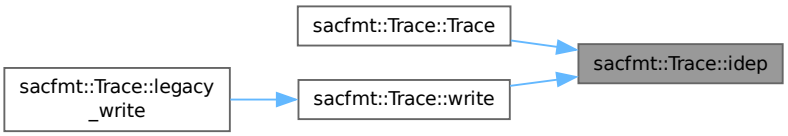
11.3.3.50 ibody() [2/2]

```
void sacfmt::Trace::ibody (
            int input )  [noexcept]
01473                                       {
01474   ints[sac_map.at(name::ibody)] = input;
01475 }
```

11.3.3.51 idep() [1/2]

```
int sacfmt::Trace::idep ( ) const  [noexcept]
01098 { return ints[sac_map.at(name::idep)]; }
```

Here is the caller graph for this function:



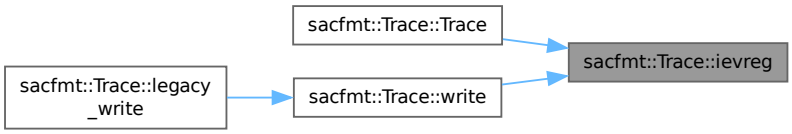
11.3.3.52 idep() [2/2]

```
void sacfmt::Trace::idep (
            int input )  [noexcept]
01443                                       {
01444   ints[sac_map.at(name::idep)] = input;
01445 }
```

11.3.3.53 ievreg() [1/2]

```
int sacfmt::Trace::ievreg ( ) const  [noexcept]
01102 { return ints[sac_map.at(name::ievreg)]; }
```

Here is the caller graph for this function:



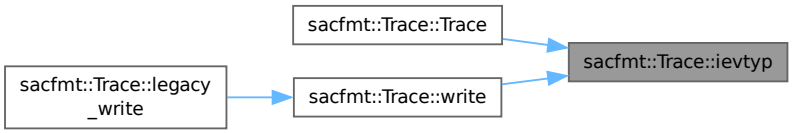
11.3.3.54 ievreg() [2/2]

```
void sacfmt::Trace::ievreg (
            int input )  [noexcept]
01455                                   {
01456   ints[sac_map.at(name::ievreg)] = input;
01457 }
```

11.3.3.55 ievtyp() [1/2]

```
int sacfmt::Trace::ievtyp ( ) const  [noexcept]
01103 { return ints[sac_map.at(name::ievtyp)]; }
```

Here is the caller graph for this function:



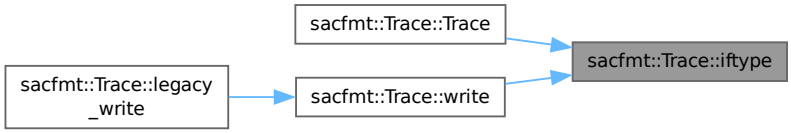
11.3.3.56 ievtyp() [2/2]

```
void sacfmt::Trace::ievtyp (
            int input )  [noexcept]
01458                                   {
01459   ints[sac_map.at(name::ievtyp)] = input;
01460 }
```

11.3.3.57 iftype() [1/2]

```
int sacfmt::Trace::iftype ( ) const  [noexcept]
01097 { return ints[sac_map.at(name::iftype)]; }
```

Here is the caller graph for this function:



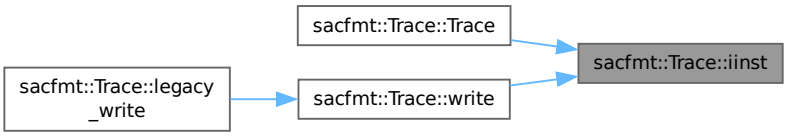
11.3.3.58 iftype() [2/2]

```
void sacfmt::Trace::iftype (
            int input )  [noexcept]
01434                                    {
01435   ints[sac_map.at(name::iftype)] = input;
01436   const size_t size{npts() >= 0 ? static_cast<size_t>(npts()) : 0};
01437   // Uneven 2D data not supported as not in specification
01438   if ((input > 1) && !leven()) {
01439     leven(true);
01440   }
01441   resize_data2(size);
01442 }
```

11.3.3.59 iinst() [1/2]

```
int sacfmt::Trace::iinst ( ) const  [noexcept]
01100 { return ints[sac_map.at(name::iinst)]; }
```

Here is the caller graph for this function:



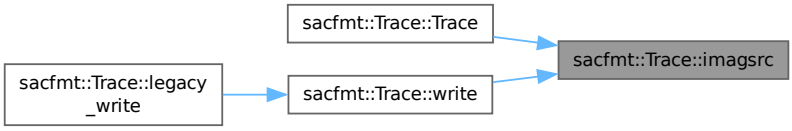
11.3.3.60 iinst() [2/2]

```
void sacfmt::Trace::iinst (
            int input )  [noexcept]
01449                                    {
01450   ints[sac_map.at(name::iinst)] = input;
01451 }
```

11.3.3.61 imagsrc() [1/2]

```
int sacfmt::Trace::imagsrc ( ) const  [noexcept]
01107 { return ints[sac_map.at(name::imagsrc)]; }
```

Here is the caller graph for this function:



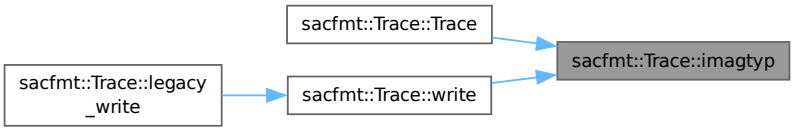
11.3.3.62 imagsrc() [2/2]

```
void sacfmt::Trace::imagsrc (
            int input )  [noexcept]
01470                                           {
01471   ints[sac_map.at(name::imagsrc)] = input;
01472 }
```

11.3.3.63 imagtyp() [1/2]

```
int sacfmt::Trace::imagtyp ( ) const  [noexcept]
01106 { return ints[sac_map.at(name::imagtyp)]; }
```

Here is the caller graph for this function:



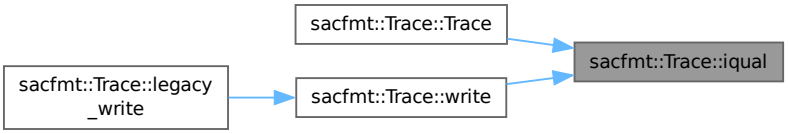
11.3.3.64 imagtyp() [2/2]

```
void sacfmt::Trace::imagtyp (
            int input )  [noexcept]
01467                                           {
01468   ints[sac_map.at(name::imagtyp)] = input;
01469 }
```

11.3.3.65 iqual() [1/2]

```
int sacfmt::Trace::iqual ( ) const  [noexcept]
01104 { return ints[sac_map.at(name::iqual)]; }
```

Here is the caller graph for this function:



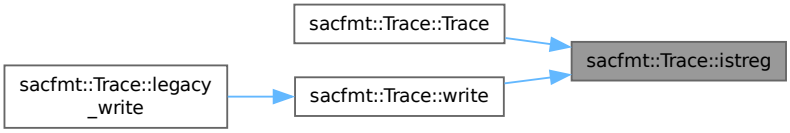
11.3.3.66 iqual() [2/2]

```
void sacfmt::Trace::iqual (
            int input )  [noexcept]
01461                                    {
01462   ints[sac_map.at(name::iqual)] = input;
01463 }
```

11.3.3.67 istreg() [1/2]

```
int sacfmt::Trace::istreg ( ) const  [noexcept]
01101 { return ints[sac_map.at(name::istreg)]; }
```

Here is the caller graph for this function:



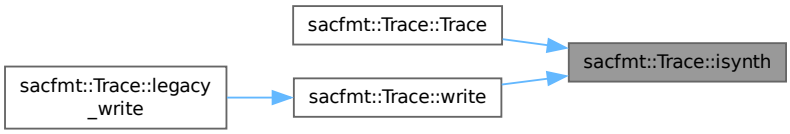
11.3.3.68 istreg() [2/2]

```
void sacfmt::Trace::istreg (
            int input )  [noexcept]
01452                                    {
01453   ints[sac_map.at(name::istreg)] = input;
01454 }
```

11.3.3.69 isynth() [1/2]

```
int sacfmt::Trace::isynth ( ) const  [noexcept]
01105 { return ints[sac_map.at(name::isynth)]; }
```

Here is the caller graph for this function:



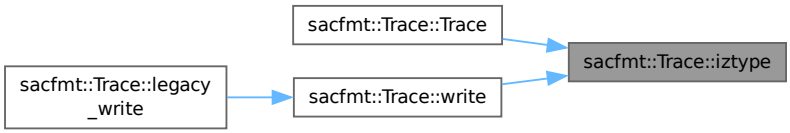
11.3.3.70 isynth() [2/2]

```
void sacfmt::Trace::isynth (
            int input )  [noexcept]
01464                                    {
01465   ints[sac_map.at(name::isynth)] = input;
01466 }
```

11.3.3.71 iztype() [1/2]

```
int sacfmt::Trace::iztype ( ) const  [noexcept]
01099 { return ints[sac_map.at(name::iztype)]; }
```

Here is the caller graph for this function:



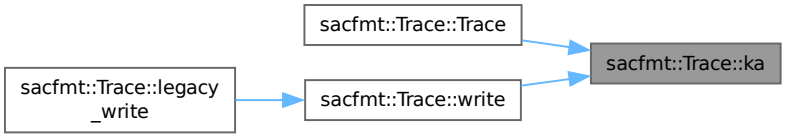
11.3.3.72 iztype() [2/2]

```
void sacfmt::Trace::iztype (
            int input )  [noexcept]
01446                                    {
01447   ints[sac_map.at(name::iztype)] = input;
01448 }
```

11.3.3.73 ka() [1/2]

```
std::string sacfmt::Trace::ka ( ) const  [noexcept]
01125 { return strings[sac_map.at(name::ka)]; }
```

Here is the caller graph for this function:



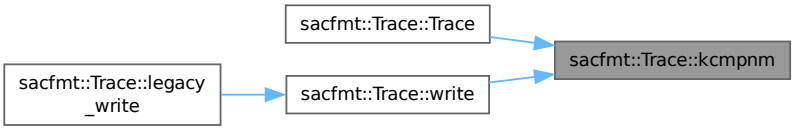
11.3.3.74 ka() [2/2]

```
void sacfmt::Trace::ka (
            const std::string & input )  [noexcept]
01508                                    {
01509   strings[sac_map.at(name::ka)] = input;
01510 }
```

11.3.3.75 kcmpnm() [1/2]

```
std::string sacfmt::Trace::kcmpnm ( ) const  [noexcept]
01166                                        {
01167   return strings[sac_map.at(name::kcmpnm)];
01168 }
```

Here is the caller graph for this function:



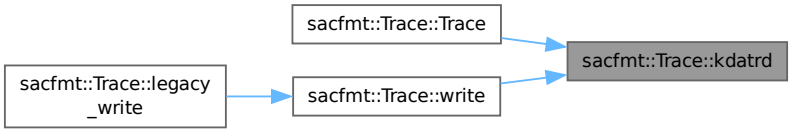
11.3.3.76 kcmpnm() [2/2]

```
void sacfmt::Trace::kcmpnm (
            const std::string & input )  [noexcept]
01553                                            {
01554   strings[sac_map.at(name::kcmpnm)] = input;
01555 }
```

11.3.3.77 kdatrd() [1/2]

```
std::string sacfmt::Trace::kdatrd ( ) const  [noexcept]
01172                                        {
01173   return strings[sac_map.at(name::kdatrd)];
01174 }
```

Here is the caller graph for this function:



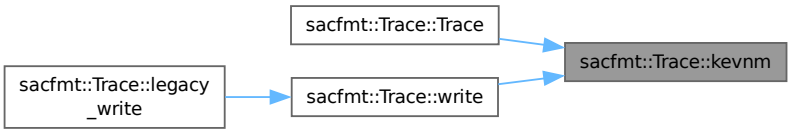
11.3.3.78 kdatrd() [2/2]

```
void sacfmt::Trace::kdatrd (
            const std::string & input )  [noexcept]
01559                                    {
01560   strings[sac_map.at(name::kdatrd)] = input;
01561 }
```

11.3.3.79 kevnm() [1/2]

```
std::string sacfmt::Trace::kevnm ( ) const  [noexcept]
01118                                       {
01119   return strings[sac_map.at(name::kevnm)];
01120 }
```

Here is the caller graph for this function:



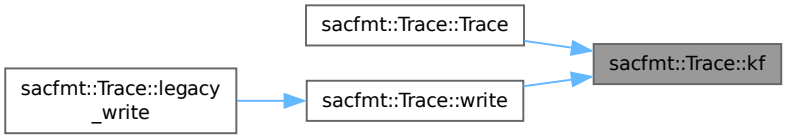
11.3.3.80 kevnm() [2/2]

```
void sacfmt::Trace::kevnm (
            const std::string & input )  [noexcept]
01499                                    {
01500   strings[sac_map.at(name::kevnm)] = input;
01501 }
```

11.3.3.81 kf() [1/2]

```
std::string sacfmt::Trace::kf ( ) const  [noexcept]
01156 { return strings[sac_map.at(name::kf)]; }
```

Here is the caller graph for this function:



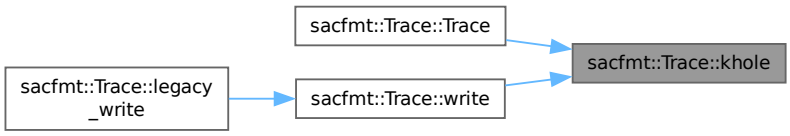
11.3.3.82 kf() [2/2]

```
void sacfmt::Trace::kf (
            const std::string & input )  [noexcept]
01541                                    {
01542   strings[sac_map.at(name::kf)] = input;
01543 }
```

11.3.3.83 khole() [1/2]

```
std::string sacfmt::Trace::khole ( ) const  [noexcept]
01121                                       {
01122   return strings[sac_map.at(name::khole)];
01123 }
```

Here is the caller graph for this function:



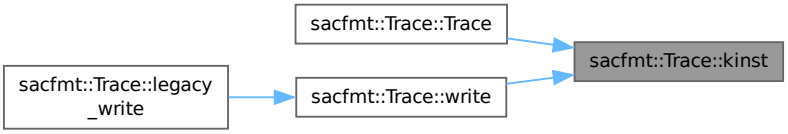
11.3.3.84 khole() [2/2]

```
void sacfmt::Trace::khole (
            const std::string & input )  [noexcept]
01502                                       {
01503   strings[sac_map.at(name::khole)] = input;
01504 }
```

11.3.3.85 kinst() [1/2]

```
std::string sacfmt::Trace::kinst ( ) const  [noexcept]
01175                                       {
01176   return strings[sac_map.at(name::kinst)];
01177 }
```

Here is the caller graph for this function:



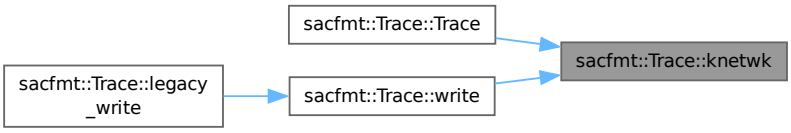
11.3.3.86 kinst() [2/2]

```
void sacfmt::Trace::kinst (
            const std::string & input )  [noexcept]
01562                                                {
01563   strings[sac_map.at(name::kinst)] = input;
01564 }
```

11.3.3.87 knetwk() [1/2]

```
std::string sacfmt::Trace::knetwk ( ) const  [noexcept]
01169                                        {
01170   return strings[sac_map.at(name::knetwk)];
01171 }
```

Here is the caller graph for this function:



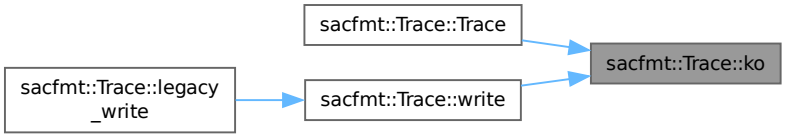
11.3.3.88 knetwk() [2/2]

```
void sacfmt::Trace::knetwk (
            const std::string & input )  [noexcept]
01556                                                {
01557   strings[sac_map.at(name::knetwk)] = input;
01558 }
```

11.3.3.89 ko() [1/2]

```
std::string sacfmt::Trace::ko ( ) const  [noexcept]
01124 { return strings[sac_map.at(name::ko)]; }
```

Here is the caller graph for this function:



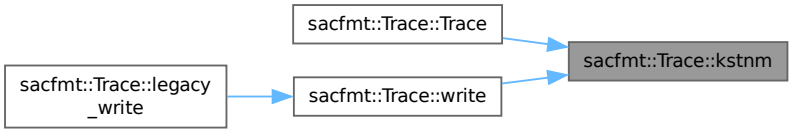
11.3.3.90 ko() [2/2]

```
void sacfmt::Trace::ko (
            const std::string & input )  [noexcept]
01505                                    {
01506   strings[sac_map.at(name::ko)] = input;
01507 }
```

11.3.3.91 kstnm() [1/2]

```
std::string sacfmt::Trace::kstnm ( ) const  [noexcept]
01115                                       {
01116   return strings[sac_map.at(name::kstnm)];
01117 }
```

Here is the caller graph for this function:



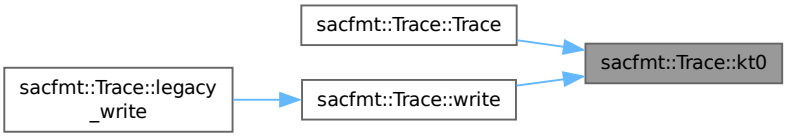
11.3.3.92 kstnm() [2/2]

```
void sacfmt::Trace::kstnm (
            const std::string & input )  [noexcept]
01496                                    {
01497   strings[sac_map.at(name::kstnm)] = input;
01498 }
```

11.3.3.93 kt0() [1/2]

```
std::string sacfmt::Trace::kt0 ( ) const  [noexcept]
01126                                     {
01127   return strings[sac_map.at(name::kt0)];
01128 }
```

Here is the caller graph for this function:



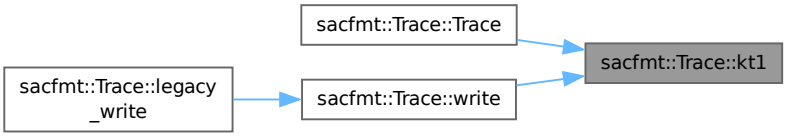
11.3.3.94 kt0() [2/2]

```
void sacfmt::Trace::kt0 (
            const std::string & input )  [noexcept]
01511                                        {
01512   strings[sac_map.at(name::kt0)] = input;
01513 }
```

11.3.3.95 kt1() [1/2]

```
std::string sacfmt::Trace::kt1 ( ) const  [noexcept]
01129                                     {
01130   return strings[sac_map.at(name::kt1)];
01131 }
```

Here is the caller graph for this function:



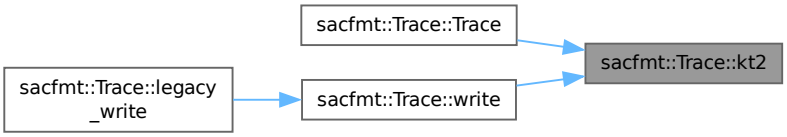
11.3.3.96 kt1() [2/2]

```
void sacfmt::Trace::kt1 (
            const std::string & input )  [noexcept]
01514                                        {
01515   strings[sac_map.at(name::kt1)] = input;
01516 }
```

11.3.3.97 kt2() [1/2]

```
std::string sacfmt::Trace::kt2 ( ) const  [noexcept]
01132                                     {
01133   return strings[sac_map.at(name::kt2)];
01134 }
```

Here is the caller graph for this function:



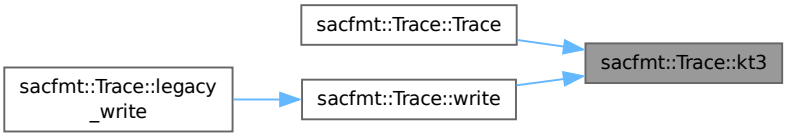
11.3.3.98 kt2() [2/2]

```
void sacfmt::Trace::kt2 (
            const std::string & input )  [noexcept]
01517                                        {
01518   strings[sac_map.at(name::kt2)] = input;
01519 }
```

11.3.3.99 kt3() [1/2]

```
std::string sacfmt::Trace::kt3 ( ) const  [noexcept]
01135                                     {
01136   return strings[sac_map.at(name::kt3)];
01137 }
```

Here is the caller graph for this function:



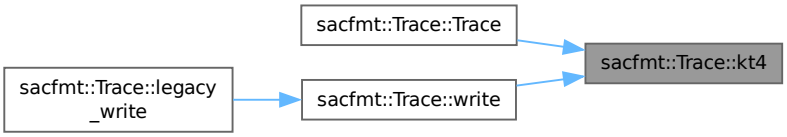
11.3.3.100 kt3() [2/2]

```
void sacfmt::Trace::kt3 (
            const std::string & input )  [noexcept]
01520                                        {
01521   strings[sac_map.at(name::kt3)] = input;
01522 }
```

11.3.3.101 kt4() [1/2]

```
std::string sacfmt::Trace::kt4 ( ) const  [noexcept]
01138                                     {
01139   return strings[sac_map.at(name::kt4)];
01140 }
```

Here is the caller graph for this function:



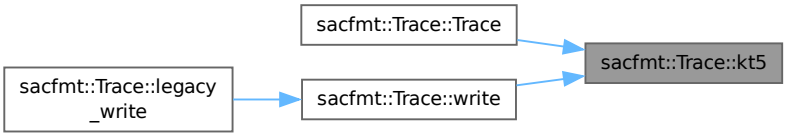
11.3.3.102 kt4() [2/2]

```
void sacfmt::Trace::kt4 (
            const std::string & input )  [noexcept]
01523                                        {
01524   strings[sac_map.at(name::kt4)] = input;
01525 }
```

11.3.3.103 kt5() [1/2]

```
std::string sacfmt::Trace::kt5 ( ) const  [noexcept]
01141                                     {
01142   return strings[sac_map.at(name::kt5)];
01143 }
```

Here is the caller graph for this function:



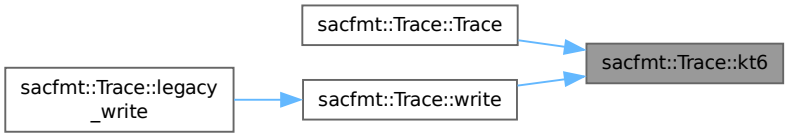
11.3.3.104 kt5() [2/2]

```
void sacfmt::Trace::kt5 (
            const std::string & input )  [noexcept]
01526                                        {
01527   strings[sac_map.at(name::kt5)] = input;
01528 }
```

11.3.3.105 kt6() [1/2]

```
std::string sacfmt::Trace::kt6 ( ) const  [noexcept]
01144                                     {
01145   return strings[sac_map.at(name::kt6)];
01146 }
```

Here is the caller graph for this function:



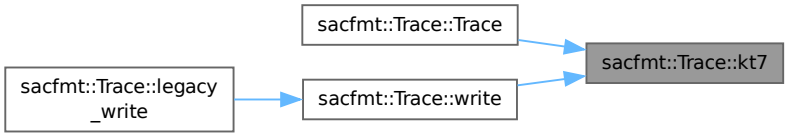
11.3.3.106 kt6() [2/2]

```
void sacfmt::Trace::kt6 (
            const std::string & input )  [noexcept]
01529                                        {
01530   strings[sac_map.at(name::kt6)] = input;
01531 }
```

11.3.3.107 kt7() [1/2]

```
std::string sacfmt::Trace::kt7 ( ) const  [noexcept]
01147                                     {
01148   return strings[sac_map.at(name::kt7)];
01149 }
```

Here is the caller graph for this function:



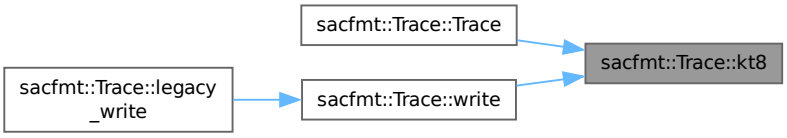
11.3.3.108 kt7() [2/2]

```
void sacfmt::Trace::kt7 (
            const std::string & input )  [noexcept]
01532                                        {
01533   strings[sac_map.at(name::kt7)] = input;
01534 }
```

11.3.3.109 kt8() [1/2]

```
std::string sacfmt::Trace::kt8 ( ) const  [noexcept]
01150                                     {
01151   return strings[sac_map.at(name::kt8)];
01152 }
```

Here is the caller graph for this function:



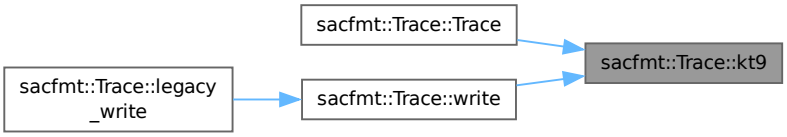
11.3.3.110 kt8() [2/2]

```
void sacfmt::Trace::kt8 (
            const std::string & input )  [noexcept]
01535                                        {
01536   strings[sac_map.at(name::kt8)] = input;
01537 }
```

11.3.3.111 kt9() [1/2]

```
std::string sacfmt::Trace::kt9 ( ) const  [noexcept]
01153                                     {
01154   return strings[sac_map.at(name::kt9)];
01155 }
```

Here is the caller graph for this function:



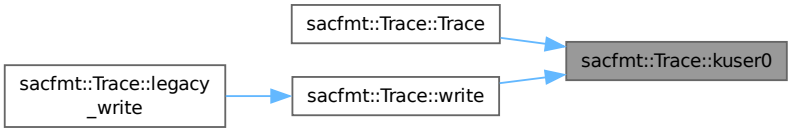
11.3.3.112 kt9() [2/2]

```
void sacfmt::Trace::kt9 (
            const std::string & input )  [noexcept]
01538                                        {
01539   strings[sac_map.at(name::kt9)] = input;
01540 }
```

11.3.3.113 kuser0() [1/2]

```
std::string sacfmt::Trace::kuser0 ( ) const  [noexcept]
01157                                        {
01158   return strings[sac_map.at(name::kuser0)];
01159 }
```

Here is the caller graph for this function:



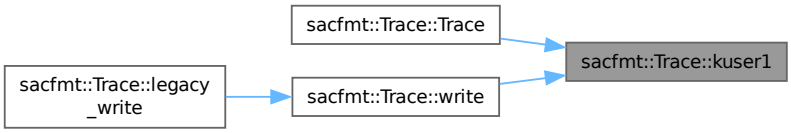
11.3.3.114 kuser0() [2/2]

```
void sacfmt::Trace::kuser0 (
            const std::string & input )  [noexcept]
01544                                            {
01545   strings[sac_map.at(name::kuser0)] = input;
01546 }
```

11.3.3.115 kuser1() [1/2]

```
std::string sacfmt::Trace::kuser1 ( ) const  [noexcept]
01160                                        {
01161   return strings[sac_map.at(name::kuser1)];
01162 }
```

Here is the caller graph for this function:



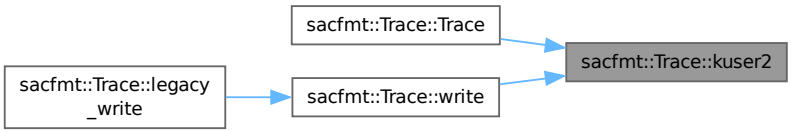
11.3.3.116 kuser1() [2/2]

```
void sacfmt::Trace::kuser1 (
            const std::string & input )  [noexcept]
01547                                            {
01548   strings[sac_map.at(name::kuser1)] = input;
01549 }
```

11.3.3.117 kuser2() [1/2]

```
std::string sacfmt::Trace::kuser2 ( ) const  [noexcept]
01163                                        {
01164   return strings[sac_map.at(name::kuser2)];
01165 }
```

Here is the caller graph for this function:



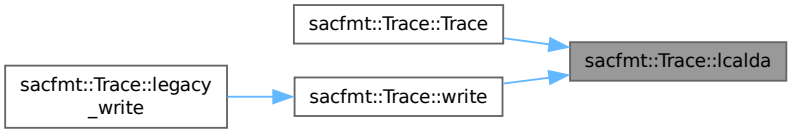
11.3.3.118 kuser2() [2/2]

```
void sacfmt::Trace::kuser2 (
            const std::string & input )  [noexcept]
01550                                                {
01551   strings[sac_map.at(name::kuser2)] = input;
01552 }
```

11.3.3.119 lcalda() [1/2]

```
bool sacfmt::Trace::lcalda ( ) const  [noexcept]
01113 { return bools[sac_map.at(name::lcalda)]; }
```

Here is the caller graph for this function:



11.3.3.120 lcalda() [2/2]

```
void sacfmt::Trace::lcalda (
            bool input )  [noexcept]
01492                                    {
01493   bools[sac_map.at(name::lcalda)] = input;
01494 }
```

11.3.3.121 legacy_write()

```
void sacfmt::Trace::legacy_write (
            const std::filesystem::path & path ) const
```

Binary SAC-file legacy-write convenience function.

Parameters

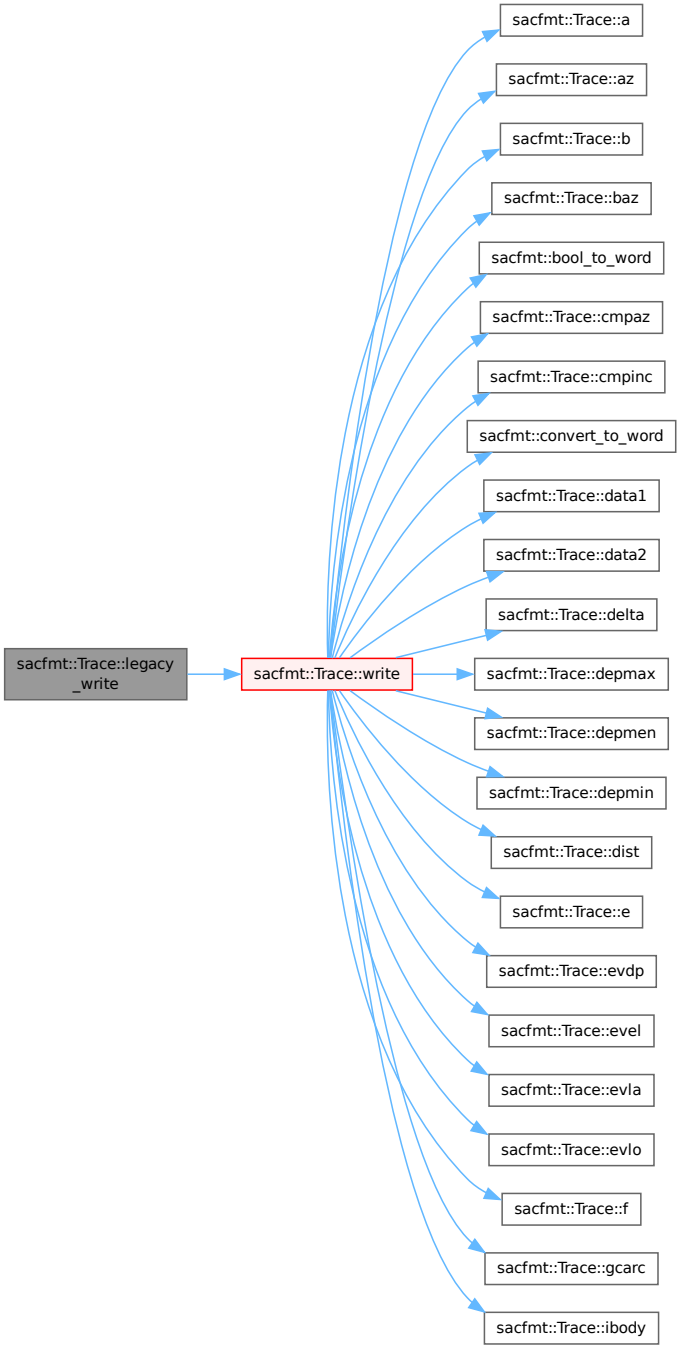
in	*path*	std::filesystem::path SAC-file to be written.

Exceptions

io_error	If the file cannot be written (bad path or bad permissions).
std::execption	Other unwritable issues (not enough space, disk failure, etc.).

```
02190                                                                {
02191   write(path, true);
02192 }
```

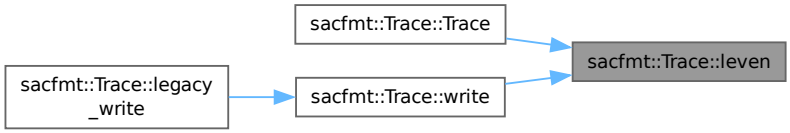
Here is the call graph for this function:



11.3.3.122 leven() [1/2]

```
bool sacfmt::Trace::leven ( ) const  [noexcept]
01110 { return bools[sac_map.at(name::leven)]; }
```

Here is the caller graph for this function:



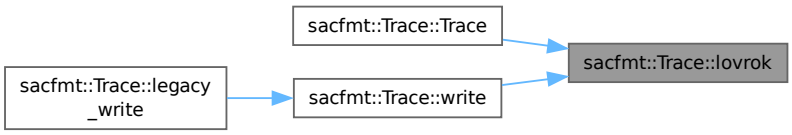
11.3.3.123 leven() [2/2]

```
void sacfmt::Trace::leven (
            bool input )  [noexcept]
01477                                       {
01478   bools[sac_map.at(name::leven)] = input;
01479   const size_t size{npts() >= 0 ? static_cast<size_t>(npts()) : 0};
01480   // Uneven 2D data not supported since not in specification
01481   if (!input && (iftype() > 1)) {
01482     iftype(unset_int);
01483   }
01484   resize_data2(size);
01485 }
```

11.3.3.124 lovrok() [1/2]

```
bool sacfmt::Trace::lovrok ( ) const  [noexcept]
01112 { return bools[sac_map.at(name::lovrok)]; }
```

Here is the caller graph for this function:



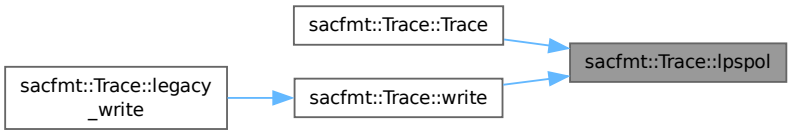
11.3.3.125 lovrok() [2/2]

```
void sacfmt::Trace::lovrok (
            bool input )  [noexcept]
01489                                       {
01490   bools[sac_map.at(name::lovrok)] = input;
01491 }
```

11.3.3.126 lpspol() [1/2]

```
bool sacfmt::Trace::lpspol ( ) const  [noexcept]
01111 { return bools[sac_map.at(name::lpspol)]; }
```

Here is the caller graph for this function:



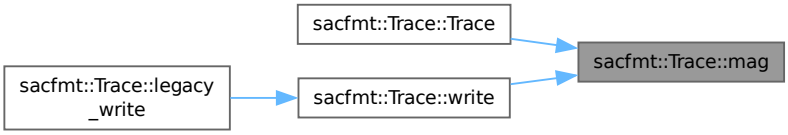
11.3.3.127 lpspol() [2/2]

```
void sacfmt::Trace::lpspol (
            bool input )  [noexcept]
01486                                    {
01487   bools[sac_map.at(name::lpspol)] = input;
01488 }
```

11.3.3.128 mag() [1/2]

```
float sacfmt::Trace::mag ( ) const  [noexcept]
01021 { return floats[sac_map.at(name::mag)]; }
```

Here is the caller graph for this function:



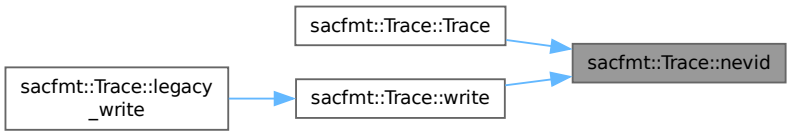
11.3.3.129 mag() [2/2]

```
void sacfmt::Trace::mag (
            float input )  [noexcept]
01238                                    {
01239   floats[sac_map.at(name::mag)] = input;
01240 }
```

11.3.3.130 nevid() [1/2]

```
int sacfmt::Trace::nevid ( ) const  [noexcept]
01091 { return ints[sac_map.at(name::nevid)]; }
```

Here is the caller graph for this function:



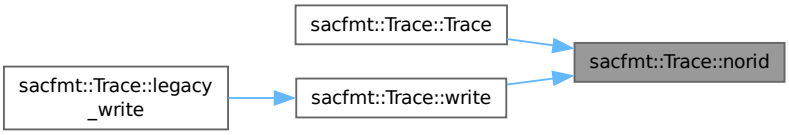
11.3.3.131 nevid() [2/2]

```
void sacfmt::Trace::nevid (
            int input )  [noexcept]
01412                                       {
01413   ints[sac_map.at(name::nevid)] = input;
01414 }
```

11.3.3.132 norid() [1/2]

```
int sacfmt::Trace::norid ( ) const  [noexcept]
01090 { return ints[sac_map.at(name::norid)]; }
```

Here is the caller graph for this function:



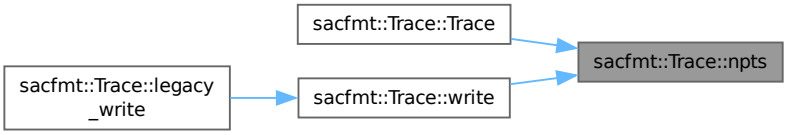
11.3.3.133 norid() [2/2]

```
void sacfmt::Trace::norid (
            int input )  [noexcept]
01409                                       {
01410   ints[sac_map.at(name::norid)] = input;
01411 }
```

11.3.3.134 npts() [1/2]

```
int sacfmt::Trace::npts ( ) const  [noexcept]
01092 { return ints[sac_map.at(name::npts)]; }
```

Here is the caller graph for this function:



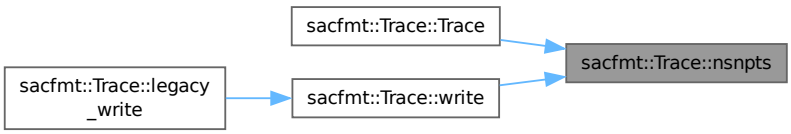
11.3.3.135 npts() [2/2]

```
void sacfmt::Trace::npts (
            int input )  [noexcept]
01415                                   {
01416   if ((input >= 0) || (input == unset_int)) {
01417     ints[sac_map.at(name::npts)] = input;
01418     const size_t size{static_cast<size_t>(input >= 0 ? input : 0)};
01419     resize_data(size);
01420   }
01421 }
```

11.3.3.136 nsnpts() [1/2]

```
int sacfmt::Trace::nsnpts ( ) const  [noexcept]
01093 { return ints[sac_map.at(name::nsnpts)]; }
```

Here is the caller graph for this function:



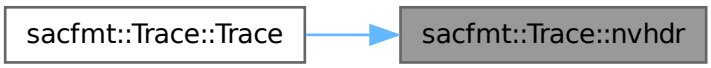
11.3.3.137 nsnpts() [2/2]

```
void sacfmt::Trace::nsnpts (
            int input )  [noexcept]
01422                                     {
01423   ints[sac_map.at(name::nsnpts)] = input;
01424 }
```

11.3.3.138 nvhdr() [1/2]

```
int sacfmt::Trace::nvhdr ( ) const  [noexcept]
01089 { return ints[sac_map.at(name::nvhdr)]; }
```

Here is the caller graph for this function:



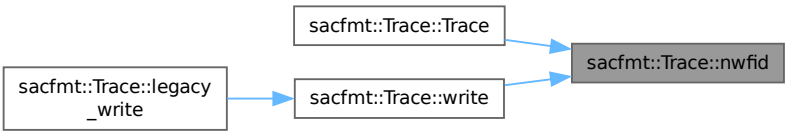
11.3.3.139 nvhdr() [2/2]

```
void sacfmt::Trace::nvhdr (
            int input )  [noexcept]
01406                                    {
01407   ints[sac_map.at(name::nvhdr)] = input;
01408 }
```

11.3.3.140 nwfid() [1/2]

```
int sacfmt::Trace::nwfid ( ) const  [noexcept]
01094 { return ints[sac_map.at(name::nwfid)]; }
```

Here is the caller graph for this function:



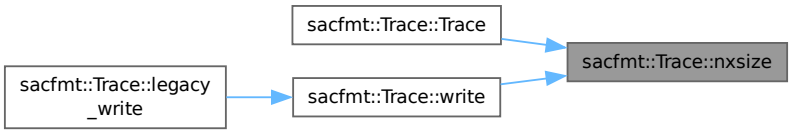
11.3.3.141 nwfid() [2/2]

```
void sacfmt::Trace::nwfid (
            int input )  [noexcept]
01425                                    {
01426   ints[sac_map.at(name::nwfid)] = input;
01427 }
```

11.3.3.142 nxsize() [1/2]

```
int sacfmt::Trace::nxsize ( ) const  [noexcept]
01095 { return ints[sac_map.at(name::nxsize)]; }
```

Here is the caller graph for this function:



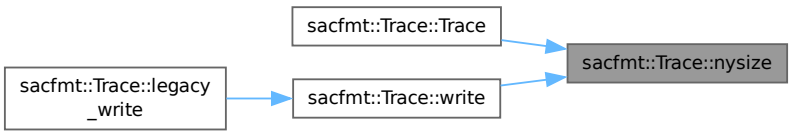
11.3.3.143 nxsize() [2/2]

```
void sacfmt::Trace::nxsize (
            int input )  [noexcept]
01428                                    {
01429   ints[sac_map.at(name::nxsize)] = input;
01430 }
```

11.3.3.144 nysize() [1/2]

```
int sacfmt::Trace::nysize ( ) const  [noexcept]
01096 { return ints[sac_map.at(name::nysize)]; }
```

Here is the caller graph for this function:



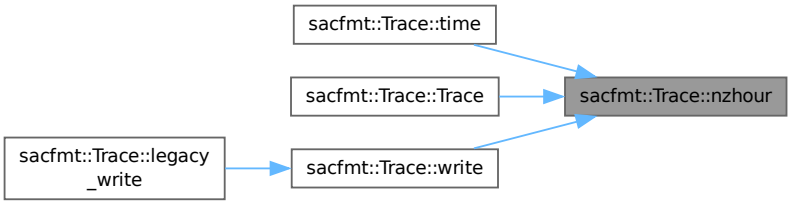
11.3.3.145 nysize() [2/2]

```
void sacfmt::Trace::nysize (
            int input )  [noexcept]
01431                                    {
01432   ints[sac_map.at(name::nysize)] = input;
01433 }
```

11.3.3.146 nzhour() [1/2]

```
int sacfmt::Trace::nzhour ( ) const  [noexcept]
01085 { return ints[sac_map.at(name::nzhour)]; }
```

Here is the caller graph for this function:



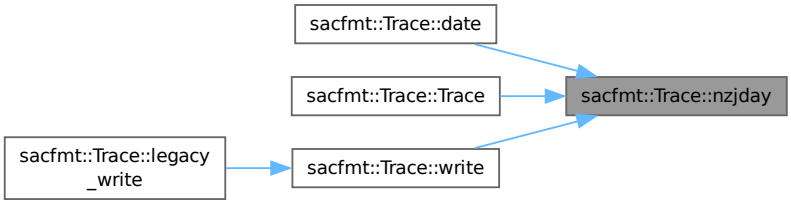
11.3.3.147 nzhour() [2/2]

```
void sacfmt::Trace::nzhour (
            int input )  [noexcept]
01394                                    {
01395   ints[sac_map.at(name::nzhour)] = input;
01396 }
```

11.3.3.148 nzjday() [1/2]

```
int sacfmt::Trace::nzjday ( ) const  [noexcept]
01084 { return ints[sac_map.at(name::nzjday)]; }
```

Here is the caller graph for this function:



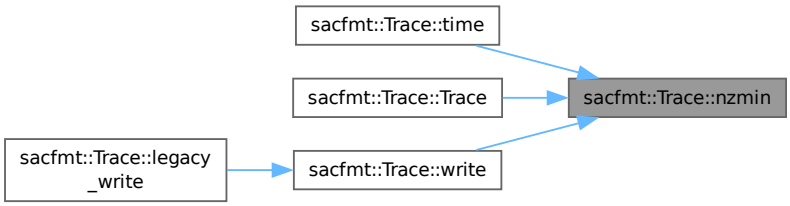
11.3.3.149 nzjday() [2/2]

```
void sacfmt::Trace::nzjday (
            int input )  [noexcept]
01391                                    {
01392   ints[sac_map.at(name::nzjday)] = input;
01393 }
```

11.3.3.150 nzmin() [1/2]

```
int sacfmt::Trace::nzmin ( ) const  [noexcept]
01086 { return ints[sac_map.at(name::nzmin)]; }
```

Here is the caller graph for this function:



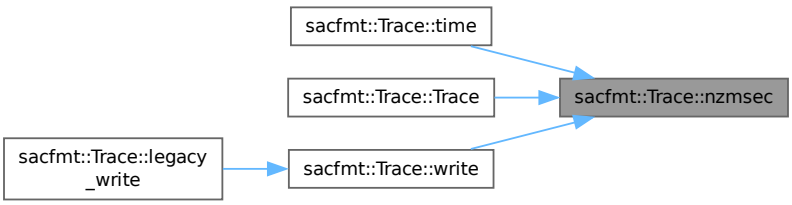
11.3.3.151 nzmin() [2/2]

```
void sacfmt::Trace::nzmin (
            int input )  [noexcept]
01397                                       {
01398   ints[sac_map.at(name::nzmin)] = input;
01399 }
```

11.3.3.152 nzmsec() [1/2]

```
int sacfmt::Trace::nzmsec ( ) const  [noexcept]
01088 { return ints[sac_map.at(name::nzmsec)]; }
```

Here is the caller graph for this function:



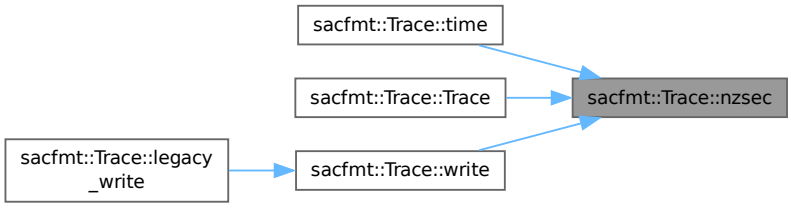
11.3.3.153 nzmsec() [2/2]

```
void sacfmt::Trace::nzmsec (
            int input )  [noexcept]
01403                                        {
01404   ints[sac_map.at(name::nzmsec)] = input;
01405 }
```

11.3.3.154 nzsec() [1/2]

```
int sacfmt::Trace::nzsec ( ) const  [noexcept]
01087 { return ints[sac_map.at(name::nzsec)]; }
```

Here is the caller graph for this function:



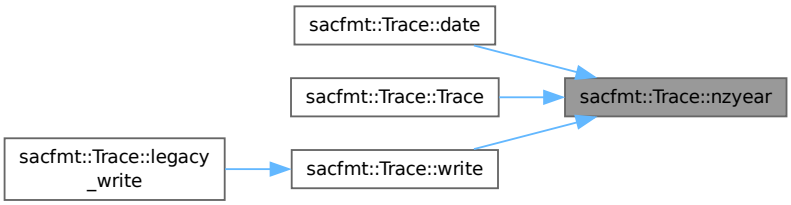
11.3.3.155 nzsec() [2/2]

```
void sacfmt::Trace::nzsec (
            int input )  [noexcept]
01400                                    {
01401   ints[sac_map.at(name::nzsec)] = input;
01402 }
```

11.3.3.156 nzyear() [1/2]

```
int sacfmt::Trace::nzyear ( ) const  [noexcept]
01083 { return ints[sac_map.at(name::nzyear)]; }
```

Here is the caller graph for this function:



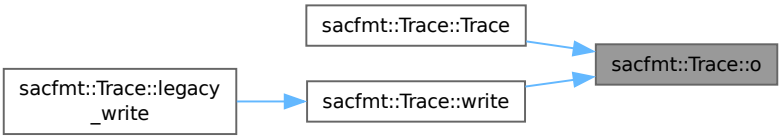
11.3.3.157 nzyear() [2/2]

```
void sacfmt::Trace::nzyear (
            int input )  [noexcept]
01388                                    {
01389   ints[sac_map.at(name::nzyear)] = input;
01390 }
```

11.3.3.158 o() [1/2]

```
double sacfmt::Trace::o ( ) const  [noexcept]
01061 { return doubles[sac_map.at(name::o)]; }
```

Here is the caller graph for this function:



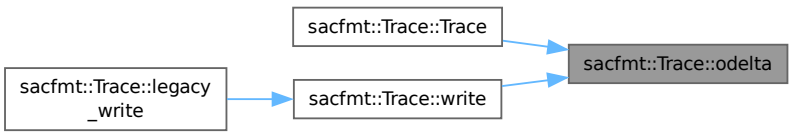
11.3.3.159 o() [2/2]

```
void sacfmt::Trace::o (
            double input )  [noexcept]
01314                                   {
01315   doubles[sac_map.at(name::o)] = input;
01316 }
```

11.3.3.160 odelta() [1/2]

```
float sacfmt::Trace::odelta ( ) const  [noexcept]
01004                                  {
01005   return floats[sac_map.at(name::odelta)];
01006 }
```

Here is the caller graph for this function:



11.3.3.161 odelta() [2/2]

```
void sacfmt::Trace::odelta (
            float input )  [noexcept]
01193                                      {
01194   floats[sac_map.at(name::odelta)] = input;
01195 }
```

11.3.3.162 operator==()

```
bool sacfmt::Trace::operator== (
            const Trace & other ) const  [noexcept]
```

Trace equality operator.

Parameters

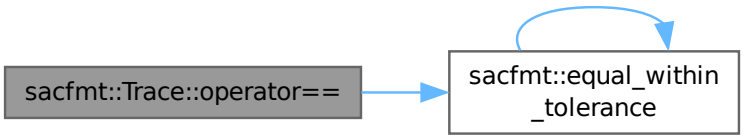
in	*this*	First Trace in comparison (LHS).
in	*other*	Second Trace in comparison (RHS).

Returns

bool Truth value of equality.

```
00847                                                    {
00848   if (floats != other.floats) {
00849     return false;
00850   }
00851   if (doubles != other.doubles) {
00852     return false;
00853   }
00854   if (ints != other.ints) {
00855     return false;
00856   }
00857   if (strings != other.strings) {
00858     return false;
00859   }
00860   if (!equal_within_tolerance(data[0], other.data[0])) {
00861     return false;
00862   }
00863   if (!equal_within_tolerance(data[1], other.data[1])) {
00864     return false;
00865   }
00866   return true;
00867 }
```

Here is the call graph for this function:



11.3.3.163 resize_data()

```
void sacfmt::Trace::resize_data (
            size_t size )  [private], [noexcept]
```

Resize data vectors (only if eligible).

Will always resize data1, data2 only resizes if it can have non-zero size.

```
01625                                                {
01626   resize_data1(size);
01627   resize_data2(size);
01628 }
```

11.3.3.164 resize_data1()

```
void sacfmt::Trace::resize_data1 (
            size_t size )  [private], [noexcept]
01596                                                   {
01597   if (size != data1().size()) {
01598     std::vector<double> new_data1{data1()};
01599     new_data1.resize(size, 0.0);
01600     data1(new_data1);
01601   }
01602 }
```

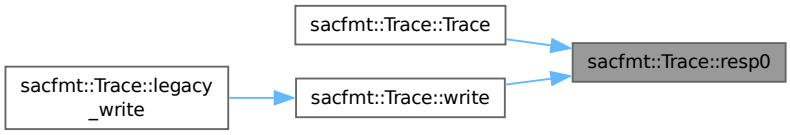
11.3.3.165 resize_data2()

```
void sacfmt::Trace::resize_data2 (
            size_t size )  [private], [noexcept]
01604                                                   {
01605   // Data2 is legal
01606   if (!leven() || (iftype() > 1)) {
01607     if (size != data2().size()) {
01608       std::vector<double> new_data2{data2()};
01609       new_data2.resize(size, 0.0);
01610       data2(new_data2);
01611     }
01612   } else {
01613     if (!data2().empty()) {
01614       std::vector<double> new_data2{};
01615       data2(new_data2);
01616     }
01617   }
01618 }
```

11.3.3.166 resp0() [1/2]

```
float sacfmt::Trace::resp0 ( ) const  [noexcept]
01007 { return floats[sac_map.at(name::resp0)]; }
```

Here is the caller graph for this function:



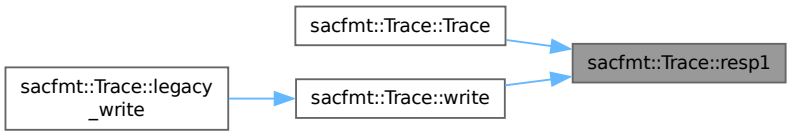
11.3.3.167 resp0() [2/2]

```
void sacfmt::Trace::resp0 (
            float input )  [noexcept]
01196                                           {
01197   floats[sac_map.at(name::resp0)] = input;
01198 }
```

11.3.3.168 resp1() [1/2]

```
float sacfmt::Trace::resp1 ( ) const  [noexcept]
01008 { return floats[sac_map.at(name::resp1)]; }
```

Here is the caller graph for this function:



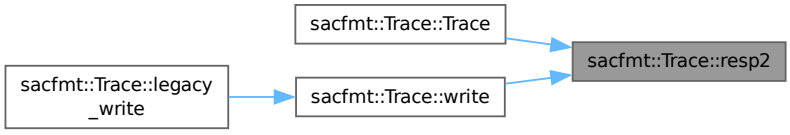
11.3.3.169 resp1() [2/2]

```
void sacfmt::Trace::resp1 (
            float input )  [noexcept]
01199                                   {
01200   floats[sac_map.at(name::resp1)] = input;
01201 }
```

11.3.3.170 resp2() [1/2]

```
float sacfmt::Trace::resp2 ( ) const  [noexcept]
01009 { return floats[sac_map.at(name::resp2)]; }
```

Here is the caller graph for this function:



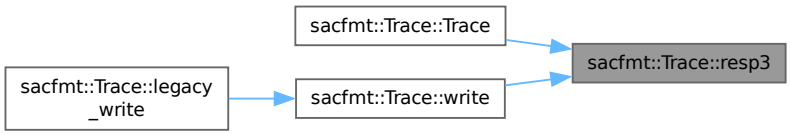
11.3.3.171 resp2() [2/2]

```
void sacfmt::Trace::resp2 (
            float input )  [noexcept]
01202                                   {
01203   floats[sac_map.at(name::resp2)] = input;
01204 }
```

11.3.3.172 resp3() [1/2]

```
float sacfmt::Trace::resp3 ( ) const  [noexcept]
01010 { return floats[sac_map.at(name::resp3)]; }
```

Here is the caller graph for this function:



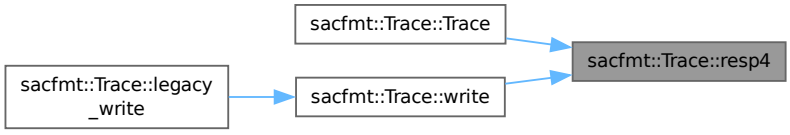
11.3.3.173 resp3() [2/2]

```
void sacfmt::Trace::resp3 (
            float input )  [noexcept]
01205                                   {
01206   floats[sac_map.at(name::resp3)] = input;
01207 }
```

11.3.3.174 resp4() [1/2]

```
float sacfmt::Trace::resp4 ( ) const  [noexcept]
01011 { return floats[sac_map.at(name::resp4)]; }
```

Here is the caller graph for this function:



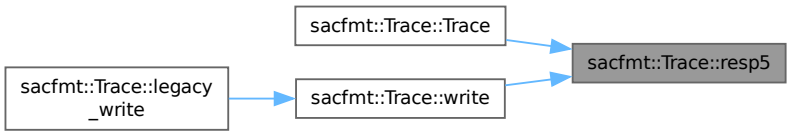
11.3.3.175 resp4() [2/2]

```
void sacfmt::Trace::resp4 (
            float input )  [noexcept]
01208                                   {
01209   floats[sac_map.at(name::resp4)] = input;
01210 }
```

11.3.3.176 resp5() [1/2]

```
float sacfmt::Trace::resp5 ( ) const  [noexcept]
01012 { return floats[sac_map.at(name::resp5)]; }
```

Here is the caller graph for this function:



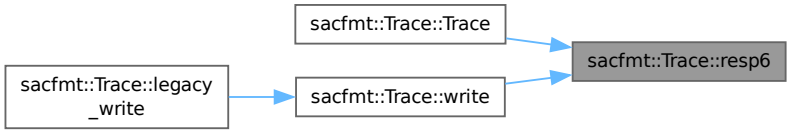
11.3.3.177 resp5() [2/2]

```
void sacfmt::Trace::resp5 (
            float input )  [noexcept]
01211                                   {
01212   floats[sac_map.at(name::resp5)] = input;
01213 }
```

11.3.3.178 resp6() [1/2]

```
float sacfmt::Trace::resp6 ( ) const  [noexcept]
01013 { return floats[sac_map.at(name::resp6)]; }
```

Here is the caller graph for this function:



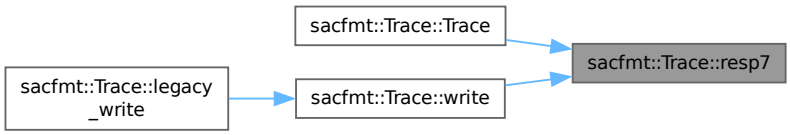
11.3.3.179 resp6() [2/2]

```
void sacfmt::Trace::resp6 (
            float input )  [noexcept]
01214                                   {
01215   floats[sac_map.at(name::resp6)] = input;
01216 }
```

11.3.3.180 resp7() [1/2]

```
float sacfmt::Trace::resp7 ( ) const  [noexcept]
01014 { return floats[sac_map.at(name::resp7)]; }
```

Here is the caller graph for this function:



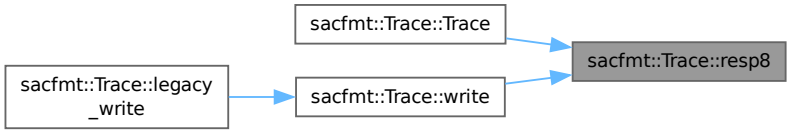
11.3.3.181 resp7() [2/2]

```
void sacfmt::Trace::resp7 (
            float input )  [noexcept]
01217                                   {
01218   floats[sac_map.at(name::resp7)] = input;
01219 }
```

11.3.3.182 resp8() [1/2]

```
float sacfmt::Trace::resp8 ( ) const  [noexcept]
01015 { return floats[sac_map.at(name::resp8)]; }
```

Here is the caller graph for this function:



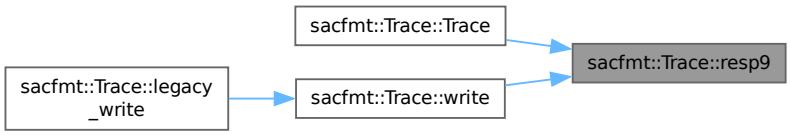
11.3.3.183 resp8() [2/2]

```
void sacfmt::Trace::resp8 (
            float input )  [noexcept]
01220                                   {
01221   floats[sac_map.at(name::resp8)] = input;
01222 }
```

11.3.3.184 resp9() [1/2]

```
float sacfmt::Trace::resp9 ( ) const  [noexcept]
01016 { return floats[sac_map.at(name::resp9)]; }
```

Here is the caller graph for this function:



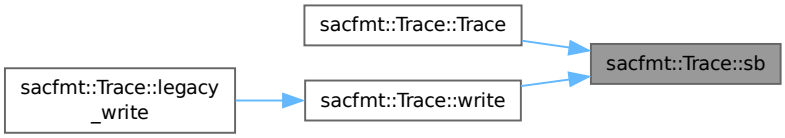
11.3.3.185 resp9() [2/2]

```
void sacfmt::Trace::resp9 (
            float input )  [noexcept]
01223                                    {
01224   floats[sac_map.at(name::resp9)] = input;
01225 }
```

11.3.3.186 sb() [1/2]

```
double sacfmt::Trace::sb ( ) const  [noexcept]
01078 { return doubles[sac_map.at(name::sb)]; }
```

Here is the caller graph for this function:



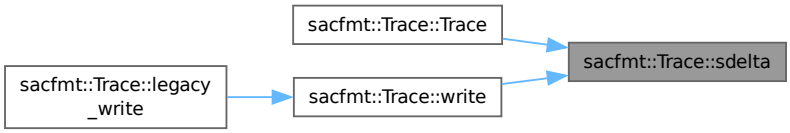
11.3.3.187 sb() [2/2]

```
void sacfmt::Trace::sb (
            double input )  [noexcept]
01381                                    {
01382   doubles[sac_map.at(name::sb)] = input;
01383 }
```

11.3.3.188 sdelta() [1/2]

```
double sacfmt::Trace::sdelta ( ) const  [noexcept]
01079                                    {
01080   return doubles[sac_map.at(name::sdelta)];
01081 }
```

Here is the caller graph for this function:



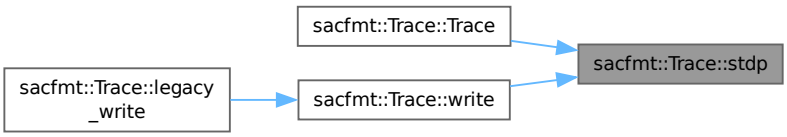
11.3.3.189 sdelta() [2/2]

```
void sacfmt::Trace::sdelta (
            double input )  [noexcept]
01384                                        {
01385   doubles[sac_map.at(name::sdelta)] = input;
01386 }
```

11.3.3.190 stdp() [1/2]

```
float sacfmt::Trace::stdp ( ) const  [noexcept]
01018 { return floats[sac_map.at(name::stdp)]; }
```

Here is the caller graph for this function:



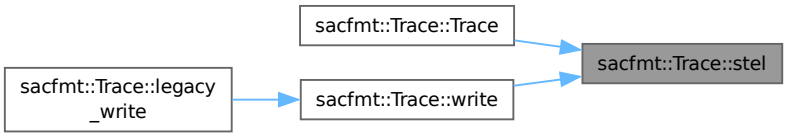
11.3.3.191 stdp() [2/2]

```
void sacfmt::Trace::stdp (
            float input )  [noexcept]
01229                                       {
01230   floats[sac_map.at(name::stdp)] = input;
01231 }
```

11.3.3.192 stel() [1/2]

```
float sacfmt::Trace::stel ( ) const  [noexcept]
01017 { return floats[sac_map.at(name::stel)]; }
```

Here is the caller graph for this function:



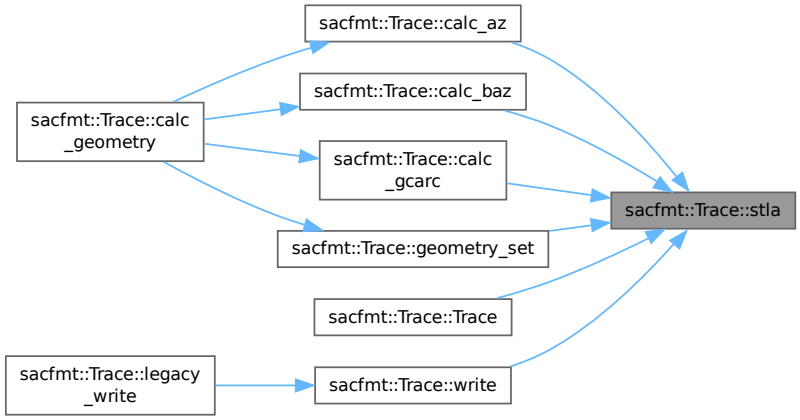
11.3.3.193 stel() [2/2]

```
void sacfmt::Trace::stel (
            float input )  [noexcept]
01226                                          {
01227   floats[sac_map.at(name::stel)] = input;
01228 }
```

11.3.3.194 stla() [1/2]

```
double sacfmt::Trace::stla ( ) const  [noexcept]
01074 { return doubles[sac_map.at(name::stla)]; }
```

Here is the caller graph for this function:



11.3.3.195 stla() [2/2]

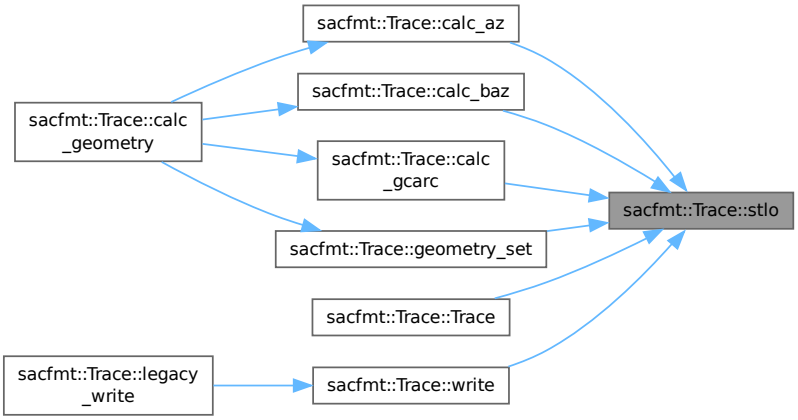
```
void sacfmt::Trace::stla (
            double input )  [noexcept]
01353                                           {
01354   double clean_input{input};
01355   if (clean_input != unset_double) {
01356     clean_input = limit_90(clean_input);
01357   }
01358   doubles[sac_map.at(name::stla)] = clean_input;
01359 }
```

Here is the call graph for this function:

11.3.3.196 stlo() [1/2]

```
double sacfmt::Trace::stlo ( ) const  [noexcept]
01075 { return doubles[sac_map.at(name::stlo)]; }
```

Here is the caller graph for this function:

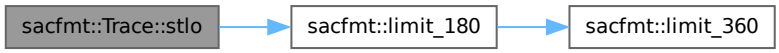


11.3.3.197 stlo() [2/2]

```
void sacfmt::Trace::stlo (
            double input )  [noexcept]
01360                                           {
01361   double clean_input{input};
01362   if (clean_input != unset_double) {
01363     clean_input = limit_180(clean_input);
01364   }
01365   doubles[sac_map.at(name::stlo)] = clean_input;
```

```
01366 }
```

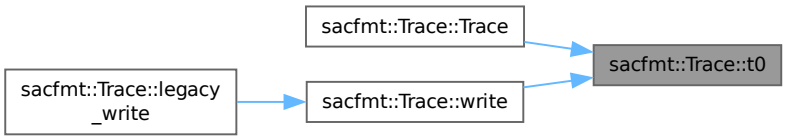
Here is the call graph for this function:



11.3.3.198 t0() [1/2]

```
double sacfmt::Trace::t0 ( ) const  [noexcept]
01063 { return doubles[sac_map.at(name::t0)]; }
```

Here is the caller graph for this function:



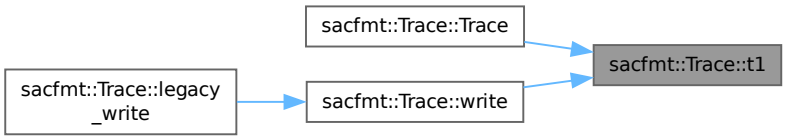
11.3.3.199 t0() [2/2]

```
void sacfmt::Trace::t0 (
            double input )  [noexcept]
01320                                       {
01321   doubles[sac_map.at(name::t0)] = input;
01322 }
```

11.3.3.200 t1() [1/2]

```
double sacfmt::Trace::t1 ( ) const  [noexcept]
01064 { return doubles[sac_map.at(name::t1)]; }
```

Here is the caller graph for this function:



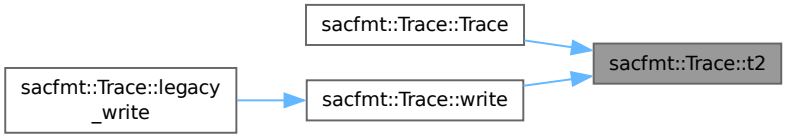
11.3.3.201 t1() [2/2]

```
void sacfmt::Trace::t1 (
            double input )  [noexcept]
01323                                       {
01324   doubles[sac_map.at(name::t1)] = input;
01325 }
```

11.3.3.202 t2() [1/2]

```
double sacfmt::Trace::t2 ( ) const  [noexcept]
01065 { return doubles[sac_map.at(name::t2)]; }
```

Here is the caller graph for this function:



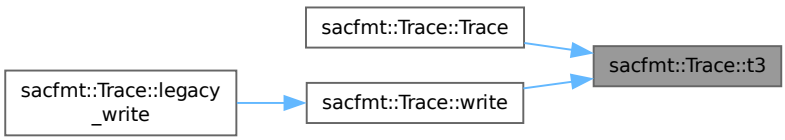
11.3.3.203 t2() [2/2]

```
void sacfmt::Trace::t2 (
            double input )  [noexcept]
01326                                       {
01327   doubles[sac_map.at(name::t2)] = input;
01328 }
```

11.3.3.204 t3() [1/2]

```
double sacfmt::Trace::t3 ( ) const  [noexcept]
01066 { return doubles[sac_map.at(name::t3)]; }
```

Here is the caller graph for this function:



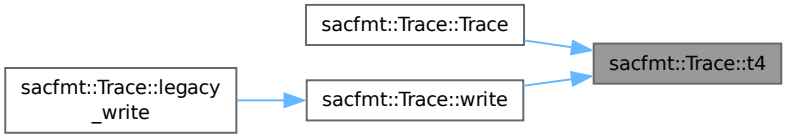
11.3.3.205 t3() [2/2]

```
void sacfmt::Trace::t3 (
            double input )  [noexcept]
01329                                       {
01330   doubles[sac_map.at(name::t3)] = input;
01331 }
```

11.3.3.206 t4() [1/2]

```
double sacfmt::Trace::t4 ( ) const  [noexcept]
01067 { return doubles[sac_map.at(name::t4)]; }
```

Here is the caller graph for this function:



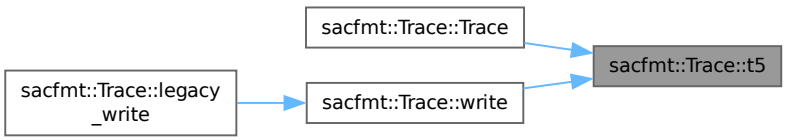
11.3.3.207 t4() [2/2]

```
void sacfmt::Trace::t4 (
            double input )  [noexcept]
01332                                       {
01333   doubles[sac_map.at(name::t4)] = input;
01334 }
```

11.3.3.208 t5() [1/2]

```
double sacfmt::Trace::t5 ( ) const  [noexcept]
01068 { return doubles[sac_map.at(name::t5)]; }
```

Here is the caller graph for this function:



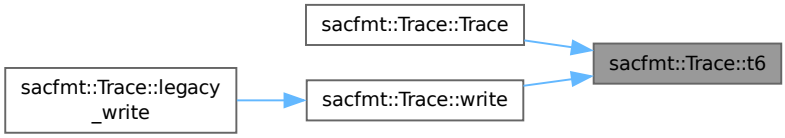
11.3.3.209 t5() [2/2]

```
void sacfmt::Trace::t5 (
            double input )  [noexcept]
01335                                       {
01336   doubles[sac_map.at(name::t5)] = input;
01337 }
```

11.3.3.210 t6() [1/2]

```
double sacfmt::Trace::t6 ( ) const  [noexcept]
01069 { return doubles[sac_map.at(name::t6)]; }
```

Here is the caller graph for this function:



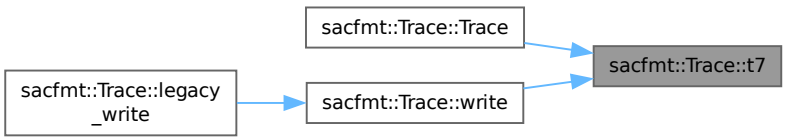
11.3.3.211 t6() [2/2]

```
void sacfmt::Trace::t6 (
            double input )  [noexcept]
01338                                       {
01339   doubles[sac_map.at(name::t6)] = input;
01340 }
```

11.3.3.212 t7() [1/2]

```
double sacfmt::Trace::t7 ( ) const  [noexcept]
01070 { return doubles[sac_map.at(name::t7)]; }
```

Here is the caller graph for this function:



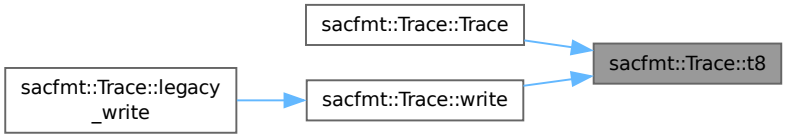
11.3.3.213 t7() [2/2]

```
void sacfmt::Trace::t7 (
            double input )  [noexcept]
01341                                       {
01342   doubles[sac_map.at(name::t7)] = input;
01343 }
```

11.3.3.214 t8() [1/2]

```
double sacfmt::Trace::t8 ( ) const  [noexcept]
01071 { return doubles[sac_map.at(name::t8)]; }
```

Here is the caller graph for this function:



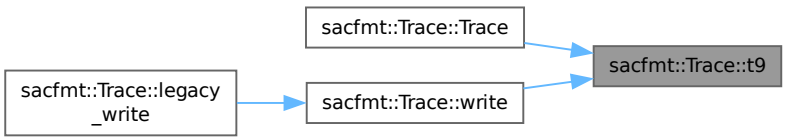
11.3.3.215 t8() [2/2]

```
void sacfmt::Trace::t8 (
            double input )  [noexcept]
01344                                       {
01345   doubles[sac_map.at(name::t8)] = input;
01346 }
```

11.3.3.216 t9() [1/2]

```
double sacfmt::Trace::t9 ( ) const  [noexcept]
01072 { return doubles[sac_map.at(name::t9)]; }
```

Here is the caller graph for this function:



11.3.3.217 t9() [2/2]

```
void sacfmt::Trace::t9 (
            double input )  [noexcept]
01347                                       {
01348   doubles[sac_map.at(name::t9)] = input;
01349 }
```

11.3.3.218 time()

```
std::string sacfmt::Trace::time ( ) const  [noexcept]
```

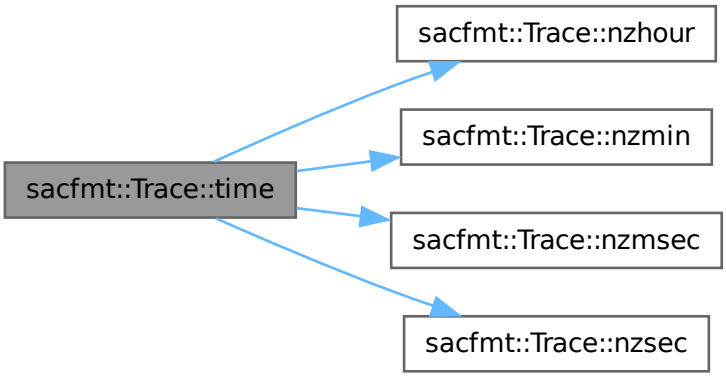
Get time string.

Returns

sstd::string Time (HH::MM:SS.sss).

```
00979                                       {
00980   // Require all to be set
00981   if ((nzhour() == unset_int) || (nzmin() == unset_int) ||
00982       (nzsec() == unset_int) || (nzmsec() == unset_int)) {
00983     return unset_word;
00984   }
00985   std::ostringstream oss{};
00986   oss « nzhour();
00987   oss « ':';
00988   oss « nzmin();
00989   oss « ':';
00990   oss « nzsec();
00991   oss « '.';
00992   oss « nzmsec();
00993   return oss.str();
00994 }
```

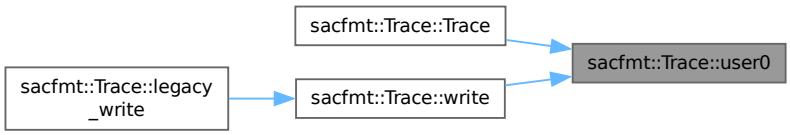
Here is the call graph for this function:



11.3.3.219 user0() [1/2]

```
float sacfmt::Trace::user0 ( ) const  [noexcept]
01022 { return floats[sac_map.at(name::user0)]; }
```

Here is the caller graph for this function:



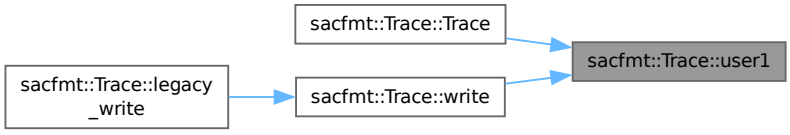
11.3.3.220 user0() [2/2]

```
void sacfmt::Trace::user0 (
            float input )  [noexcept]
01241                                   {
01242   floats[sac_map.at(name::user0)] = input;
01243 }
```

11.3.3.221 user1() [1/2]

```
float sacfmt::Trace::user1 ( ) const  [noexcept]
01023 { return floats[sac_map.at(name::user1)]; }
```

Here is the caller graph for this function:



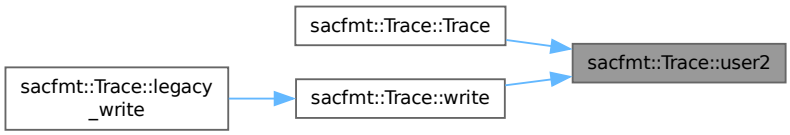
11.3.3.222 user1() [2/2]

```
void sacfmt::Trace::user1 (
            float input )  [noexcept]
01244                                   {
01245   floats[sac_map.at(name::user1)] = input;
01246 }
```

11.3.3.223 user2() [1/2]

```
float sacfmt::Trace::user2 ( ) const  [noexcept]
01024 { return floats[sac_map.at(name::user2)]; }
```

Here is the caller graph for this function:



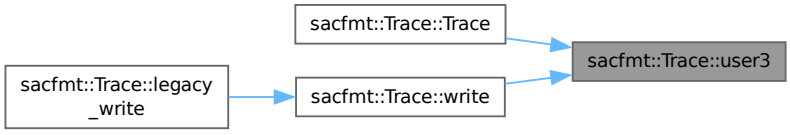
11.3.3.224 user2() [2/2]

```
void sacfmt::Trace::user2 (
            float input )  [noexcept]
01247                                       {
01248   floats[sac_map.at(name::user2)] = input;
01249 }
```

11.3.3.225 user3() [1/2]

```
float sacfmt::Trace::user3 ( ) const  [noexcept]
01025 { return floats[sac_map.at(name::user3)]; }
```

Here is the caller graph for this function:



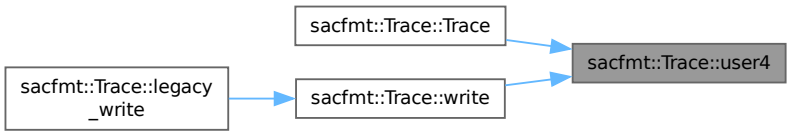
11.3.3.226 user3() [2/2]

```
void sacfmt::Trace::user3 (
            float input )  [noexcept]
01250                                       {
01251   floats[sac_map.at(name::user3)] = input;
01252 }
```

11.3.3.227 user4() [1/2]

```
float sacfmt::Trace::user4 ( ) const  [noexcept]
01026 { return floats[sac_map.at(name::user4)]; }
```

Here is the caller graph for this function:



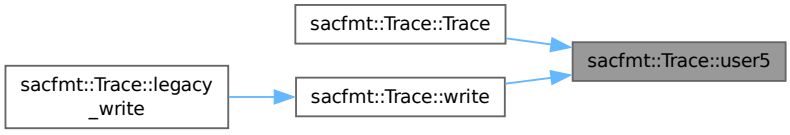
11.3.3.228 user4() [2/2]

```
void sacfmt::Trace::user4 (
            float input )  [noexcept]
01253                                   {
01254   floats[sac_map.at(name::user4)] = input;
01255 }
```

11.3.3.229 user5() [1/2]

```
float sacfmt::Trace::user5 ( ) const  [noexcept]
01027 { return floats[sac_map.at(name::user5)]; }
```

Here is the caller graph for this function:



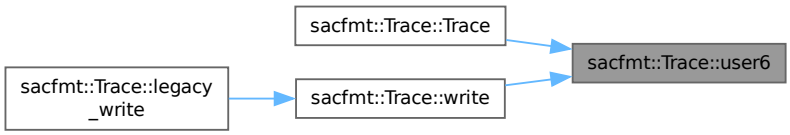
11.3.3.230 user5() [2/2]

```
void sacfmt::Trace::user5 (
            float input )  [noexcept]
01256                                   {
01257   floats[sac_map.at(name::user5)] = input;
01258 }
```

11.3.3.231 user6() [1/2]

```
float sacfmt::Trace::user6 ( ) const  [noexcept]
01028 { return floats[sac_map.at(name::user6)]; }
```

Here is the caller graph for this function:



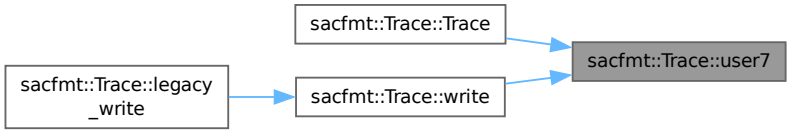
11.3.3.232 user6() [2/2]

```
void sacfmt::Trace::user6 (
            float input )  [noexcept]
01259                                       {
01260   floats[sac_map.at(name::user6)] = input;
01261 }
```

11.3.3.233 user7() [1/2]

```
float sacfmt::Trace::user7 ( ) const  [noexcept]
01029 { return floats[sac_map.at(name::user7)]; }
```

Here is the caller graph for this function:



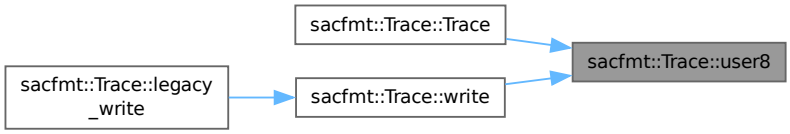
11.3.3.234 user7() [2/2]

```
void sacfmt::Trace::user7 (
            float input )  [noexcept]
01262                                       {
01263   floats[sac_map.at(name::user7)] = input;
01264 }
```

11.3.3.235 user8() [1/2]

```
float sacfmt::Trace::user8 ( ) const  [noexcept]
01030 { return floats[sac_map.at(name::user8)]; }
```

Here is the caller graph for this function:



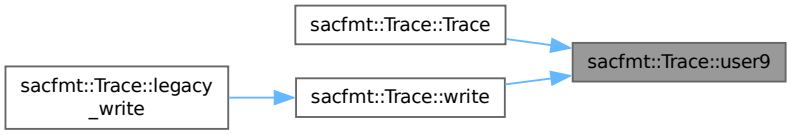
11.3.3.236 user8() [2/2]

```
void sacfmt::Trace::user8 (
            float input )  [noexcept]
01265                                       {
01266   floats[sac_map.at(name::user8)] = input;
01267 }
```

11.3.3.237 user9() [1/2]

```
float sacfmt::Trace::user9 ( ) const  [noexcept]
01031 { return floats[sac_map.at(name::user9)]; }
```

Here is the caller graph for this function:



11.3.3.238 user9() [2/2]

```
void sacfmt::Trace::user9 (
            float input )  [noexcept]
01268                                       {
01269   floats[sac_map.at(name::user9)] = input;
01270 }
```

11.3.3.239 write()

```
void sacfmt::Trace::write (
            const std::filesystem::path & path,
            bool legacy = false ) const
```

Binary SAC-file writer.

Parameters

in	*path*	std::filesystem::path SAC-file to write.
in	*legacy*	bool Legacy-write flag (default false = v7, true = v6).

Exceptions

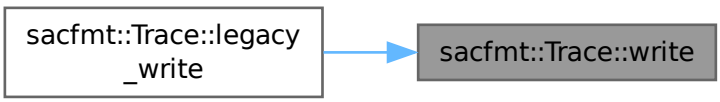
io_error	If the file cannot be written (bad path or bad permissions).
std::exception	Other unwritable issues (not enough space, disk failure, etc.).

```
01959                                                                                {
01960   std::ofstream file(path, std::ios::binary | std::ios::out | std::ios::trunc);
01961   if (!file) {
01962     throw io_error(path.string() + " cannot be opened to write.");
01963   }
01964   const int header_version{legacy ? old_hdr_version : modern_hdr_version};
01965   write_words(&file, convert_to_word(static_cast<float>(delta())));
01966   write_words(&file, convert_to_word(depmin()));
01967   write_words(&file, convert_to_word(depmax()));
01968   // Fill 'unused'
01969   write_words(&file, convert_to_word(depmax()));
01970   write_words(&file, convert_to_word(odelta()));
01971   write_words(&file, convert_to_word(static_cast<float>(b())));
01972   write_words(&file, convert_to_word(static_cast<float>(e())));
01973   write_words(&file, convert_to_word(static_cast<float>(o())));
01974   write_words(&file, convert_to_word(static_cast<float>(a())));
01975   // Fill 'internal'
01976   write_words(&file, convert_to_word(depmin()));
01977   write_words(&file, convert_to_word(static_cast<float>(t0())));
01978   write_words(&file, convert_to_word(static_cast<float>(t1())));
01979   write_words(&file, convert_to_word(static_cast<float>(t2())));
01980   write_words(&file, convert_to_word(static_cast<float>(t3())));
01981   write_words(&file, convert_to_word(static_cast<float>(t4())));
01982   write_words(&file, convert_to_word(static_cast<float>(t5())));
01983   write_words(&file, convert_to_word(static_cast<float>(t6())));
01984   write_words(&file, convert_to_word(static_cast<float>(t7())));
01985   write_words(&file, convert_to_word(static_cast<float>(t8())));
01986   write_words(&file, convert_to_word(static_cast<float>(t9())));
01987   write_words(&file, convert_to_word(static_cast<float>(f())));
01988   write_words(&file, convert_to_word(resp0()));
01989   write_words(&file, convert_to_word(resp1()));
01990   write_words(&file, convert_to_word(resp2()));
01991   write_words(&file, convert_to_word(resp3()));
01992   write_words(&file, convert_to_word(resp4()));
01993   write_words(&file, convert_to_word(resp5()));
01994   write_words(&file, convert_to_word(resp6()));
01995   write_words(&file, convert_to_word(resp7()));
01996   write_words(&file, convert_to_word(resp8()));
01997   write_words(&file, convert_to_word(resp9()));
01998   write_words(&file, convert_to_word(static_cast<float>(stla())));
01999   write_words(&file, convert_to_word(static_cast<float>(stlo())));
02000   write_words(&file, convert_to_word(stel()));
02001   write_words(&file, convert_to_word(stdp()));
02002   write_words(&file, convert_to_word(static_cast<float>(evla())));
02003   write_words(&file, convert_to_word(static_cast<float>(evlo())));
02004   write_words(&file, convert_to_word(evel()));
02005   write_words(&file, convert_to_word(evdp()));
02006   write_words(&file, convert_to_word(mag()));
02007   write_words(&file, convert_to_word(user0()));
02008   write_words(&file, convert_to_word(user1()));
02009   write_words(&file, convert_to_word(user2()));
02010   write_words(&file, convert_to_word(user3()));
02011   write_words(&file, convert_to_word(user4()));
02012   write_words(&file, convert_to_word(user5()));
02013   write_words(&file, convert_to_word(user6()));
02014   write_words(&file, convert_to_word(user7()));
02015   write_words(&file, convert_to_word(user8()));
02016   write_words(&file, convert_to_word(user9()));
02017   write_words(&file, convert_to_word(dist()));
02018   write_words(&file, convert_to_word(az()));
02019   write_words(&file, convert_to_word(baz()));
02020   write_words(&file, convert_to_word(gcarc()));
02021   write_words(&file, convert_to_word(static_cast<float>(sb())));
02022   write_words(&file, convert_to_word(static_cast<float>(sdelta())));
02023   write_words(&file, convert_to_word(depmen()));
02024   write_words(&file, convert_to_word(cmpaz()));
02025   write_words(&file, convert_to_word(cmpinc()));
02026   write_words(&file, convert_to_word(xminimum()));
02027   write_words(&file, convert_to_word(xmaximum()));
02028   write_words(&file, convert_to_word(yminimum()));
```

```
02029   write_words(&file, convert_to_word(ymaximum()));
02030   // Fill 'unused' (xcommon_skip_num)
02031   for (int i{0}; i < common_skip_num; ++i) {
02032     write_words(&file, convert_to_word(az()));
02033   }
02034   write_words(&file, convert_to_word(nzyear()));
02035   write_words(&file, convert_to_word(nzjday()));
02036   write_words(&file, convert_to_word(nzhour()));
02037   write_words(&file, convert_to_word(nzmin()));
02038   write_words(&file, convert_to_word(nzsec()));
02039   write_words(&file, convert_to_word(nzmsec()));
02040   write_words(&file, convert_to_word(header_version));
02041   write_words(&file, convert_to_word(norid()));
02042   write_words(&file, convert_to_word(nevid()));
02043   write_words(&file, convert_to_word(npts()));
02044   write_words(&file, convert_to_word(nsnpts()));
02045   write_words(&file, convert_to_word(nwfid()));
02046   write_words(&file, convert_to_word(nxsize()));
02047   write_words(&file, convert_to_word(nysize()));
02048   // Fill 'unused'
02049   write_words(&file, convert_to_word(nysize()));
02050   write_words(&file, convert_to_word(iftype()));
02051   write_words(&file, convert_to_word(idep()));
02052   write_words(&file, convert_to_word(iztype()));
02053   // Fill 'unused'
02054   write_words(&file, convert_to_word(iztype()));
02055   write_words(&file, convert_to_word(iinst()));
02056   write_words(&file, convert_to_word(istreg()));
02057   write_words(&file, convert_to_word(ievreg()));
02058   write_words(&file, convert_to_word(ievtyp()));
02059   write_words(&file, convert_to_word(iqual()));
02060   write_words(&file, convert_to_word(isynth()));
02061   write_words(&file, convert_to_word(imagtyp()));
02062   write_words(&file, convert_to_word(imagsrc()));
02063   write_words(&file, convert_to_word(ibody()));
02064   // Fill 'unused' (xcommon_skip_num)
02065   for (int i{0}; i < common_skip_num; ++i) {
02066     write_words(&file, convert_to_word(ibody()));
02067   }
02068   write_words(&file, bool_to_word(leven()));
02069   write_words(&file, bool_to_word(lpspol()));
02070   write_words(&file, bool_to_word(lovrok()));
02071   write_words(&file, bool_to_word(lcalda()));
02072   // Fill 'unused'
02073   write_words(&file, bool_to_word(lcalda()));
02074   // Strings are special
02075   std::array<char, static_cast<size_t>(2) * word_length> two_words{
02076       convert_to_words<sizeof(two_words)>(kstnm(), 2)};
02077   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02078
02079   std::array<char, static_cast<size_t>(4) * word_length> four_words{
02080       convert_to_words<sizeof(four_words)>(kevnm(), 4)};
02081   write_words(&file, std::vector<char>(four_words.begin(), four_words.end()));
02082
02083   two_words = convert_to_words<sizeof(two_words)>(khole(), 2);
02084   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02085
02086   two_words = convert_to_words<sizeof(two_words)>(ko(), 2);
02087   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02088
02089   two_words = convert_to_words<sizeof(two_words)>(ka(), 2);
02090   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02091
02092   two_words = convert_to_words<sizeof(two_words)>(kt0(), 2);
02093   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02094
02095   two_words = convert_to_words<sizeof(two_words)>(kt1(), 2);
02096   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02097
02098   two_words = convert_to_words<sizeof(two_words)>(kt2(), 2);
02099   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02100
02101   two_words = convert_to_words<sizeof(two_words)>(kt3(), 2);
02102   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02103
02104   two_words = convert_to_words<sizeof(two_words)>(kt4(), 2);
02105   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02106
02107   two_words = convert_to_words<sizeof(two_words)>(kt5(), 2);
02108   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02109
02110   two_words = convert_to_words<sizeof(two_words)>(kt6(), 2);
02111   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02112
02113   two_words = convert_to_words<sizeof(two_words)>(kt7(), 2);
02114   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02115
```

```
02116   two_words = convert_to_words<sizeof(two_words)>(kt8(), 2);
02117   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02118
02119   two_words = convert_to_words<sizeof(two_words)>(kt9(), 2);
02120   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02121
02122   two_words = convert_to_words<sizeof(two_words)>(kf(), 2);
02123   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02124
02125   two_words = convert_to_words<sizeof(two_words)>(kuser0(), 2);
02126   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02127
02128   two_words = convert_to_words<sizeof(two_words)>(kuser1(), 2);
02129   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02130
02131   two_words = convert_to_words<sizeof(two_words)>(kuser2(), 2);
02132   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02133
02134   two_words = convert_to_words<sizeof(two_words)>(kcmpnm(), 2);
02135   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02136
02137   two_words = convert_to_words<sizeof(two_words)>(knetwk(), 2);
02138   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02139
02140   two_words = convert_to_words<sizeof(two_words)>(kdatrd(), 2);
02141   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02142
02143   two_words = convert_to_words<sizeof(two_words)>(kinst(), 2);
02144   write_words(&file, std::vector<char>(two_words.begin(), two_words.end()));
02145   // Data
02146   for (double dub : data1()) [[likely]] {
02147     write_words(&file, convert_to_word(static_cast<float>(dub)));
02148   }
02149   if (!leven() || (iftype() > 1)) {
02150     for (double dub : data2()) {
02151       write_words(&file, convert_to_word(static_cast<float>(dub)));
02152     }
02153   }
02154   if (header_version == modern_hdr_version) {
02155     // Write footer
02156     write_words(&file, convert_to_word(delta()));
02157     write_words(&file, convert_to_word(b()));
02158     write_words(&file, convert_to_word(e()));
02159     write_words(&file, convert_to_word(o()));
02160     write_words(&file, convert_to_word(a()));
02161     write_words(&file, convert_to_word(t0()));
02162     write_words(&file, convert_to_word(t1()));
02163     write_words(&file, convert_to_word(t2()));
02164     write_words(&file, convert_to_word(t3()));
02165     write_words(&file, convert_to_word(t4()));
02166     write_words(&file, convert_to_word(t5()));
02167     write_words(&file, convert_to_word(t6()));
02168     write_words(&file, convert_to_word(t7()));
02169     write_words(&file, convert_to_word(t8()));
02170     write_words(&file, convert_to_word(t9()));
02171     write_words(&file, convert_to_word(f()));
02172     write_words(&file, convert_to_word(evlo()));
02173     write_words(&file, convert_to_word(evla()));
02174     write_words(&file, convert_to_word(stlo()));
02175     write_words(&file, convert_to_word(stla()));
02176     write_words(&file, convert_to_word(sb()));
02177     write_words(&file, convert_to_word(sdelta()));
02178   }
02179   file.close();
02180 }
```

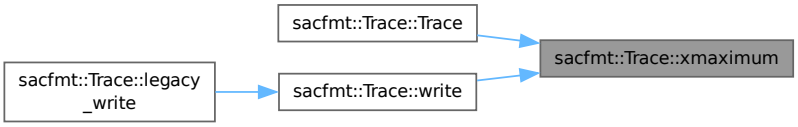
Here is the caller graph for this function:



11.3.3.240 xmaximum() [1/2]

```
float sacfmt::Trace::xmaximum ( ) const  [noexcept]
01046                                    {
01047   return floats[sac_map.at(name::xmaximum)];
01048 }
```

Here is the caller graph for this function:



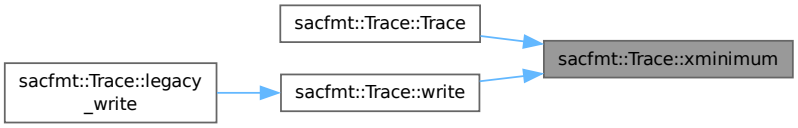
11.3.3.241 xmaximum() [2/2]

```
void sacfmt::Trace::xmaximum (
            float input )  [noexcept]
01295                                          {
01296   floats[sac_map.at(name::xmaximum)] = input;
01297 }
```

11.3.3.242 xminimum() [1/2]

```
float sacfmt::Trace::xminimum ( ) const  [noexcept]
01043                                    {
01044   return floats[sac_map.at(name::xminimum)];
01045 }
```

Here is the caller graph for this function:



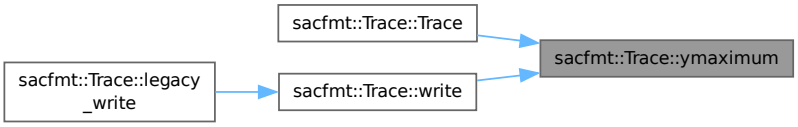
11.3.3.243 xminimum() [2/2]

```
void sacfmt::Trace::xminimum (
            float input )  [noexcept]
01292                                          {
01293   floats[sac_map.at(name::xminimum)] = input;
01294 }
```

11.3.3.244 ymaximum() [1/2]

```
float sacfmt::Trace::ymaximum ( ) const  [noexcept]
01052                                    {
01053   return floats[sac_map.at(name::ymaximum)];
01054 }
```

Here is the caller graph for this function:



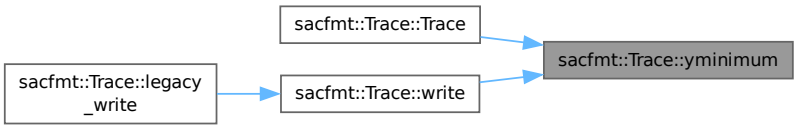
11.3.3.245 ymaximum() [2/2]

```
void sacfmt::Trace::ymaximum (
            float input )  [noexcept]
01301                                          {
01302   floats[sac_map.at(name::ymaximum)] = input;
01303 }
```

11.3.3.246 yminimum() [1/2]

```
float sacfmt::Trace::yminimum ( ) const  [noexcept]
01049                                    {
01050   return floats[sac_map.at(name::yminimum)];
01051 }
```

Here is the caller graph for this function:



11.3.3.247 yminimum() [2/2]

```
void sacfmt::Trace::yminimum (
            float input )  [noexcept]
01298                                          {
01299   floats[sac_map.at(name::yminimum)] = input;
01300 }
```

11.3.4 Member Data Documentation

11.3.4.1 bools

```
std::array<bool, num_bool> sacfmt::Trace::bools {}  [private]
```

Boolean storage array.
01323 {};

11.3.4.2 data

```
std::array<std::vector<double>, num_data> sacfmt::Trace::data {}  [private]
```

std::vector<double> storage array.
01328 {};

11.3.4.3 doubles

```
std::array<double, num_double> sacfmt::Trace::doubles {}  [private]
```

Double storage array.
01319 {};

11.3.4.4 floats

```
std::array<float, num_float> sacfmt::Trace::floats {}  [private]
```

Float storage array.
01317 {};

11.3.4.5 ints

```
std::array<int, num_int> sacfmt::Trace::ints {}  [private]
```

Integer storage array.
01321 {};

11.3.4.6 strings

```
std::array<std::string, num_string> sacfmt::Trace::strings {}  [private]
```

String storage array.
01325 {};

The documentation for this class was generated from the following files:

- include/sac-format/sac_format.hpp
- src/sac_format.cpp

11.4 sacfmt::bitset_type::uint< nbits > Struct Template Reference

Ensure type-safety for conversions between floats/doubles and bitsets.

`#include <sac_format.hpp>`

11.4.1 Detailed Description

template<unsigned nbits>
struct sacfmt::bitset_type::uint< nbits >

Ensure type-safety for conversions between floats/doubles and bitsets.

The documentation for this struct was generated from the following file:

- include/sac-format/sac_format.hpp

11.5 sacfmt::bitset_type::uint< 4 *bits_per_byte > Struct Reference

One-word (floats).

`#include <sac_format.hpp>`

Public Types

- using type = uint32_t

11.5.1 Detailed Description

One-word (floats).

11.5.2 Member Typedef Documentation

11.5.2.1 type

```
using sacfmt::bitset_type::uint< 4 *bits_per_byte >::type = uint32_t
```

The documentation for this struct was generated from the following file:

- include/sac-format/sac_format.hpp

11.6 sacfmt::bitset_type::uint< bytes *bits_per_byte > Struct Reference

Two-words (doubles)

`#include <sac_format.hpp>`

Public Types

- using type = uint64_t

11.6.1 Detailed Description

Two-words (doubles)

11.6.2 Member Typedef Documentation

11.6.2.1 type

```
using sacfmt::bitset_type::uint< bytes *bits_per_byte >::type = uint64_t
```

The documentation for this struct was generated from the following file:

- include/sac-format/sac_format.hpp

11.7 sacfmt::word_pair< T > Struct Template Reference

Struct containing a pair of words.

`#include <sac_format.hpp>`

Public Attributes

- T first {}
  First 'word' in the pair.
- T second {}
  Second 'word' in the pair.

11.7.1 Detailed Description

template<typename T>
struct sacfmt::word_pair< T >

Struct containing a pair of words.

Prevents bug-prone word-swapping in functions that use a pair of words.

These are not necessarily single words, it could be a pair of word_one or a pair of word_two.

11.7.2 Member Data Documentation

11.7.2.1 first

```
template<typename T >
T sacfmt::word_pair< T >::first {}
```

First 'word' in the pair.
00192 {};

11.7.2.2 second

```
template<typename T >
T sacfmt::word_pair< T >::second {}
```

Second 'word' in the pair.
00193 {};

The documentation for this struct was generated from the following file:

- include/sac-format/sac_format.hpp

Index

a
    sacfmt, 54
    sacfmt::Trace, 112
ascii_space
    sacfmt, 94
az
    sacfmt, 53
    sacfmt::Trace, 113
azimuth
    sacfmt, 58

b
    sacfmt, 54
    sacfmt::Trace, 113
Basic Documentation, 17
baz
    sacfmt, 53
    sacfmt::Trace, 114
binary_to_bool
    sacfmt, 59
binary_to_double
    sacfmt, 61
binary_to_float
    sacfmt, 62
binary_to_int
    sacfmt, 62
binary_to_long_string
    sacfmt, 63
binary_to_string
    sacfmt, 64
binary_word_size
    sacfmt, 94
bits_per_byte
    sacfmt, 94
bits_string
    sacfmt, 65
bool_to_binary
    sacfmt, 66
bool_to_word
    sacfmt, 66
bools
    sacfmt::Trace, 187
Build Instructions, 39
bytes
    sacfmt::bitset_type, 100

calc_az
    sacfmt::Trace, 114
calc_baz
    sacfmt::Trace, 115
calc_dist
    sacfmt::Trace, 116
calc_gcarc
    sacfmt::Trace, 117
calc_geometry
    sacfmt::Trace, 118
char_bit
    sacfmt, 51
circle_deg
    sacfmt, 94
cmpaz
    sacfmt, 53
    sacfmt::Trace, 119
cmpinc
    sacfmt, 53
    sacfmt::Trace, 119, 120
common_skip_num
    sacfmt, 95
concat_words
    sacfmt, 67
convert_to_word
    sacfmt, 68, 69
convert_to_words
    sacfmt, 69

data
    sacfmt::Trace, 187
data1
    sacfmt, 57
    sacfmt::Trace, 120
data2
    sacfmt, 57
    sacfmt::Trace, 120, 121
data_word
    sacfmt, 95
date
    sacfmt::Trace, 121
deg_per_rad
    sacfmt, 95
degrees_to_radians
    sacfmt, 70
delta
    sacfmt, 54
    sacfmt::Trace, 122
depmax
    sacfmt, 52
    sacfmt::Trace, 122, 123
depmen
    sacfmt, 53
    sacfmt::Trace, 123

depmin
sacfmt, 52
sacfmt::Trace, 123, 124
dist
sacfmt, 53
sacfmt::Trace, 124
double_to_binary
sacfmt, 70
doubles
sacfmt::Trace, 187

e
sacfmt, 54
sacfmt::Trace, 124, 125
earth_radius
sacfmt, 95
equal_within_tolerance
sacfmt, 71
evdp
sacfmt, 53
sacfmt::Trace, 125
evel
sacfmt, 53
sacfmt::Trace, 125, 126
evla
sacfmt, 54
sacfmt::Trace, 126
evlo
sacfmt, 54
sacfmt::Trace, 127

f
sacfmt, 54
sacfmt::Trace, 128
f_eps
sacfmt, 95
first
sacfmt::word_pair< T >, 190
float_to_binary
sacfmt, 72
floats
sacfmt::Trace, 187
frequency
sacfmt::Trace, 128

gcarc
sacfmt, 53, 73
sacfmt::Trace, 129
geometry_set
sacfmt::Trace, 129

ibody
sacfmt, 56
sacfmt::Trace, 130, 131
idep
sacfmt, 55
sacfmt::Trace, 131
ievreg
sacfmt, 55
sacfmt::Trace, 131, 132
ievtyp
sacfmt, 55
sacfmt::Trace, 132
iftype
sacfmt, 55
sacfmt::Trace, 132, 133
iinst
sacfmt, 55
sacfmt::Trace, 133
imagsrc
sacfmt, 56
sacfmt::Trace, 133, 134
imagtyp
sacfmt, 55
sacfmt::Trace, 134
Installation, 3
int_to_binary
sacfmt, 74
Introduction, 1
ints
sacfmt::Trace, 187
io_error
sacfmt::io_error, 102
iqual
sacfmt, 55
sacfmt::Trace, 134, 135
istreg
sacfmt, 55
sacfmt::Trace, 135
isynth
sacfmt, 55
sacfmt::Trace, 135, 136
iztype
sacfmt, 55
sacfmt::Trace, 136

ka
sacfmt, 56
sacfmt::Trace, 136, 137
kcmpnm
sacfmt, 56
sacfmt::Trace, 137
kdatrd
sacfmt, 56
sacfmt::Trace, 137, 138
kevnm
sacfmt, 56
sacfmt::Trace, 138
kf
sacfmt, 56
sacfmt::Trace, 138, 139
khole
sacfmt, 56
sacfmt::Trace, 139
kinst
sacfmt, 57
sacfmt::Trace, 139, 140
knetwk

sacfmt, 56
sacfmt::Trace, 140
ko
sacfmt, 56
sacfmt::Trace, 140, 141
kstnm
sacfmt, 56
sacfmt::Trace, 141
kt0
sacfmt, 56
sacfmt::Trace, 141, 142
kt1
sacfmt, 56
sacfmt::Trace, 142
kt2
sacfmt, 56
sacfmt::Trace, 142, 143
kt3
sacfmt, 56
sacfmt::Trace, 143
kt4
sacfmt, 56
sacfmt::Trace, 143, 144
kt5
sacfmt, 56
sacfmt::Trace, 144
kt6
sacfmt, 56
sacfmt::Trace, 144, 145
kt7
sacfmt, 56
sacfmt::Trace, 145
kt8
sacfmt, 56
sacfmt::Trace, 145, 146
kt9
sacfmt, 56
sacfmt::Trace, 146
kuser0
sacfmt, 56
sacfmt::Trace, 146, 147
kuser1
sacfmt, 56
sacfmt::Trace, 147
kuser2
sacfmt, 56
sacfmt::Trace, 147, 148

lcalda
sacfmt, 56
sacfmt::Trace, 148
legacy_write
sacfmt::Trace, 148
leven
sacfmt, 56
sacfmt::Trace, 150, 151
limit_180
sacfmt, 75
limit_360
sacfmt, 76
limit_90
sacfmt, 77
long_string_to_binary
sacfmt, 77
lovrok
sacfmt, 56
sacfmt::Trace, 151
lpspol
sacfmt, 56
sacfmt::Trace, 151, 152

mag
sacfmt, 53
sacfmt::Trace, 152
message
sacfmt::io_error, 103
modern_hdr_version
sacfmt, 95

name
sacfmt, 52
nevid
sacfmt, 55
sacfmt::Trace, 152, 153
norid
sacfmt, 55
sacfmt::Trace, 153
npts
sacfmt, 55
sacfmt::Trace, 153, 154
nsnpts
sacfmt, 55
sacfmt::Trace, 154
num_bool
sacfmt, 95
num_data
sacfmt, 96
num_double
sacfmt, 96
num_float
sacfmt, 96
num_footer
sacfmt, 96
num_int
sacfmt, 96
num_string
sacfmt, 96
num_words
sacfmt::read_spec, 103
nvhdr
sacfmt, 55
sacfmt::Trace, 154, 155
nwfid
sacfmt, 55
sacfmt::Trace, 155
nwords_after_current
sacfmt, 78
nxsize

sacfmt, 55
sacfmt::Trace, 155, 156
nysize
sacfmt, 55
sacfmt::Trace, 156
nzhour
sacfmt, 54
sacfmt::Trace, 156, 157
nzjday
sacfmt, 54
sacfmt::Trace, 157
nzmin
sacfmt, 55
sacfmt::Trace, 157, 158
nzmsec
sacfmt, 55
sacfmt::Trace, 158
nzsec
sacfmt, 55
sacfmt::Trace, 158, 159
nzyear
sacfmt, 54
sacfmt::Trace, 159

o
sacfmt, 54
sacfmt::Trace, 159, 160
odelta
sacfmt, 52
sacfmt::Trace, 160
old_hdr_version
sacfmt, 96
operator==
sacfmt::Trace, 160

prep_string
sacfmt, 79

Quickstart, 15

rad_per_deg
sacfmt, 97
radians_to_degrees
sacfmt, 80
read_data
sacfmt, 81
read_four_words
sacfmt, 82
read_two_words
sacfmt, 83
read_word
sacfmt, 84
remove_leading_spaces
sacfmt, 84
remove_trailing_spaces
sacfmt, 85
resize_data
sacfmt::Trace, 161
resize_data1
sacfmt::Trace, 161
resize_data2
sacfmt::Trace, 162
resp0
sacfmt, 52
sacfmt::Trace, 162
resp1
sacfmt, 52
sacfmt::Trace, 162, 163
resp2
sacfmt, 52
sacfmt::Trace, 163
resp3
sacfmt, 53
sacfmt::Trace, 163, 164
resp4
sacfmt, 53
sacfmt::Trace, 164
resp5
sacfmt, 53
sacfmt::Trace, 164, 165
resp6
sacfmt, 53
sacfmt::Trace, 165
resp7
sacfmt, 53
sacfmt::Trace, 165, 166
resp8
sacfmt, 53
sacfmt::Trace, 166
resp9
sacfmt, 53
sacfmt::Trace, 166, 167

SAC-file format, 27
sac_map
sacfmt, 97
sacfmt, 47
a, 54
ascii_space, 94
az, 53
azimuth, 58
b, 54
baz, 53
binary_to_bool, 59
binary_to_double, 61
binary_to_float, 62
binary_to_int, 62
binary_to_long_string, 63
binary_to_string, 64
binary_word_size, 94
bits_per_byte, 94
bits_string, 65
bool_to_binary, 66
bool_to_word, 66
char_bit, 51
circle_deg, 94
cmpaz, 53
cmpinc, 53

common_skip_num, 95
concat_words, 67
convert_to_word, 68, 69
convert_to_words, 69
data1, 57
data2, 57
data_word, 95
deg_per_rad, 95
degrees_to_radians, 70
delta, 54
depmax, 52
depmen, 53
depmin, 52
dist, 53
double_to_binary, 70
e, 54
earth_radius, 95
equal_within_tolerance, 71
evdp, 53
evel, 53
evla, 54
evlo, 54
f, 54
f_eps, 95
float_to_binary, 72
gcarc, 53, 73
ibody, 56
idep, 55
ievreg, 55
ievtyp, 55
iftype, 55
iinst, 55
imagsrc, 56
imagtyp, 55
int_to_binary, 74
iqual, 55
istreg, 55
isynth, 55
iztype, 55
ka, 56
kcmpnm, 56
kdatrd, 56
kevnm, 56
kf, 56
khole, 56
kinst, 57
knetwk, 56
ko, 56
kstnm, 56
kt0, 56
kt1, 56
kt2, 56
kt3, 56
kt4, 56
kt5, 56
kt6, 56
kt7, 56
kt8, 56
kt9, 56
kuser0, 56
kuser1, 56
kuser2, 56
lcalda, 56
leven, 56
limit_180, 75
limit_360, 76
limit_90, 77
long_string_to_binary, 77
lovrok, 56
lpspol, 56
mag, 53
modern_hdr_version, 95
name, 52
nevid, 55
norid, 55
npts, 55
nsnpts, 55
num_bool, 95
num_data, 96
num_double, 96
num_float, 96
num_footer, 96
num_int, 96
num_string, 96
nvhdr, 55
nwfid, 55
nwords_after_current, 78
nxsize, 55
nysize, 55
nzhour, 54
nzjday, 54
nzmin, 55
nzmsec, 55
nzsec, 55
nzyear, 54
o, 54
odelta, 52
old_hdr_version, 96
prep_string, 79
rad_per_deg, 97
radians_to_degrees, 80
read_data, 81
read_four_words, 82
read_two_words, 83
read_word, 84
remove_leading_spaces, 84
remove_trailing_spaces, 85
resp0, 52
resp1, 52
resp2, 52
resp3, 53
resp4, 53
resp5, 53
resp6, 53
resp7, 53
resp8, 53

resp9, 53
sac_map, 97
safe_to_finish_reading, 86
safe_to_read_data, 87
safe_to_read_footer, 88
safe_to_read_header, 89
sb, 54
sdelta, 54
stdp, 53
stel, 53
stla, 54
stlo, 54
string_bits, 90
string_cleaning, 91
string_to_binary, 91
t0, 54
t1, 54
t2, 54
t3, 54
t4, 54
t5, 54
t6, 54
t7, 54
t8, 54
t9, 54
uint_to_binary, 92
unset_bool, 98
unset_double, 98
unset_float, 99
unset_int, 99
unset_word, 99
unsigned_int, 51
user0, 53
user1, 53
user2, 53
user3, 53
user4, 53
user5, 53
user6, 53
user7, 53
user8, 53
user9, 53
word_four, 52
word_length, 99
word_one, 52
word_position, 93
word_two, 52
write_words, 93
xmaximum, 53
xminimum, 53
ymaximum, 54
yminimum, 54
sacfmt::bitset_type, 99
bytes, 100
sacfmt::bitset_type::uint< 4 *bits_per_byte >, 188
type, 188
sacfmt::bitset_type::uint< bytes *bits_per_byte >, 189
type, 189
sacfmt::bitset_type::uint< nbits >, 188
sacfmt::io_error, 101
io_error, 102
message, 103
what, 102
sacfmt::read_spec, 103
num_words, 103
start_word, 103
sacfmt::Trace, 104
a, 112
az, 113
b, 113
baz, 114
bools, 187
calc_az, 114
calc_baz, 115
calc_dist, 116
calc_gcarc, 117
calc_geometry, 118
cmpaz, 119
cmpinc, 119, 120
data, 187
data1, 120
data2, 120, 121
date, 121
delta, 122
depmax, 122, 123
depmen, 123
depmin, 123, 124
dist, 124
doubles, 187
e, 124, 125
evdp, 125
evel, 125, 126
evla, 126
evlo, 127
f, 128
floats, 187
frequency, 128
gcarc, 129
geometry_set, 129
ibody, 130, 131
idep, 131
ievreg, 131, 132
ievtyp, 132
iftype, 132, 133
iinst, 133
imagsrc, 133, 134
imagtyp, 134
ints, 187
iqual, 134, 135
istreg, 135
isynth, 135, 136
iztype, 136
ka, 136, 137
kcmpnm, 137
kdatrd, 137, 138
kevnm, 138

kf, 138, 139
khole, 139
kinst, 139, 140
knetwk, 140
ko, 140, 141
kstnm, 141
kt0, 141, 142
kt1, 142
kt2, 142, 143
kt3, 143
kt4, 143, 144
kt5, 144
kt6, 144, 145
kt7, 145
kt8, 145, 146
kt9, 146
kuser0, 146, 147
kuser1, 147
kuser2, 147, 148
lcalda, 148
legacy_write, 148
leven, 150, 151
lovrok, 151
lpspol, 151, 152
mag, 152
nevid, 152, 153
norid, 153
npts, 153, 154
nsnpts, 154
nvhdr, 154, 155
nwfid, 155
nxsize, 155, 156
nysize, 156
nzhour, 156, 157
nzjday, 157
nzmin, 157, 158
nzmsec, 158
nzsec, 158, 159
nzyear, 159
o, 159, 160
odelta, 160
operator==, 160
resize_data, 161
resize_data1, 161
resize_data2, 162
resp0, 162
resp1, 162, 163
resp2, 163
resp3, 163, 164
resp4, 164
resp5, 164, 165
resp6, 165
resp7, 165, 166
resp8, 166
resp9, 166, 167
sb, 167
sdelta, 167, 168
stdp, 168
stel, 168, 169
stla, 169
stlo, 170
strings, 187
t0, 171
t1, 171
t2, 172
t3, 172
t4, 173
t5, 173
t6, 174
t7, 174
t8, 175
t9, 175
time, 176
Trace, 109
user0, 176, 177
user1, 177
user2, 177, 178
user3, 178
user4, 178, 179
user5, 179
user6, 179, 180
user7, 180
user8, 180, 181
user9, 181
write, 181
xmaximum, 184, 185
xminimum, 185
ymaximum, 185, 186
yminimum, 186
sacfmt::word_pair< T >, 189
first, 190
second, 190
safe_to_finish_reading
sacfmt, 86
safe_to_read_data
sacfmt, 87
safe_to_read_footer
sacfmt, 88
safe_to_read_header
sacfmt, 89
sb
sacfmt, 54
sacfmt::Trace, 167
sdelta
sacfmt, 54
sacfmt::Trace, 167, 168
second
sacfmt::word_pair< T >, 190
start_word
sacfmt::read_spec, 103
stdp
sacfmt, 53
sacfmt::Trace, 168
stel
sacfmt, 53
sacfmt::Trace, 168, 169

stla
sacfmt, 54
sacfmt::Trace, 169
stlo
sacfmt, 54
sacfmt::Trace, 170
string_bits
sacfmt, 90
string_cleaning
sacfmt, 91
string_to_binary
sacfmt, 91
strings
sacfmt::Trace, 187

t0
sacfmt, 54
sacfmt::Trace, 171
t1
sacfmt, 54
sacfmt::Trace, 171
t2
sacfmt, 54
sacfmt::Trace, 172
t3
sacfmt, 54
sacfmt::Trace, 172
t4
sacfmt, 54
sacfmt::Trace, 173
t5
sacfmt, 54
sacfmt::Trace, 173
t6
sacfmt, 54
sacfmt::Trace, 174
t7
sacfmt, 54
sacfmt::Trace, 174
t8
sacfmt, 54
sacfmt::Trace, 175
t9
sacfmt, 54
sacfmt::Trace, 175
time
sacfmt::Trace, 176
Trace
sacfmt::Trace, 109
type
sacfmt::bitset_type::uint< 4 *bits_per_byte >, 188
sacfmt::bitset_type::uint< bytes *bits_per_byte >, 189

uint_to_binary
sacfmt, 92
unset_bool
sacfmt, 98
unset_double
sacfmt, 98
unset_float
sacfmt, 99
unset_int
sacfmt, 99
unset_word
sacfmt, 99
unsigned_int
sacfmt, 51
user0
sacfmt, 53
sacfmt::Trace, 176, 177
user1
sacfmt, 53
sacfmt::Trace, 177
user2
sacfmt, 53
sacfmt::Trace, 177, 178
user3
sacfmt, 53
sacfmt::Trace, 178
user4
sacfmt, 53
sacfmt::Trace, 178, 179
user5
sacfmt, 53
sacfmt::Trace, 179
user6
sacfmt, 53
sacfmt::Trace, 179, 180
user7
sacfmt, 53
sacfmt::Trace, 180
user8
sacfmt, 53
sacfmt::Trace, 180, 181
user9
sacfmt, 53
sacfmt::Trace, 181

what
sacfmt::io_error, 102
word_four
sacfmt, 52
word_length
sacfmt, 99
word_one
sacfmt, 52
word_position
sacfmt, 93
word_two
sacfmt, 52
write
sacfmt::Trace, 181
write_words
sacfmt, 93

xmaximum
sacfmt, 53

sacfmt::Trace, 184, 185
xminimum
sacfmt, 53
sacfmt::Trace, 185

ymaximum
sacfmt, 54
sacfmt::Trace, 185, 186
yminimum
sacfmt, 54
sacfmt::Trace, 186