

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	28
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	29
5.1.11 user(0–9)	29
5.1.12 dist	29
5.1.13 az	29
5.1.14 baz	29
5.1.15 gcarc	29
5.1.16 cmpaz	29
5.1.17 cmpinc	30
5.1.18 xminimum	30
5.1.19 xmaximum	30
5.1.20 yminimum	30
5.1.21 ymaximum	30
5.2 Double (22)	30
5.2.1 delta	31
5.2.2 b	31
5.2.3 e	31
5.2.4 o	31
5.2.5 a	31
5.2.6 t(0–9)	31
5.2.7 f	31
5.2.8 stla	32
5.2.9 stlo	32
5.2.10 evla	32
5.2.11 evlo	32
5.2.12 sb	32

5.2.13 sdelta	32
5.3 Integer (26)	32
5.3.1 nzyear	33
5.3.2 nzjday	33
5.3.3 nzhour	33
5.3.4 nzmin	33
5.3.5 nzsec	33
5.3.6 nzmsec	33
5.3.7 nvhdr	33
5.3.8 norid	34
5.3.9 nevid	34
5.3.10 npts	34
5.3.11 nsnpts	34
5.3.12 nwfid	34
5.3.13 nxsize	34
5.3.14 nysize	34
5.3.15 iftype	34
5.3.16 idep	35
5.3.17 iztype	35
5.3.18 iinst	35
5.3.19 istreg	36
5.3.20 ievreg	36
5.3.21 ievtyp	36
5.3.22 iqual	37
5.3.23 isynth	37
5.3.24 imagtyp	37
5.3.25 imagsrc	37
5.3.26 ibody	38
5.4 Boolean (4)	38
5.4.1 leven	38
5.4.2 lpspol	38
5.4.3 lovrok	39
5.4.4 lcalda	39
5.5 String (23)	39
5.5.1 kstnm	39
5.5.2 kevnm	39
5.5.3 khole	39
5.5.4 ko	39
5.5.5 ka	40
5.5.6 kt(0–9)	40
5.5.7 kf	40
5.5.8 kuser(0–2)	40

5.5.9 kcmpnm	40
5.5.10 knetwk	40
5.5.11 kdatrd	40
5.5.12 kinst	41
5.6 Data (2)	41
5.6.1 data1	41
5.6.2 data2	41
6 Build Instructions	43
6.1 Dependencies	43
6.1.1 Automatic (CMake)	43
6.1.2 Manual	43
6.1.2.1 macOS and Linux	43
6.2 Building	43
6.2.1 GCC	43
6.2.2 Clang	44
6.2.3 MSVC	44
7 Namespace Index	45
7.1 Namespace List	45
8 Hierarchical Index	47
8.1 Class Hierarchy	47
9 Class Index	49
9.1 Class List	49
10 Namespace Documentation	51
10.1 sacfmt Namespace Reference	51
10.1.1 Detailed Description	55
10.1.2 Typedef Documentation	55
10.1.2.1 char_bit	55
10.1.2.2 unsigned_int	56
10.1.2.3 word_four	56
10.1.2.4 word_one	56
10.1.2.5 word_two	56
10.1.3 Enumeration Type Documentation	56
10.1.3.1 name	56
10.1.4 Function Documentation	62
10.1.4.1 azimuth()	62
10.1.4.2 binary_to_bool()	63
10.1.4.3 binary_to_double()	64
10.1.4.4 binary_to_float()	65
10.1.4.5 binary_to_int()	65

10.1.4.6	<code>binary_to_long_string()</code>	66
10.1.4.7	<code>binary_to_string()</code>	67
10.1.4.8	<code>bits_string()</code>	68
10.1.4.9	<code>bool_to_binary()</code>	68
10.1.4.10	<code>bool_to_word()</code>	69
10.1.4.11	<code>concat_words()</code> [1/2]	69
10.1.4.12	<code>concat_words()</code> [2/2]	70
10.1.4.13	<code>convert_to_word()</code> [1/4]	70
10.1.4.14	<code>convert_to_word()</code> [2/4]	72
10.1.4.15	<code>convert_to_word()</code> [3/4]	72
10.1.4.16	<code>convert_to_word()</code> [4/4]	72
10.1.4.17	<code>convert_to_words()</code> [1/2]	73
10.1.4.18	<code>convert_to_words()</code> [2/2]	73
10.1.4.19	<code>degrees_to_radians()</code>	74
10.1.4.20	<code>double_to_binary()</code>	74
10.1.4.21	<code>equal_within_tolerance()</code> [1/2]	75
10.1.4.22	<code>equal_within_tolerance()</code> [2/2]	75
10.1.4.23	<code>float_to_binary()</code>	76
10.1.4.24	<code>gcarc()</code>	77
10.1.4.25	<code>int_to_binary()</code>	78
10.1.4.26	<code>limit_180()</code>	78
10.1.4.27	<code>limit_360()</code>	79
10.1.4.28	<code>limit_90()</code>	80
10.1.4.29	<code>long_string_to_binary()</code>	81
10.1.4.30	<code>nwords_after_current()</code>	82
10.1.4.31	<code>prep_string()</code>	83
10.1.4.32	<code>radians_to_degrees()</code>	83
10.1.4.33	<code>read_data()</code>	84
10.1.4.34	<code>read_four_words()</code>	85
10.1.4.35	<code>read_two_words()</code>	86
10.1.4.36	<code>read_word()</code>	87
10.1.4.37	<code>remove_leading_spaces()</code>	88
10.1.4.38	<code>remove_trailing_spaces()</code>	89
10.1.4.39	<code>safe_to_finish_reading()</code>	89
10.1.4.40	<code>safe_to_read_data()</code>	90
10.1.4.41	<code>safe_to_read_footer()</code>	91
10.1.4.42	<code>safe_to_read_header()</code>	92
10.1.4.43	<code>string_bits()</code>	93
10.1.4.44	<code>string_cleaning()</code>	94
10.1.4.45	<code>string_to_binary()</code>	95
10.1.4.46	<code>uint_to_binary()</code>	95
10.1.4.47	<code>word_position()</code>	96

10.1.4.48 write_words()	97
10.1.5 Variable Documentation	98
10.1.5.1 ascii_space	98
10.1.5.2 binary_word_size	98
10.1.5.3 bits_per_byte	98
10.1.5.4 circle_deg	98
10.1.5.5 common_skip_num	99
10.1.5.6 data_word	99
10.1.5.7 deg_per_rad	99
10.1.5.8 earth_radius	99
10.1.5.9 f_eps	99
10.1.5.10 modern_hdr_version	99
10.1.5.11 num_bool	99
10.1.5.12 num_data	100
10.1.5.13 num_double	100
10.1.5.14 num_float	100
10.1.5.15 num_footer	100
10.1.5.16 num_int	100
10.1.5.17 num_string	100
10.1.5.18 old_hdr_version	100
10.1.5.19 rad_per_deg	101
10.1.5.20 sac_map	101
10.1.5.21 unset_bool	102
10.1.5.22 unset_double	102
10.1.5.23 unset_float	103
10.1.5.24 unset_int	103
10.1.5.25 unset_word	103
10.1.5.26 word_length	103
10.2 sacfmt::bitset_type Namespace Reference	103
10.2.1 Detailed Description	103
10.2.2 Variable Documentation	103
10.2.2.1 bytes	103
11 Class Documentation	105
11.1 sacfmt::coord Class Reference	105
11.1.1 Detailed Description	105
11.1.2 Constructor & Destructor Documentation	106
11.1.2.1 coord() [1/2]	106
11.1.2.2 coord() [2/2]	106
11.1.3 Member Function Documentation	106
11.1.3.1 degrees() [1/2]	106
11.1.3.2 degrees() [2/2]	107

11.1.3.3 radians() [1/2]	107
11.1.3.4 radians() [2/2]	107
11.1.4 Member Data Documentation	108
11.1.4.1 deg	108
11.1.4.2 rad	108
11.2 sacfmt::io_error Class Reference	108
11.2.1 Detailed Description	109
11.2.2 Constructor & Destructor Documentation	109
11.2.2.1 io_error()	109
11.2.3 Member Function Documentation	110
11.2.3.1 what()	110
11.2.4 Member Data Documentation	110
11.2.4.1 message	110
11.3 sacfmt::point Struct Reference	110
11.3.1 Detailed Description	111
11.3.2 Constructor & Destructor Documentation	111
11.3.2.1 point()	111
11.3.3 Member Data Documentation	111
11.3.3.1 latitude	111
11.3.3.2 longitude	111
11.4 sacfmt::read_spec Struct Reference	112
11.4.1 Detailed Description	112
11.4.2 Member Data Documentation	112
11.4.2.1 num_words	112
11.4.2.2 start_word	112
11.5 sacfmt::Trace Class Reference	112
11.5.1 Detailed Description	119
11.5.2 Constructor & Destructor Documentation	120
11.5.2.1 Trace() [1/2]	120
11.5.2.2 Trace() [2/2]	120
11.5.3 Member Function Documentation	121
11.5.3.1 a() [1/2]	121
11.5.3.2 a() [2/2]	122
11.5.3.3 az() [1/2]	122
11.5.3.4 az() [2/2]	122
11.5.3.5 b() [1/2]	123
11.5.3.6 b() [2/2]	123
11.5.3.7 baz() [1/2]	123
11.5.3.8 baz() [2/2]	123
11.5.3.9 calc_az()	124
11.5.3.10 calc_baz()	124
11.5.3.11 calc_dist()	125

11.5.3.12 <code>calc_gcarc()</code>	126
11.5.3.13 <code>calc_geometry()</code>	127
11.5.3.14 <code>cmpaz()</code> [1/2]	128
11.5.3.15 <code>cmpaz()</code> [2/2]	128
11.5.3.16 <code>cmpinc()</code> [1/2]	128
11.5.3.17 <code>cmpinc()</code> [2/2]	128
11.5.3.18 <code>data1()</code> [1/2]	129
11.5.3.19 <code>data1()</code> [2/2]	129
11.5.3.20 <code>data2()</code> [1/2]	129
11.5.3.21 <code>data2()</code> [2/2]	130
11.5.3.22 <code>date()</code>	130
11.5.3.23 <code>delta()</code> [1/2]	131
11.5.3.24 <code>delta()</code> [2/2]	131
11.5.3.25 <code>depmax()</code> [1/2]	131
11.5.3.26 <code>depmax()</code> [2/2]	131
11.5.3.27 <code>depmen()</code> [1/2]	132
11.5.3.28 <code>depmen()</code> [2/2]	132
11.5.3.29 <code>depmin()</code> [1/2]	132
11.5.3.30 <code>depmin()</code> [2/2]	132
11.5.3.31 <code>dist()</code> [1/2]	133
11.5.3.32 <code>dist()</code> [2/2]	133
11.5.3.33 <code>e()</code> [1/2]	133
11.5.3.34 <code>e()</code> [2/2]	133
11.5.3.35 <code>evdp()</code> [1/2]	134
11.5.3.36 <code>evdp()</code> [2/2]	134
11.5.3.37 <code>evel()</code> [1/2]	134
11.5.3.38 <code>evel()</code> [2/2]	134
11.5.3.39 <code>event_location()</code>	135
11.5.3.40 <code>evla()</code> [1/2]	135
11.5.3.41 <code>evla()</code> [2/2]	136
11.5.3.42 <code>evlo()</code> [1/2]	136
11.5.3.43 <code>evlo()</code> [2/2]	137
11.5.3.44 <code>f()</code> [1/2]	137
11.5.3.45 <code>f()</code> [2/2]	138
11.5.3.46 <code>frequency()</code>	138
11.5.3.47 <code>gcarc()</code> [1/2]	138
11.5.3.48 <code>gcarc()</code> [2/2]	139
11.5.3.49 <code>geometry_set()</code>	139
11.5.3.50 <code>ibody()</code> [1/2]	140
11.5.3.51 <code>ibody()</code> [2/2]	140
11.5.3.52 <code>idep()</code> [1/2]	140
11.5.3.53 <code>idep()</code> [2/2]	140

11.5.3.54	ievreg()	[1/2]	141
11.5.3.55	ievreg()	[2/2]	141
11.5.3.56	ievtyp()	[1/2]	141
11.5.3.57	ievtyp()	[2/2]	141
11.5.3.58	iftype()	[1/2]	142
11.5.3.59	iftype()	[2/2]	142
11.5.3.60	iinst()	[1/2]	142
11.5.3.61	iinst()	[2/2]	142
11.5.3.62	imgsrc()	[1/2]	143
11.5.3.63	imgsrc()	[2/2]	143
11.5.3.64	imagtyp()	[1/2]	143
11.5.3.65	imagtyp()	[2/2]	143
11.5.3.66	igual()	[1/2]	144
11.5.3.67	igual()	[2/2]	144
11.5.3.68	istreg()	[1/2]	144
11.5.3.69	istreg()	[2/2]	144
11.5.3.70	isynth()	[1/2]	145
11.5.3.71	isynth()	[2/2]	145
11.5.3.72	iztype()	[1/2]	145
11.5.3.73	iztype()	[2/2]	145
11.5.3.74	ka()	[1/2]	146
11.5.3.75	ka()	[2/2]	146
11.5.3.76	kcmpnm()	[1/2]	146
11.5.3.77	kcmpnm()	[2/2]	146
11.5.3.78	kdatrd()	[1/2]	147
11.5.3.79	kdatrd()	[2/2]	147
11.5.3.80	kevnrm()	[1/2]	147
11.5.3.81	kevnrm()	[2/2]	147
11.5.3.82	kf()	[1/2]	148
11.5.3.83	kf()	[2/2]	148
11.5.3.84	khole()	[1/2]	148
11.5.3.85	khole()	[2/2]	148
11.5.3.86	kinst()	[1/2]	149
11.5.3.87	kinst()	[2/2]	149
11.5.3.88	knetwk()	[1/2]	149
11.5.3.89	knetwk()	[2/2]	149
11.5.3.90	ko()	[1/2]	150
11.5.3.91	ko()	[2/2]	150
11.5.3.92	kstnm()	[1/2]	150
11.5.3.93	kstnm()	[2/2]	150
11.5.3.94	kt0()	[1/2]	151
11.5.3.95	kt0()	[2/2]	151

11.5.3.96 kt1() [1/2]	151
11.5.3.97 kt1() [2/2]	151
11.5.3.98 kt2() [1/2]	152
11.5.3.99 kt2() [2/2]	152
11.5.3.100 kt3() [1/2]	152
11.5.3.101 kt3() [2/2]	152
11.5.3.102 kt4() [1/2]	153
11.5.3.103 kt4() [2/2]	153
11.5.3.104 kt5() [1/2]	153
11.5.3.105 kt5() [2/2]	153
11.5.3.106 kt6() [1/2]	154
11.5.3.107 kt6() [2/2]	154
11.5.3.108 kt7() [1/2]	154
11.5.3.109 kt7() [2/2]	154
11.5.3.110 kt8() [1/2]	155
11.5.3.111 kt8() [2/2]	155
11.5.3.112 kt9() [1/2]	155
11.5.3.113 kt9() [2/2]	155
11.5.3.114 kuser0() [1/2]	156
11.5.3.115 kuser0() [2/2]	156
11.5.3.116 kuser1() [1/2]	156
11.5.3.117 kuser1() [2/2]	156
11.5.3.118 kuser2() [1/2]	157
11.5.3.119 kuser2() [2/2]	157
11.5.3.120 lcalda() [1/2]	157
11.5.3.121 lcalda() [2/2]	157
11.5.3.122 legacy_write()	157
11.5.3.123 leven() [1/2]	158
11.5.3.124 leven() [2/2]	159
11.5.3.125 lovrok() [1/2]	159
11.5.3.126 lovrok() [2/2]	159
11.5.3.127 lp spol() [1/2]	160
11.5.3.128 lp spol() [2/2]	160
11.5.3.129 mag() [1/2]	160
11.5.3.130 mag() [2/2]	160
11.5.3.131 nevid() [1/2]	161
11.5.3.132 nevid() [2/2]	161
11.5.3.133 norid() [1/2]	161
11.5.3.134 norid() [2/2]	161
11.5.3.135 npts() [1/2]	162
11.5.3.136 npts() [2/2]	162
11.5.3.137 nsnpts() [1/2]	162

11.5.3.138 nsnpts() [2/2]	162
11.5.3.139 nvhdr() [1/2]	163
11.5.3.140 nvhdr() [2/2]	163
11.5.3.141 nwfid() [1/2]	163
11.5.3.142 nwfid() [2/2]	163
11.5.3.143 nxsize() [1/2]	164
11.5.3.144 nxsize() [2/2]	164
11.5.3.145 nysize() [1/2]	164
11.5.3.146 nysize() [2/2]	164
11.5.3.147 nzhour() [1/2]	165
11.5.3.148 nzhour() [2/2]	165
11.5.3.149 nzjday() [1/2]	165
11.5.3.150 nzjday() [2/2]	165
11.5.3.151 nzmin() [1/2]	166
11.5.3.152 nzmin() [2/2]	166
11.5.3.153 nzmsec() [1/2]	166
11.5.3.154 nzmsec() [2/2]	166
11.5.3.155 nzsec() [1/2]	167
11.5.3.156 nzsec() [2/2]	167
11.5.3.157 nzyear() [1/2]	167
11.5.3.158 nzyear() [2/2]	167
11.5.3.159 o() [1/2]	168
11.5.3.160 o() [2/2]	168
11.5.3.161 odelta() [1/2]	168
11.5.3.162 odelta() [2/2]	168
11.5.3.163 operator==()	168
11.5.3.164 read_bool_headers()	169
11.5.3.165 read_datas()	170
11.5.3.166 read_float_headers()	172
11.5.3.167 read_float_headers_geometry()	174
11.5.3.168 read_float_headers_meta()	175
11.5.3.169 read_float_headers_resp()	177
11.5.3.170 read_float_headers_starter()	179
11.5.3.171 read_float_headers_station_event()	180
11.5.3.172 read_float_headers_t()	182
11.5.3.173 read_float_headers_user()	185
11.5.3.174 read_footers()	187
11.5.3.175 read_int_headers()	189
11.5.3.176 read_int_headers_datetime()	191
11.5.3.177 read_int_headers_meta()	192
11.5.3.178 read_string_headers()	195
11.5.3.179 resize_data()	197

11.5.3.180	resize_data1()	197
11.5.3.181	resize_data2()	197
11.5.3.182	resp0() [1/2]	198
11.5.3.183	resp0() [2/2]	198
11.5.3.184	resp1() [1/2]	198
11.5.3.185	resp1() [2/2]	198
11.5.3.186	resp2() [1/2]	199
11.5.3.187	resp2() [2/2]	199
11.5.3.188	resp3() [1/2]	199
11.5.3.189	resp3() [2/2]	199
11.5.3.190	resp4() [1/2]	200
11.5.3.191	resp4() [2/2]	200
11.5.3.192	resp5() [1/2]	200
11.5.3.193	resp5() [2/2]	200
11.5.3.194	resp6() [1/2]	201
11.5.3.195	resp6() [2/2]	201
11.5.3.196	resp7() [1/2]	201
11.5.3.197	resp7() [2/2]	201
11.5.3.198	resp8() [1/2]	202
11.5.3.199	resp8() [2/2]	202
11.5.3.200	resp9() [1/2]	202
11.5.3.201	resp9() [2/2]	202
11.5.3.202	sb() [1/2]	203
11.5.3.203	sb() [2/2]	203
11.5.3.204	sdelta() [1/2]	203
11.5.3.205	sdelta() [2/2]	203
11.5.3.206	station_location()	204
11.5.3.207	stdp() [1/2]	204
11.5.3.208	stdp() [2/2]	205
11.5.3.209	stel() [1/2]	205
11.5.3.210	stel() [2/2]	205
11.5.3.211	stla() [1/2]	205
11.5.3.212	stla() [2/2]	206
11.5.3.213	stlo() [1/2]	206
11.5.3.214	stlo() [2/2]	206
11.5.3.215	t0() [1/2]	207
11.5.3.216	t0() [2/2]	207
11.5.3.217	t1() [1/2]	207
11.5.3.218	t1() [2/2]	208
11.5.3.219	t2() [1/2]	208
11.5.3.220	t2() [2/2]	208
11.5.3.221	t3() [1/2]	208

11.5.3.222 t3() [2/2]	209
11.5.3.223 t4() [1/2]	209
11.5.3.224 t4() [2/2]	209
11.5.3.225 t5() [1/2]	209
11.5.3.226 t5() [2/2]	210
11.5.3.227 t6() [1/2]	210
11.5.3.228 t6() [2/2]	210
11.5.3.229 t7() [1/2]	210
11.5.3.230 t7() [2/2]	211
11.5.3.231 t8() [1/2]	211
11.5.3.232 t8() [2/2]	211
11.5.3.233 t9() [1/2]	211
11.5.3.234 t9() [2/2]	212
11.5.3.235 time()	212
11.5.3.236 user0() [1/2]	213
11.5.3.237 user0() [2/2]	213
11.5.3.238 user1() [1/2]	213
11.5.3.239 user1() [2/2]	213
11.5.3.240 user2() [1/2]	214
11.5.3.241 user2() [2/2]	214
11.5.3.242 user3() [1/2]	214
11.5.3.243 user3() [2/2]	214
11.5.3.244 user4() [1/2]	215
11.5.3.245 user4() [2/2]	215
11.5.3.246 user5() [1/2]	215
11.5.3.247 user5() [2/2]	215
11.5.3.248 user6() [1/2]	216
11.5.3.249 user6() [2/2]	216
11.5.3.250 user7() [1/2]	216
11.5.3.251 user7() [2/2]	216
11.5.3.252 user8() [1/2]	217
11.5.3.253 user8() [2/2]	217
11.5.3.254 user9() [1/2]	217
11.5.3.255 user9() [2/2]	217
11.5.3.256 write()	217
11.5.3.257 write_bool_headers()	219
11.5.3.258 write_data()	221
11.5.3.259 write_float_headers()	221
11.5.3.260 write_float_headers_geometry()	223
11.5.3.261 write_float_headers_meta()	224
11.5.3.262 write_float_headers_resp()	226
11.5.3.263 write_float_headers_starter()	228

11.5.3.264	<code>write_float_headers_station_event()</code>	229
11.5.3.265	<code>write_float_headers_t()</code>	231
11.5.3.266	<code>write_float_headers_user()</code>	233
11.5.3.267	<code>write_footers()</code>	235
11.5.3.268	<code>write_int_headers()</code>	237
11.5.3.269	<code>write_int_headers_datetime()</code>	239
11.5.3.270	<code>write_int_headers_meta()</code>	240
11.5.3.271	<code>write_string_headers()</code>	243
11.5.3.272	<code>xmaximum()</code> [1/2]	246
11.5.3.273	<code>xmaximum()</code> [2/2]	246
11.5.3.274	<code>xminimum()</code> [1/2]	246
11.5.3.275	<code>xminimum()</code> [2/2]	247
11.5.3.276	<code>ymaximum()</code> [1/2]	247
11.5.3.277	<code>ymaximum()</code> [2/2]	247
11.5.3.278	<code>yminimum()</code> [1/2]	247
11.5.3.279	<code>yminimum()</code> [2/2]	247
11.5.4	Member Data Documentation	248
11.5.4.1	<code>bools</code>	248
11.5.4.2	<code>data</code>	248
11.5.4.3	<code>doubles</code>	248
11.5.4.4	<code>floats</code>	248
11.5.4.5	<code>ints</code>	248
11.5.4.6	<code>strings</code>	248
11.6	<code>sacfmt::bitset_type::uint< nbits ></code> Struct Template Reference	249
11.6.1	Detailed Description	249
11.7	<code>sacfmt::bitset_type::uint< 4 *bits_per_byte ></code> Struct Reference	249
11.7.1	Detailed Description	249
11.7.2	Member Typedef Documentation	249
11.7.2.1	<code>type</code>	249
11.8	<code>sacfmt::bitset_type::uint< bytes *bits_per_byte ></code> Struct Reference	250
11.8.1	Detailed Description	250
11.8.2	Member Typedef Documentation	250
11.8.2.1	<code>type</code>	250
11.9	<code>sacfmt::word_pair< T ></code> Struct Template Reference	250
11.9.1	Detailed Description	250
11.9.2	Member Data Documentation	251
11.9.2.1	<code>first</code>	251
11.9.2.2	<code>second</code>	251

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:

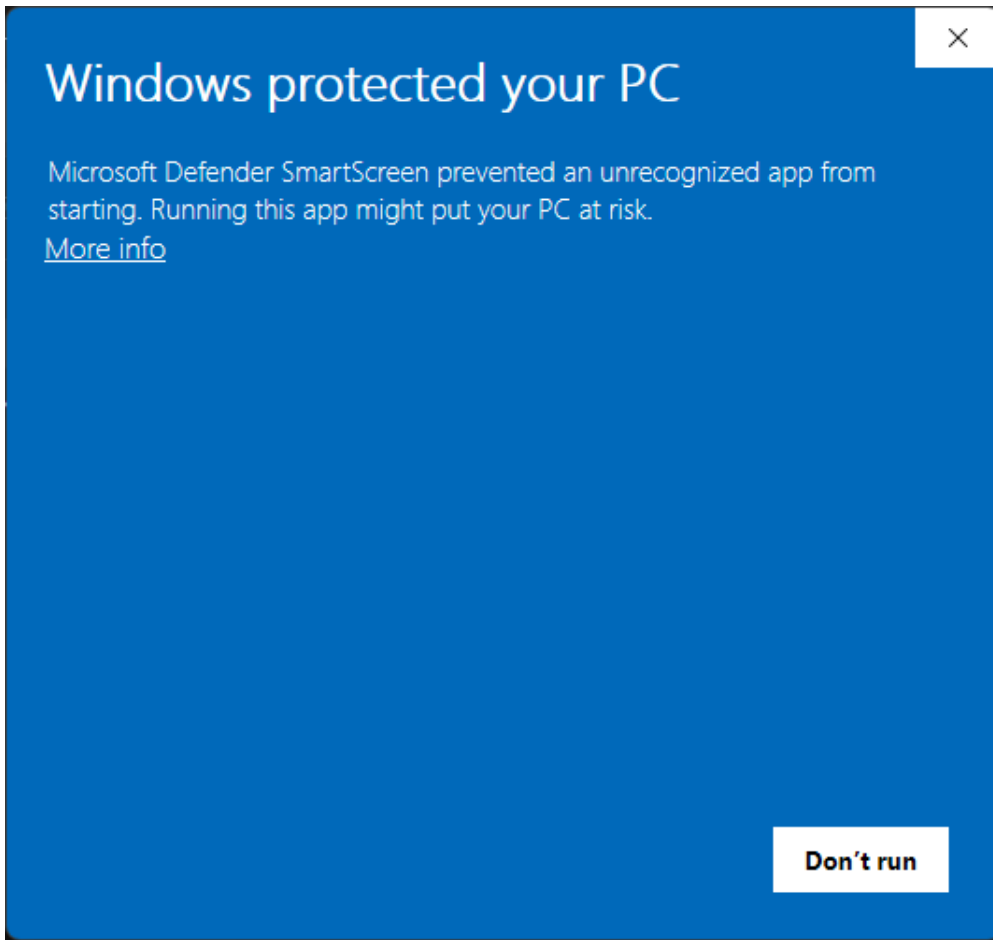


Figure 2.1 Windows Warning 1

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

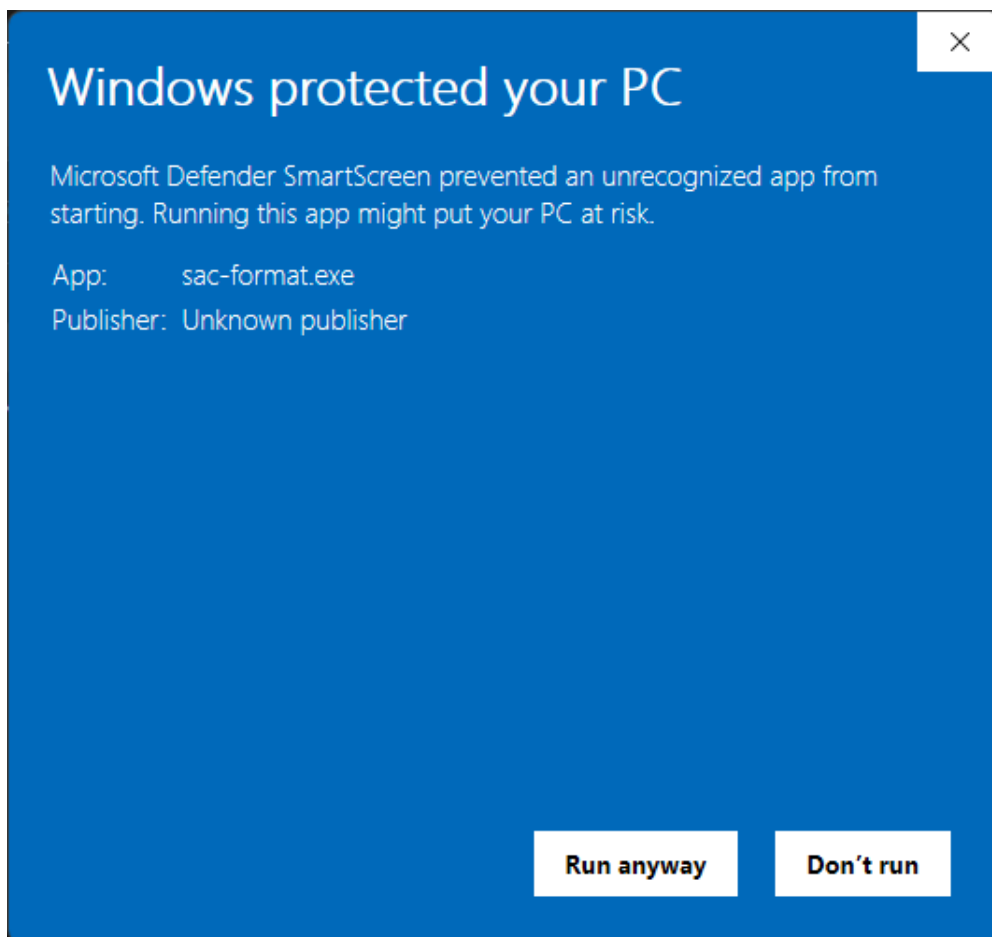


Figure 2.2 Windows Warning 2

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

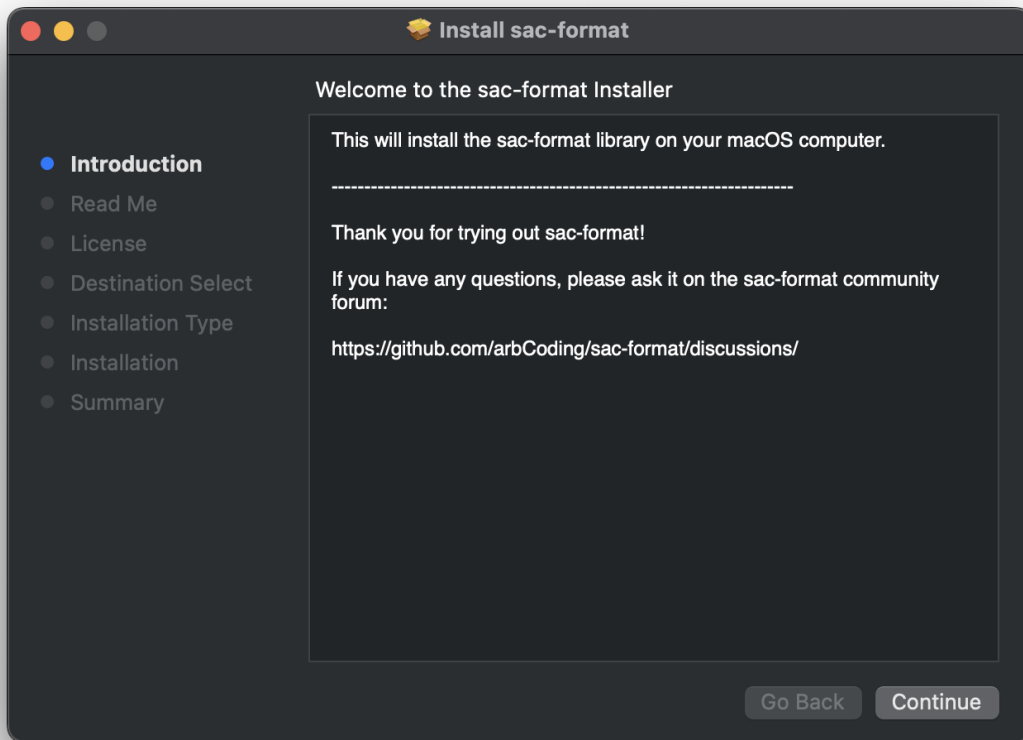


Figure 2.3 Windows Intro Install

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

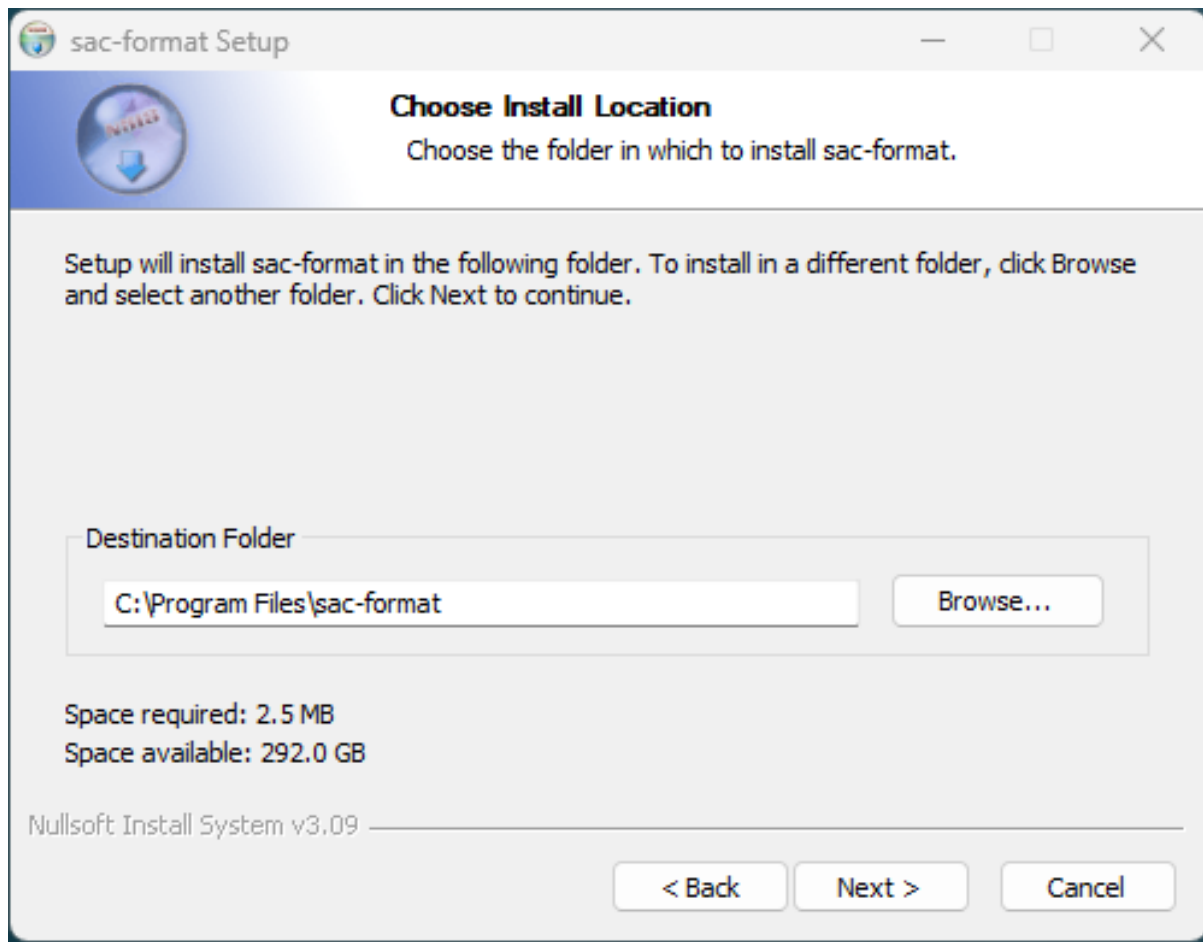


Figure 2.4 Windows Location Install

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

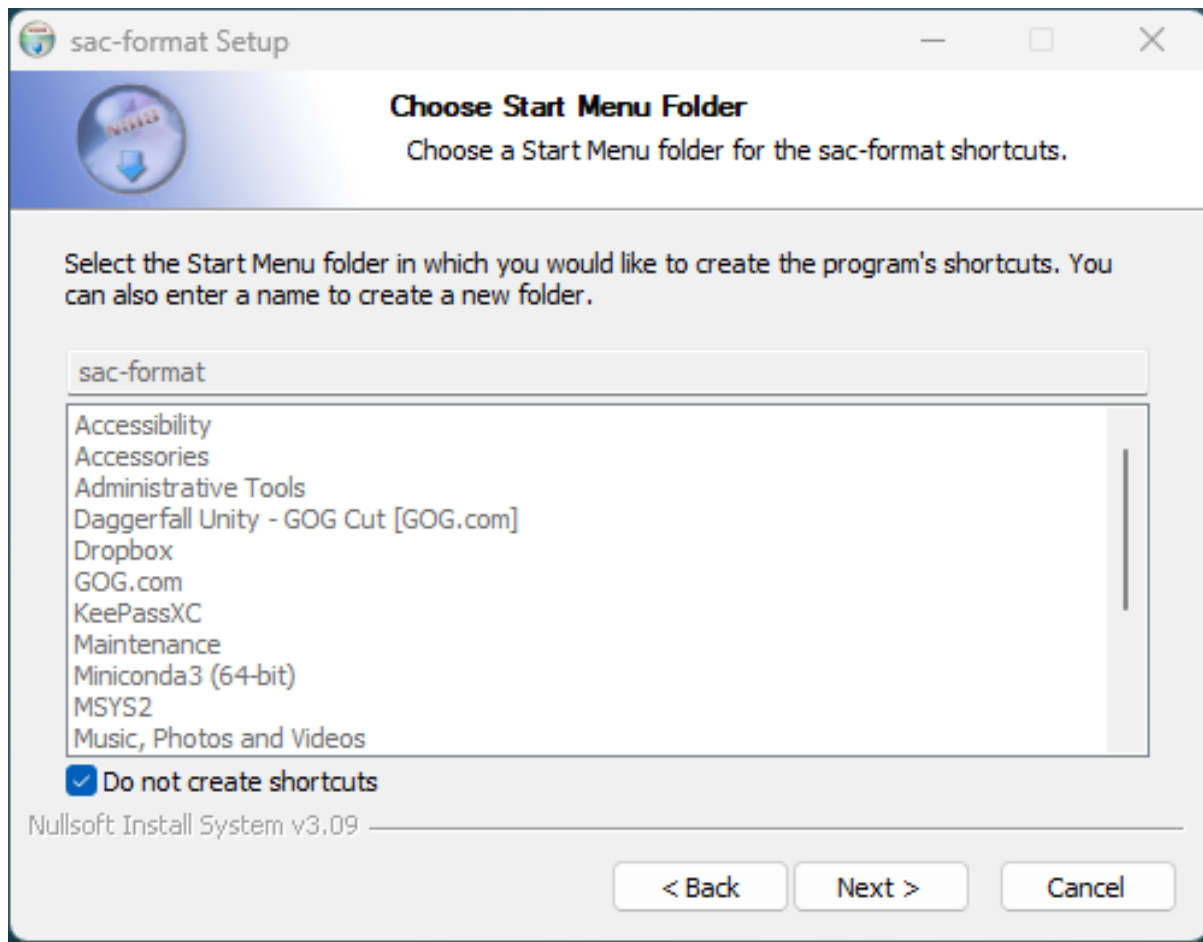


Figure 2.5 Windows No Shortcuts

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

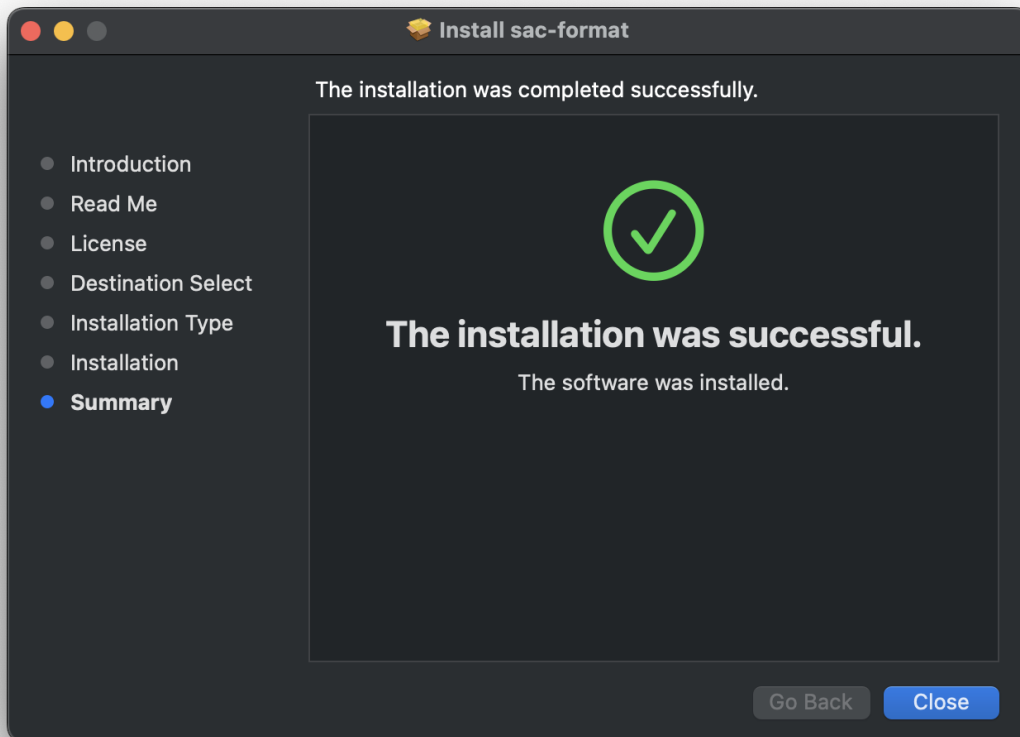


Figure 2.6 Windows Install Success

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:

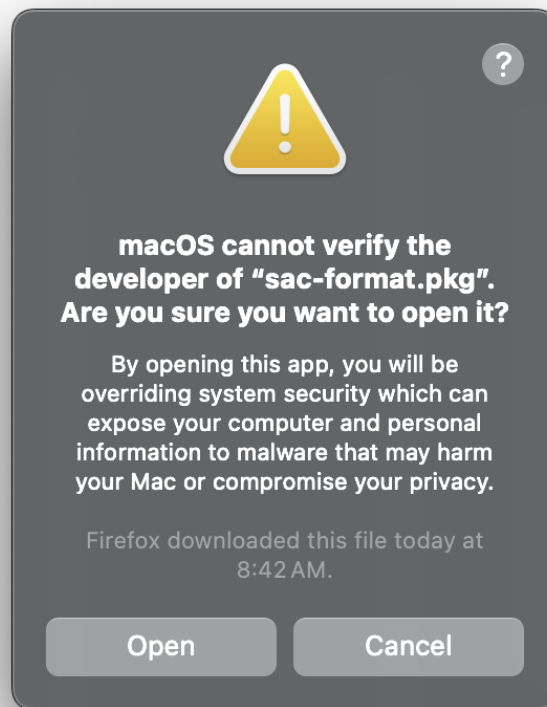


Figure 2.7 macOS Warning

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

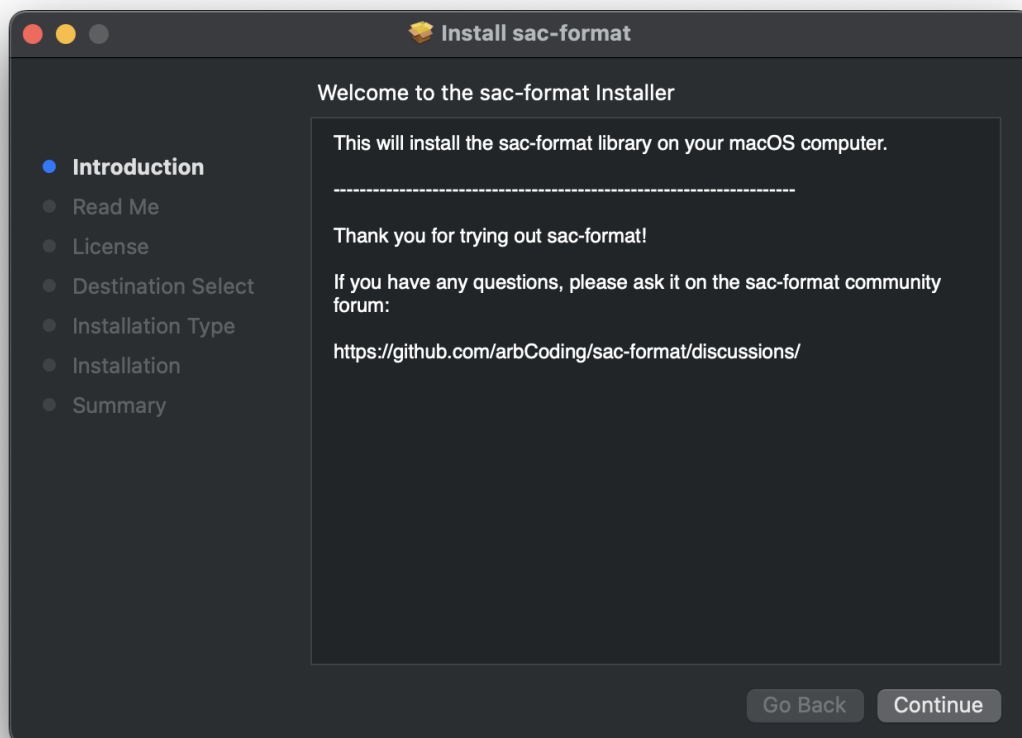


Figure 2.8 macOS Intro Install

Upon successful installation you will see:

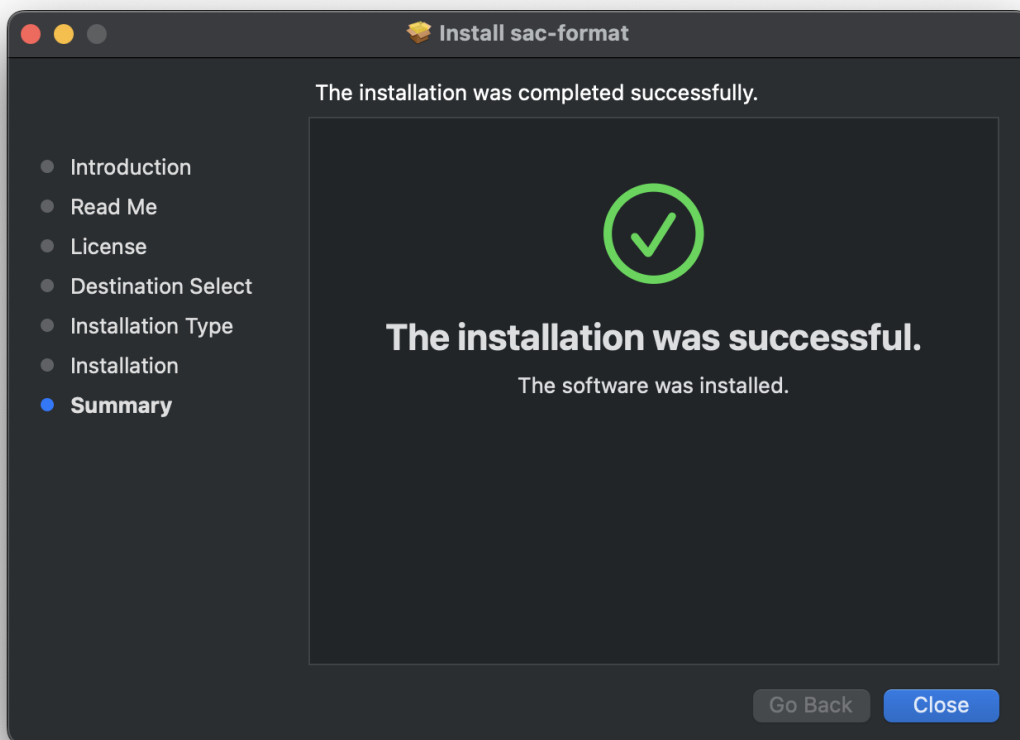


Figure 2.9 macOS Install Success

2.2.2 Command line

Command line installation is performed either using the self-extracting 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

```
#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` potentially 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).

```
const point location1(coord{latitude1}, coord{longitude1});
const point location2(coord{latitude2}, coord{longitude2});
double gcarc{sacfmt::gcarc(location1, location2)};
```

4.2.4 azimuth

Calculate azimuth between two points (spherical planet).

```
const point location1(coord{latitude1}, coord{longitude1});
const point location2(coord{latitude2}, coord{longitude2});
double azimuth{sacfmt::azimuth(location2, location1)};
double back_azimuth{sacfmt::azimuth(location1, location2)};
```

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.h` ↔ `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](https://codecov.io) and [Codacy.com](https://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 `benchmark -?`.

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

Pre-data word 001.

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

5.1.2 depmen

Pre-data word 057.

Mean value of the dependent variable.

5.1.3 depmax

Pre-data word 002.

Maximum value of the dependent variable.

5.1.4 odelta

Pre-data word 004.

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

5.1.5 resp(0–9)

Pre-data words 021–030.

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

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

5.1.6 stel

Pre-data word 033.

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

Not used by SAC free for other purposes.

5.1.7 stdp

Pre-data word 034.

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

Not used by SAC free for other purposes.

5.1.8 evel

Pre-data word 037.

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

Not used by SAC free for other purposes.

5.1.9 evdp

Pre-data word 038.

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

5.1.10 mag

Pre-data word 039.

Event magnitude.

5.1.11 user(0–9)

Pre-data words 040–049.

Storage for user-defined values.

5.1.12 dist

Pre-data word 050.

Station-Event distance in kilometers.

5.1.13 az

Pre-data word 051.

Azimuth ([Event](#) → [Station](#)), decimal degrees from North.

5.1.14 baz

Pre-data word 052.

Back-azimuth ([Station](#) → [Event](#)), decimal degrees from North.

5.1.15 gcarc

Pre-data word 053.

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

5.1.16 cmpaz

Pre-data word 057.

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

Pre-data word 058.

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

Pre-data word 059.

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

5.1.19 xmaximum

Pre-data word 060.

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

5.1.20 yminimum

Pre-data word 061.

Spectral-only equivalent of `b`.

5.1.21 ymaximum

Pre-data word 062.

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 $\nu 7$ SAC-files. In memory they're stored as doubles regardless of the SAC-file version.

5.2.1 delta

Pre-data word 000, post-data words 00-01.

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

5.2.2 b

Pre-data word 005, post-data words 02-03.

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

5.2.3 e

Pre-data word 006, post-data words 04-05.

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

5.2.4 o

Pre-data word 007, post-data words 06-07.

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

5.2.5 a

Pre-data word 008, post-data words 08-09.

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

5.2.6 t(0–9)

Pre-data words 010–019, post-data words 10–29.

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

5.2.7 f

Pre-data word 020, post-data words 30-31.

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

5.2.8 stla

Pre-data word 031, post-data words 36-37.

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

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

5.2.9 stlo

Pre-data word 032, post-data words 38-39.

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

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

5.2.10 evla

Pre-data word 035, post-data words 32-33.

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

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

5.2.11 evlo

Pre-data word 036, post-data words 34-35.

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

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

5.2.12 sb

Pre-data word 054, post-data words 40-41.

Original (*saved*) *b* value.

5.2.13 sdelta

Pre-data word 055, post-data words 42-43.

Original (*saved*) *delta* value.

5.3 Integer (26)

32-bit (1 word, 4 bytes)

5.3.1 nzyear

Pre-data word 070.

Reference time GMT year.

5.3.2 nzjday

Pre-data word 071.

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

5.3.3 nzhour

Pre-data word 072.

Reference time GMT hour (0–23).

5.3.4 nzmin

Pre-data word 073.

Reference time GMT minute (0–59).

5.3.5 nzsec

Pre-data word 074.

Reference time GMT second (0–59).

5.3.6 nzmsec

Pre-data word 075.

Reference time GMT Millisecond (0–999).

5.3.7 nvhdr

Pre-data word 076.

SAC-file version.

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

5.3.8 norid

Pre-data word 077.

Origin ID.

5.3.9 nevid

Pre-data word 078.

Event ID.

5.3.10 npts

Pre-data word 079.

Number of points in data.

5.3.11 nsnpts

Pre-data word 080.

Original (*saved*) `npts`.

5.3.12 nwfid

Pre-data word 081.

Waveform ID.

5.3.13 nxsize

Pre-data word 082.

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

5.3.14 nysize

Pre-data word 083.

Spectral-only, width of spectrum.

5.3.15 iftype

Pre-data word 085.

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

Pre-data word 086.

Dependent variable type.

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

5.3.17 iztype

Pre-data word 087.

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

Pre-data word 089.

Recording instrument type.

Not used by SAC: free for other purposes.

5.3.19 istreg

Pre-data word 090.

Station geographic region.

Not used by SAC: free for other purposes.

5.3.20 ievreg

Pre-data word 091.

Event geographic region.

Not used by SAC: free for other purposes.

5.3.21 ievtyp

Pre-data word 092.

Event type.

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

5.3.22 ional

Pre-data word 093.

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

Pre-data word 094.

Synthetic data flag.

Value	Type	Description
49	IRLDATA	Real data
XX	*	Synthetic

*Values and types not listed in the standard.

5.3.24 imagtyp

Pre-data word 095.

Magnitude type.

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

5.3.25 imagsrc

Pre-data word 096.

Source of magnitude information.

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

5.3.26 ibody

Pre-data word 097.

Body/spheroid definition used to calculate distances.

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

5.4 Boolean (4)

Pre-data word 105.

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

5.4.1 leven

Pre-data word 106.

REQUIRED Evenly-spaced data flag.

If true, then data is evenly spaced.

5.4.2 lpspol

Pre-data word 107.

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

Pre-data word 108.

File overwrite flag.

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

5.4.4 lcalda

Pre-data word 109.

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

Pre-data words 110–111.

Station name.

5.5.2 kevnm

Pre-data words 112–115.

Event name.

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

5.5.3 khole

Pre-data words 116–117.

Nuclear: Hole identifier.

Other: Location identifier (LOCID).

5.5.4 ko

Pre-data words 118–119.

Text for ○.

5.5.5 ka

Pre-data words 120–121.

Text for `a`.

5.5.6 kt(0–9)

Pre-data words 112–141.

Text for `t` (0–9).

5.5.7 kf

Pre-data words 142–143.

Text for `f`.

5.5.8 kuser(0–2)

Pre-data words 144–149.

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

5.5.9 kcmpnm

Pre-data words 150–151.

Component name.

5.5.10 knetwk

Pre-data words 152–153.

Network name.

5.5.11 kdatrd

Pre-data words 154–155.

Date the data was read onto a computer.

5.5.12 `kinst`

Pre-data words 156–157.

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`

Words 158–(158 + `npts`)

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

5.6.2 `data2`

Words (158 + 1 + `npts`)–(159 + (2 * `npts`))

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	51
sacfmt::bitset_type	Bitset type-safety namespace	103

Chapter 8

Hierarchical Index

8.1 Class Hierarchy

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

sacfmt::coord	105
std::exception	
sacfmt::io_error	108
sacfmt::point	110
sacfmt::read_spec	112
sacfmt::Trace	112
sacfmt::bitset_type::uint< nbits >	249
sacfmt::bitset_type::uint< 4 *bits_per_byte >	249
sacfmt::bitset_type::uint< bytes *bits_per_byte >	250
sacfmt::word_pair< T >	250

Chapter 9

Class Index

9.1 Class List

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

sacfmt::coord	Defines a geographic coordinant (degrees/radians)	105
sacfmt::io_error	Class for generic I/O exceptions	108
sacfmt::point	Defines a geographic point (latitude, longitude)	110
sacfmt::read_spec	Struct that specifies parameters for reading	112
sacfmt::Trace	The Trace class	112
sacfmt::bitset_type::uint< nbits >	Ensure type-safety for conversions between floats/doubles and bitsets	249
sacfmt::bitset_type::uint< 4 *bits_per_byte >	One-word (floats)	249
sacfmt::bitset_type::uint< bytes *bits_per_byte >	Two-words (doubles)	250
sacfmt::word_pair< T >	Struct containing a pair of words	250

Chapter 10

Namespace Documentation

10.1 sacfmt Namespace Reference

sac-format namespace

Namespaces

- namespace [bitset_type](#)
bitset type-safety namespace.

Classes

- class [coord](#)
Defines a geographic coordinant (degrees/radians)
- class [io_error](#)
Class for generic I/O exceptions.
- struct [point](#)
Defines a geographic point (latitude, longitude)
- 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`, `igual`, `isynth`,
 - `imagtyp`, `imagsrc`, `ibody`, `leven`,
 - `lpspol`, `lovrok`, `lcalda`, `kstnm`,
 - `kevm`, `khole`, `ko`, `ka`,
 - `kt0`, `kt1`, `kt2`, `kt3`,
 - `kt4`, `kt5`, `kt6`, `kt7`,
 - `kt8`, `kt9`, `kf`, `kuser0`,
 - `kuser1`, `kuser2`, `kcompnm`, `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, const size_t 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 point location1, const point location2) noexcept`
Calculate great circle arc distance in decimal degrees between two points.
- `double azimuth (const point location1, const point location2) 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 size_t 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 fundamental 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-precision 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, pre-data word 001. Minimum value of the dependent variable (displacement/velocity/acceleration/volts/counts).
depmax	Float, pre-data word 002. Maximum value of the dependent variable.
odelta	Float, pre-data word 004. Modified (observational) value of delta.
resp0	Float, pre-data word 021. Instrument response parameter (poles, zeros, and a constant). Not used by SAC - free for other purposes.
resp1	See resp0, pre-data word 022.
resp2	See resp0, pre-data word 023.

Enumerator

resp3	See resp0, pre-data word 024.
resp4	See resp0, pre-data word 025.
resp5	See resp0, pre-data word 026.
resp6	See resp0, pre-data word 027.
resp7	See resp0, pre-data word 028.
resp8	See resp0, pre-data word 029.
resp9	See resp0, pre-data word 030.
stel	Float, pre-data word 033. Station elevation in meters above sea level (m.a.s.l.). Not used by SAC - free for other purposes.
stdp	Float, pre-data word 034. Station depth in meters below surface (borehole/buried vault). Not used by SAC - free for other purposes.
evel	Float, pre-data word 037. Event elevation m.a.s.l. Not used by SAC - free for other purposes.
evdp	Float, pre-data word 038. Event depth in kilometers (previous meters) below surface.
mag	Float, pre-data word 039. Event magnitude.
user0	Float, pre-data word 040. Storage for user-defined values.
user1	See user0, pre-data word 041.
user2	See user0, pre-data word 042.
user3	See user0, pre-data word 043.
user4	See user0, pre-data word 044.
user5	See user0, pre-data word 045.
user6	See user0, pre-data word 046.
user7	See user0, pre-data word 047.
user8	See user0, pre-data word 048.
user9	See user0, pre-data word 049.
dist	Float, pre-data word 050. Station-Event distance in kilometers.
az	Float, pre-data word 051. Azimuth <i>Station</i> → <i>Event</i> in decimal degrees from North.
baz	Float, pre-data word 052. Back-Azimuth <i>Event</i> → <i>Station</i> in decimal degrees from North.
gcarc	Float, pre-data word 053. Great-circle arc-distance between station and event in decimal degrees.
depmen	Float, pre-data word 056. Mean value of dependent variable.
cmpaz	Float, pre-data word 057. Instrument measurement azimuth, decimal degrees from North.
cmpinc	Float, pre-data word 058. 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, pre-data word 059. Spectral-only equivalent of depmin (f_0 or ω_0).

Enumerator

xmaximum	Float, pre-data word 060. Spectral-only equivalent of depman (f_{max} or ω_{max}).
yminimum	Float, pre-data word 061. Spectral-only equivalent of b.
ymaximum	Float, pre-data word 062. Spectral-only equivalent of e.
delta	Double, pre-data word 000; post-data words 00-01. Increment between evenly-spaced samples (Δt for timeseries, Δf or $\Delta \omega$ for spectral).
b	Double, pre-data word 005; post-data words 02-03. First value (beginning) of independent variable (t_0).
e	Double, pre-data word 006; post-data words 04-05. Final value (ending) of the independent variable (t_{max}).
o	Double, pre-data word 007; post-data words 06-07. Event origin time, in seconds relative to the reference time.
a	Double, pre-data word 008; post-data words 08-09. Event first arrival time, in seconds relative to the reference time.
t0	Double, pre-data word 010; post-data words 10-11. User defined time value, in seconds relative to the reference time.
t1	See t0, pre-data word 011; post-data words 12-13.
t2	See t0, pre-data word 012; post-data words 14-15.
t3	See t0, pre-data word 013; post-data words 16-17.
t4	See t0, pre-data word 014; post-data words 18-19.
t5	See t0, pre-data word 015; post-data words 20-21.
t6	See t0, pre-data word 016; post-data words 22-23.
t7	See t0, pre-data word 017; post-data words 24-25.
t8	See t0, pre-data word 018; post-data words 26-27.
t9	See t0, pre-data word 019; post-data words 28-29.
f	Double, pre-data word 020; post-data words 30-31. Event end (fini) time, in seconds relative to the reference time.
stla	Double, pre-data word 031; post-data words 36-37. Station latitude in decimal degrees, N/S is positive/negative. sac-format automatically enforces $\phi \in [-90, 90]$.
stlo	Double, pre-data word 032; post-data words 38-39. Station longitude in decimal degrees, E/W is positive/negative. sac-format automatically enforces $\lambda \in [-180, 180]$.
evla	Double, pre-data word 035; post-data words 32-33. Event latitude in decimal degrees, N/S is positive/negative. sac-format automatically enforces $\phi \in [-90, 90]$.
evlo	Double, pre-data word 036; post-data words 34-35. Event longitude in decimal degrees, E/W is positive/negative. sac-format automatically enforces $\lambda \in [-180, 180]$.
sb	Double, pre-data word 054; post-data words 40-41. Original (saved) value of b (beginning).
sdelta	Double, pre-data word 055; post-data words 42-43. Original (saved) value of delta (sample-spacing).
nzyear	Integer, pre-data word 070. Reference time GMT year.
nzjday	Integer, pre-data word 071. Reference time GMT day-of-year (often called Julian Date). 1-366 Not enforced.

Enumerator

nzhour	Integer, pre-data word 072. Reference time GMT hour. 00-23 Not enforced.
nzmin	Integer, pre-data word 073. Reference time GMT minute. 00-59 Not enforced.
nzsec	Integer, pre-data word 074. Reference time GMT second. 00-59 Not enforced.
nzmsec	Integer, pre-data word 075. Reference time GMT millisecond. 0-999 not enforced.
nvhdr	Integer, pre-data word 076. SAC-file version. 7 = 2020+, sac 102.0+, has a Footer. 6 = pre-2020, sac 101.6a-, no Footer.
norid	Integer, pre-data word 077. Origin ID.
nevid	Integer, pre-data word 078. Event ID.
npts	Integer, pre-data word 079. Number of points in data.
nsnpts	Integer, pre-data word 080. Original (saved) npts.
nwfid	Integer, pre-data word 081. Waveform ID.
nxsize	Integer, pre-data word 082. Spectral-only equivalent of npts (length of spectrum).
nysize	Integer, pre-data word 083. Spectral-only; width of spectrum.
iftype	Integer, pre-data word 085. File type.
idep	Integer, pre-data word 086. Dependent variable type.
iztype	Integer, pre-data word 087. Reference time equivalent.
iinst	Integer, pre-data word 089. Recording instrument type. Not used by SAC - free for other purposes.
istreg	Integer, pre-data word 090. Station geographic region. Not used by SAC - free for other purposes.
ievreg	Integer, pre-data word 091. Event geographic region. Not used by SAC - free for other purposes.
ievtyp	Integer, pre-data word 092. Event type. Not used by SAC - free for other purposes.
igual	Integer, pre-data word 093. Quality of data. Not used by SAC - free for other purposes.
isynth	Integer, pre-data word 094. Synthetic data flag. Not used by SAC - free for other purposes.

Enumerator

imagtyp	Integer, pre-data word 095. Magnitude type.
imagsrc	Integer, pre-data word 096. Magnitude information source.
ibody	Integer, pre-data word 097. Body/spheroid definition used to calculate distances. Not currently-used by sac-format (SAC does use it).
leven	Boolean, pre-data word 105. REQUIRED Evenly-spaced data flag. True = even.
lpspol	Boolean, pre-data word 106. Station polarity flag. True = positive (left-handed, e.g. North-East-Up).
lovrok	Boolean, pre-data word 107. File overwrite flag. If true, okay to overwrite file. Not used by sac-format.
lcalda	Boolean, pre-data word 108. Calculate geometry flag. Not used by sac-format.
kstnm	String (2 words), pre-data words 110–111. Station name.
kevnm	String (4 words), pre-data words 112–115. Event name.
khole	String (2 words), pre-data words 116–117. Nuclear-Hole identifier. Other-Location identifier (LOCID).
ko	String (2 words), pre-data words 118–119. Text for o.
ka	String (2 words), pre-data words 120–121. Text for a.
kt0	String (2 words), pre-data words 122–123. Text for t0
kt1	See kt0, pre-data words 124–125.
kt2	See kt0, pre-data words 126–127.
kt3	See kt0, pre-data words 128–129.
kt4	See kt0, pre-data words 130–131.
kt5	See kt0, pre-data words 132–133.
kt6	See kt0, pre-data words 134–135.
kt7	See kt0, pre-data words 136–137.
kt8	See kt0, pre-data words 138–139.
kt9	See kt0, pre-data words 140–141.
kf	String (2 words), pre-data words 142–143. Text for f.
kuser0	String (2 words), pre-data words 144–145. Text for user0.
kuser1	See kuser0, pre-data words 146–147.
kuser2	See kuser0, pre-data words 148–149.
kcmpnm	String (2 words), pre-data words 150–151. Component name.
knetwk	String (2 words), pre-data words 152–153. Network name.

Enumerator

kdatrd	String (2 words), pre-data words 154-155. Date the data was read onto a computer.
kinst	String (2 words), pre-data words 156-157. Instrument name.
data1	std::vector<double>, words 158–(158 + npts) First data vector. ALWAYS present, ALWAYS begins at word 158.
data2	std::vector<double>, words (158 + 1 + npts)–(159 + (2 * npts)) 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.

```

00316
00317 // Floats
00324 depmin,
00330 depmax,
00336 odelta,
00344 resp0,
00346 resp1,
00348 resp2,
00350 resp3,
00352 resp4,
00354 resp5,
00356 resp6,
00358 resp7,
00360 resp8,
00362 resp9,
00370 stel,
00378 stdp,
00386 evel,
00392 evdp,
00398 mag,
00404 user0,
00406 user1,
00408 user2,
00410 user3,
00412 user4,
00414 user5,
00416 user6,
00418 user7,
00420 user8,
00422 user9,
00428 dist,
00435 az,
00442 baz,
00448 gcarc,
00454 depmen,
00460 cmpaz,
00470 cmpinc,
00477 xminimum,
00484 xmaximum,
00490 yminimum,
00496 ymaximum,
00497 // Doubles
00506 delta,
00512 b,
00519 e,
00525 o,
00531 a,
00537 t0,
00539 t1,
00541 t2,
00543 t3,
00545 t4,
00547 t5,
00549 t6,
00551 t7,
00553 t8,
00555 t9,
00561 f,
00569 stla,
00577 stlo,
00585 evla,
00593 evlo,
00599 sb,
00605 sdelta,
00606 // Ints
00612 nzyear,
00620 nzjday,
00628 nzhour,
00636 nzmin,

```

```

00644  nzsec,
00652  nzmsec,
00661  nvhdr,
00667  norid,
00673  nevid,
00679  npts,
00685  nsnpts,
00691  nwfid,
00697  nxsize,
00703  nysize,
00709  iftype,
00715  idep,
00721  iztype,
00729  iinst,
00737  istreg,
00745  ievreg,
00753  ievtyp,
00761  igual,
00769  isynth,
00775  imagtyp,
00781  imagsrc,
00789  ibody,
00790  // Bools
00798  leven,
00806  lspol,
00816  lovrok,
00824  lcalda,
00825  // Strings
00831  kstnm,
00837  kevnm,
00845  khole,
00851  ko,
00857  ka,
00863  kt0,
00865  kt1,
00867  kt2,
00869  kt3,
00871  kt4,
00873  kt5,
00875  kt6,
00877  kt7,
00879  kt8,
00881  kt9,
00887  kf,
00893  kuser0,
00895  kuser1,
00897  kuser2,
00903  kcmpnm,
00909  knetwk,
00915  kdatrd,
00921  kinst,
00922  // Data
00928  data1,
00937  data2
00938 };

```

10.1.4 Function Documentation

10.1.4.1 azimuth()

```

double sacfmt::azimuth (
    const point location1,
    const point location2 ) [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	<i>location1</i>	point of first location.
in	<i>location2</i>	point of second location.

Returns

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

```

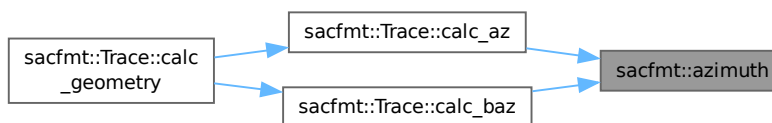
00766                                     {
00767     const double numerator{
00768         std::sin(location2.longitude.radians() - location1.longitude.radians()) *
00769         std::cos(location2.latitude.radians())};
00770     const double denominator{(std::cos(location1.latitude.radians()) *
00771         std::sin(location2.latitude.radians()) -
00772         (std::sin(location1.latitude.radians()) *
00773         std::cos(location2.latitude.radians()) *
00774         std::cos(location2.longitude.radians() -
00775         location1.longitude.radians()))};
00776     double result{radians_to_degrees(std::atan2(numerator, denominator))};
00777     while (result < 0.0) {
00778         result += circle_deg;
00779     }
00780     return result;
00781 }

```

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

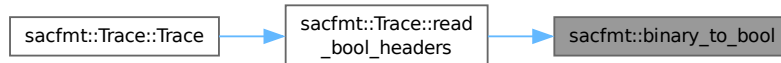
in	<i>flag</i>	<code>word_one</code> binary bitset to be converted (takes zeroth element).
----	-------------	---

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

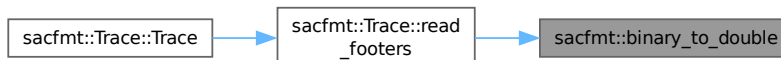
in	<i>bin</i>	<code>word_two</code> Binary value to be converted.
----	------------	---

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

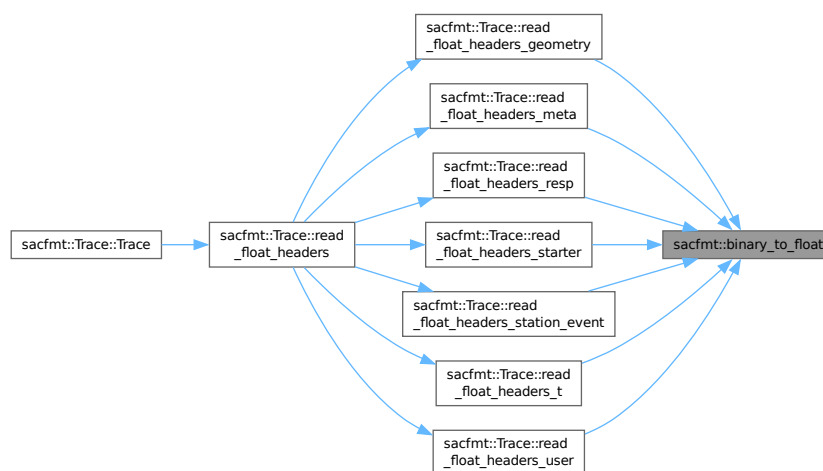
in	<i>bin</i>	<code>word_one</code> Binary value to be converted.
----	------------	---

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	<i>bin</i>	Binary value to be converted.
----	------------	-------------------------------

Returns

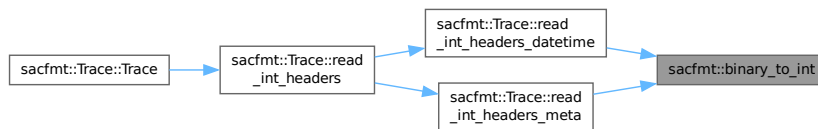
int Converted value.

```

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	<i>str</i>	<i>word_four</i> to be converted to a string.
----	------------	---

Returns

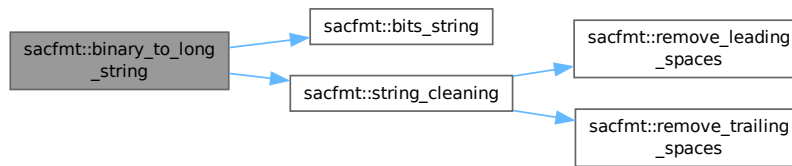
std::string Converted string.

```

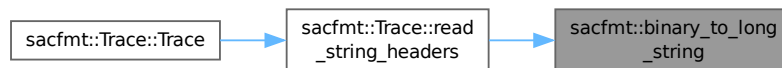
00332                                     {
00333     std::string result{bits_string(str, 4)};
00334     return string_cleaning(result);
00335 }

```

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

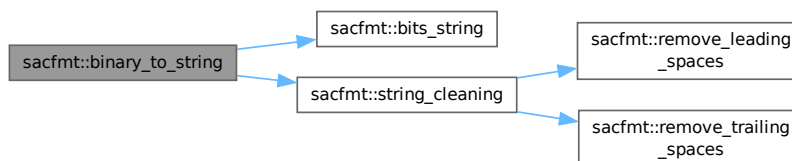
in	<i>str</i>	<code>word_two</code> to be converted to a string.
----	------------	--

Returns

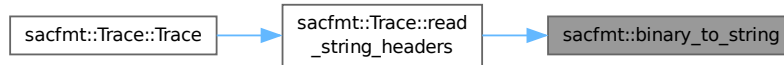
std::string Converted string.

```
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	<i>bits</i>	Source bitset for the string.
in	<i>num_words</i>	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	<i>flag</i>	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

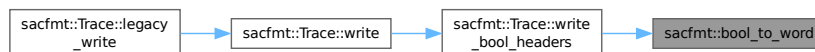
in	<i>flag</i>	Boolean to be converted.
----	-------------	--------------------------

Returns

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

```
00598                                     {
00599   std::vector<char> result;
00600   result.resize(word_length);
00601   std::fill(result.begin() + 1, result.end(), 0);
00602   result[0] = static_cast<char>(flag ? 1 : 0);
00603   return result;
00604 }
```

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	<i>pair_words</i>	word_pair Words to be concatenated.
----	-------------------	---

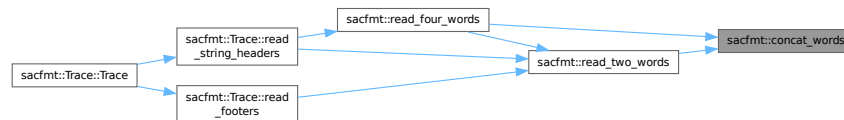
Returns

[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	<i>pair_words</i>	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	<i>input</i>	Input value to convert (double).
----	--------------	----------------------------------

Returns

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

```

00549                                     {
00550     constexpr size_t n_words{static_cast<size_t>(2) * word_length};
00551     std::array<char, n_words> tmp{};
00552     // Copy bytes from input into the tmp array
00553     // flawfinder: ignore
00554     std::memcpy(tmp.data(), &input, n_words);
00555     std::vector<char> word{};
00556     word.reserve(n_words);
00557     std::for_each(tmp.begin(), tmp.end(),
00558                 [&word](const char &character) { word.push_back(character); });
00559     return word;
00560 }
```

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	<i>input</i>	Input value (float or int) to convert.
----	--------------	--

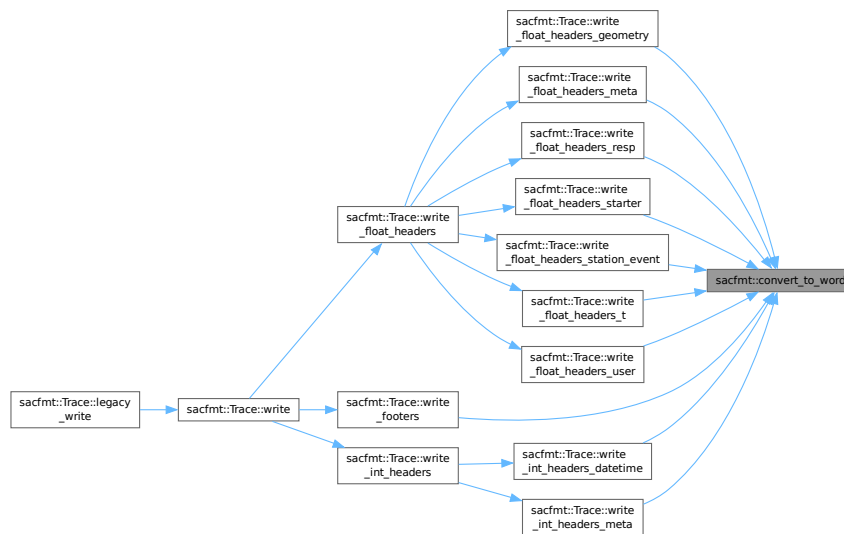
Returns

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.reserve(word_length);
00534     std::for_each(tmp.begin(), tmp.end(),
00535                 [&word](const char &character) { word.push_back(character); });
00536     return word;
00537 }
```

Here is the caller graph for this function:



10.1.4.17 convert_to_words() [1/2]

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

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

Parameters

in	<i>str</i>	Input string to convert.
in	<i>n_words</i>	Number of words

Returns

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

```
00573                                     {
00574     std::vector<char> tmp{};
00575     tmp.reserve(n_words);
00576     std::for_each(str.begin(), str.end(),
00577                 [&tmp](const char &character) { tmp.push_back(character); });
00578     std::array<char, N> all_words{};
00579     // Move vector to array
00580     std::move(tmp.begin(), tmp.end(), all_words.begin());
00581     return all_words;
00582 }
```

10.1.4.18 convert_to_words() [2/2]

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

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

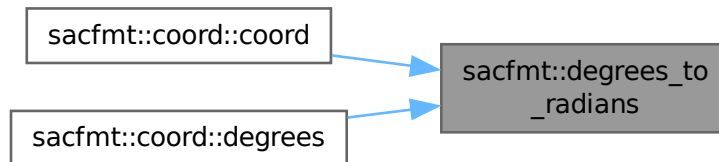
in	<i>degrees</i>	Angle in decimal degrees to be converted.
----	----------------	---

Returns

double Angle in radians.

```
00659                                     {
00660     return rad_per_deg * degrees;
00661 }
```

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	<i>num</i>	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	<i>val1</i>	First double in comparison.
in	<i>val2</i>	Second double in comparison.
in	<i>tolerance</i>	Numerical equality tolerance (default <code>f_eps</code>).

Returns

`bool` Boolean equality value.

```

00645                                     {
00646     return std::abs(val1 - val2) < tolerance;
00647 }
```

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	<i>vector1</i>	First data vector in comparison.
in	<i>vector2</i>	Second data vector in comparison.
in	<i>tolerance</i>	Numerical equality tolerance (default <code>f_eps</code>).

Returns

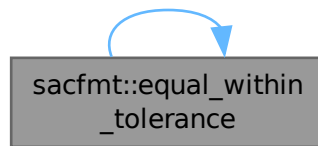
bool Boolean equality value.

```

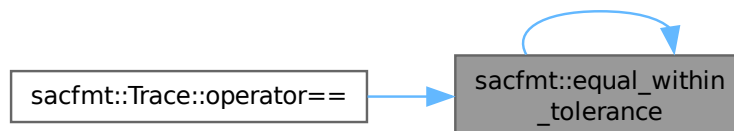
00622                                     {
00623     if (vector1.size() != vector2.size()) {
00624         return false;
00625     }
00626     for (size_t i{0}; i < vector1.size(); ++i) [[likely]] {
00627         if (!equal_within_tolerance(vector1[i], vector2[i], tolerance)) {
00628             return false;
00629         }
00630     }
00631     return true;
00632 }

```

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	<i>num</i>	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 point location1,
    const point location2 ) [noexcept]
```

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

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

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

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

Parameters

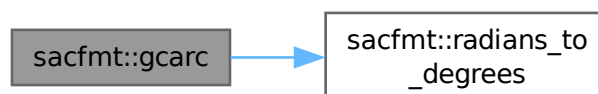
in	<code>location1</code>	point of first location.
in	<code>location2</code>	point of second location

Returns

double The great circle arc distance in decimal degrees.

```
00735
00736     return radians_to_degrees(
00737         std::acos(std::sin(location1.latitude.radians()) *
00738                 std::sin(location2.latitude.radians()) +
00739                 std::cos(location1.latitude.radians()) *
00740                 std::cos(location2.latitude.radians()) *
00741                 std::cos(location2.longitude.radians() -
00742                 location1.longitude.radians()));
00743 }
```

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	<i>num</i>	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	<i>degrees</i>	Decimal degrees to be constrained.
----	----------------	------------------------------------

Returns

double Value within limits.

```

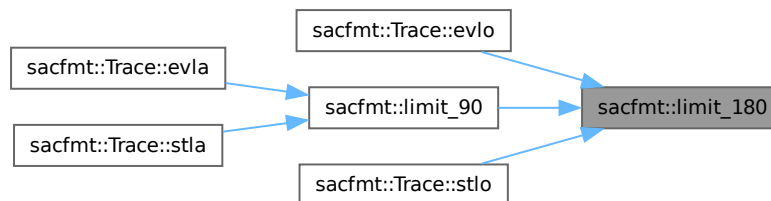
00820
00821 double result{limit_360(degrees)};
00822 constexpr double hemi{180.0};
00823 if (result > hemi) {
00824     result = result - circle_deg;
00825 }
00826 return result;
00827 }

```

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	<i>degrees</i>	Decimal degrees to be constrained.
----	----------------	------------------------------------

Returns

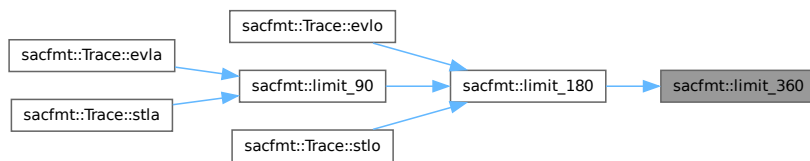
double Value within limits.

```

00794                                     {
00795     double result{degrees};
00796     while (std::abs(result) > circle_deg) {
00797         if (result > circle_deg) {
00798             result -= circle_deg;
00799         } else {
00800             result += circle_deg;
00801         }
00802     }
00803     if (result < 0) {
00804         result += circle_deg;
00805     }
00806     return result;
00807 }

```

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	<i>degrees</i>	Decimal degrees to be constrained.
----	----------------	------------------------------------

Returns

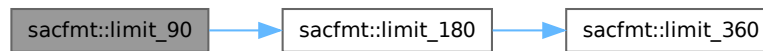
double Value within limits.

```

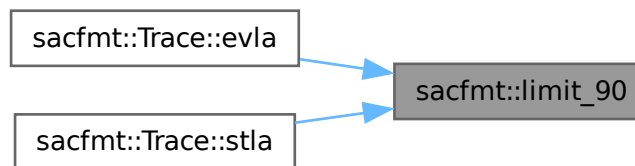
00840                                     {
00841     double result{limit_180(degrees)};
00842     constexpr double quarter{90.0};
00843     if (result > quarter) {
00844         result = (2 * quarter) - result;
00845     } else if (result < -quarter) {
00846         result = (-2 * quarter) - result;
00847     }
00848     return result;
00849 }

```

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

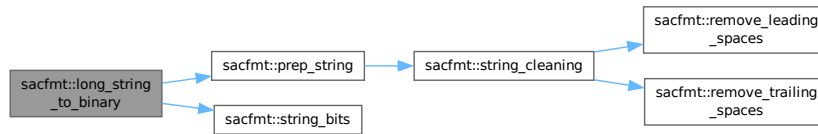
in	<i>str</i>	String to be converted to a bitset.
----	------------	-------------------------------------

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

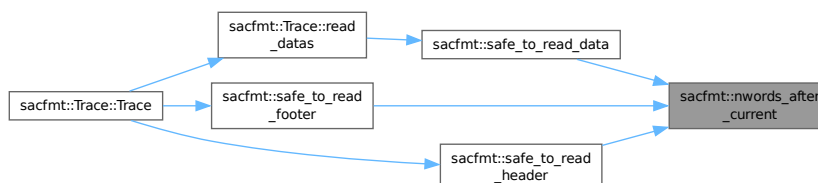
Returns

bool Truth value (true = safe to read).

```

01669                                     {
01670     bool result{false};
01671     if (sac->good()) {
01672         sac->seekg(0, std::ios::end);
01673         const std::size_t final_pos{static_cast<size_t>(sac->tellg())};
01674         // Doesn't like size_t since it wants to allow
01675         // the possibility of negative offsets (not how I use it)
01676         sac->seekg(static_cast<std::streamoff>(spec.start_word));
01677         const std::size_t diff{final_pos - spec.start_word};
01678         result = (diff >= (spec.num_words * word_length));
01679     }
01680     return result;
01681 }
  
```

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.

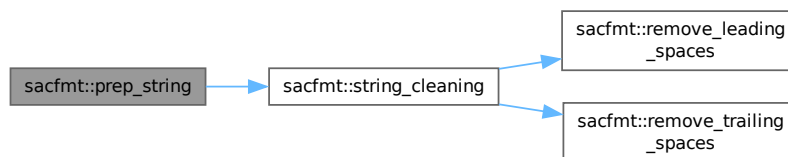
This edits the string in-place.

Parameters

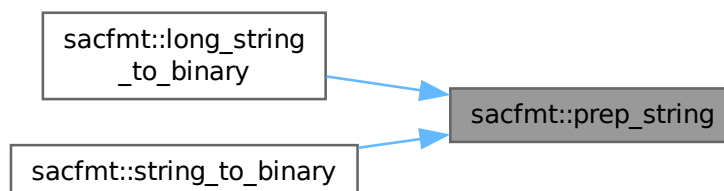
in, out	<i>str</i>	std::string* String to be prepared.
in	<i>str_size</i>	Desired string length.

```
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

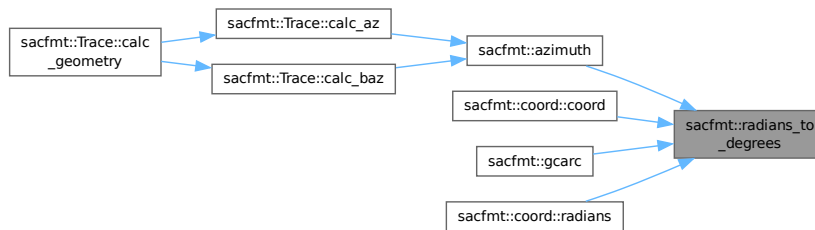
in	<i>radians</i>	Angle in radians to be converted.
----	----------------	-----------------------------------

Returns

double Angle in decimal degrees.

```
00673                                     {
00674     return deg_per_rad * radians;
00675 }
```

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	<i>sac</i>	std::ifstream* Input binary SAC-file.
in	<i>spec</i>	read_spec Reading specification.

Returns

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

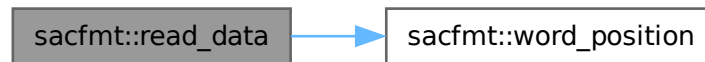
```
00487                                     {
00488     sac->seekg(word_position(spec.start_word));
```

```

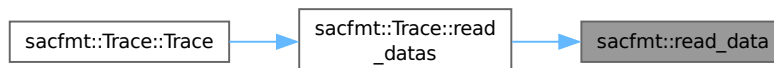
00489     std::vector<double> result{};
00490     result.resize(spec.num_words);
00491     std::for_each(result.begin(), result.end(), [&sac](double &value) {
00492         value = static_cast<double>(binary_to_float(read_word(sac)));
00493     });
00494     return result;
00495 }

```

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

<code>in, out</code>	<code>sac</code>	<code>std::ifstream*</code> 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;

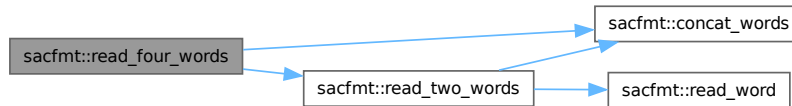
```

```

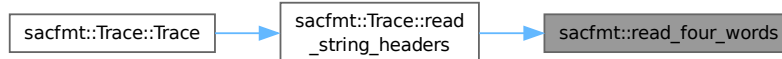
00472     }
00473     return concat_words(pair_words);
00474 }

```

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

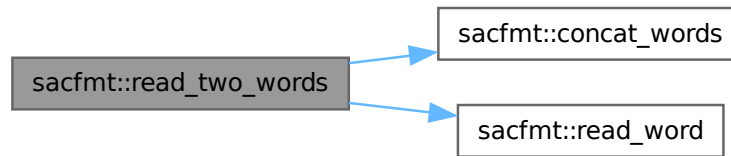
word_two Binary bitset representation of two words.

```

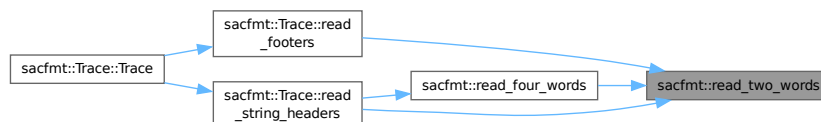
00439     {
00440     const word_one first_word{read_word(sac)};
00441     const word_one second_word{read_word(sac)};
00442     word_pair<word_one> pair_words{};
00443     if constexpr (std::endian::native == std::endian::little) {
00444         pair_words.first = first_word;
00445         pair_words.second = 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.

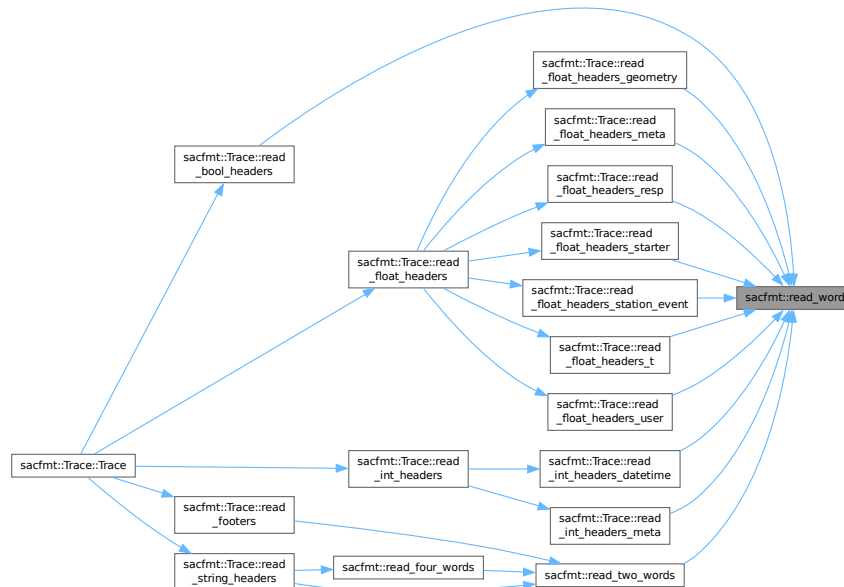
```
00407     {
00408     word_one bits{};
00409     constexpr size_t char_size(bits_per_byte);
00410     // Where we will store the characters
00411     std::array<char, word_length> word{};
00412     // Read to our character array
00413     // This can always hold the source due to careful typing/sizing
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

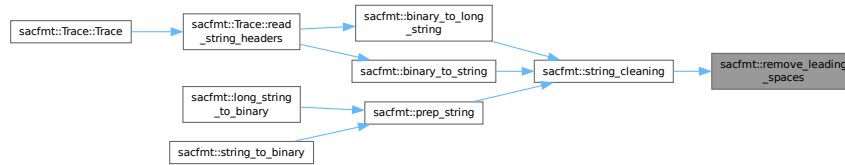
in, out	<i>str</i>	std::string* String to have spaces removed.
---------	------------	---

```

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.

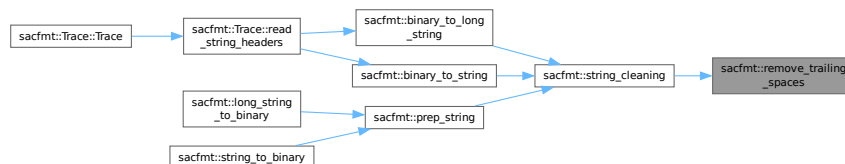
This edits the string in-place.

Parameters

<i>in, out</i>	<i>str</i>	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	<i>sac</i>	std::ifstream* SAC-file to be checked.
----	------------	--

Exceptions

io_error	If the file is not finished.
--------------------------	------------------------------

```

01749 {
01750     const std::streamoff current_pos{sac->tellg()};
01751     sac->seekg(0, std::ios::end);
01752     const std::streamoff end_pos{sac->tellg()};
01753     sac->seekg(current_pos, std::ios::beg);
01754     // How far are we from the end of the file?
01755     const std::streamoff diff{end_pos - current_pos};
01756     // If there is more, something weird happened...
01757     if (diff != 0) {
01758         std::ostringstream oss{};
01759         oss << "Filesize exceeds data specification with ";
01760         oss << diff;
01761         oss << " bytes excess. Data corruption suspected.";
01762         throw io_error(oss.str());
01763     }
01764 }

```

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

Exceptions

io_error	If unsafe to read.
--------------------------	--------------------

```

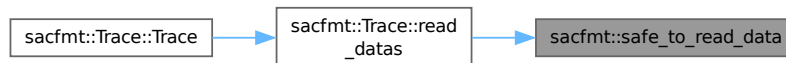
01730
01731     const std::string data{data2 ? "data2" : "data1"};
01732     const read_spec spec{n_words, static_cast<size_t>(sac->tellg())};
01733     if (!nwords_after_current(sac, spec)) {
01734         throw io_error("Insufficient filesize for " + data + '.');
01735     }
01736 }

```

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

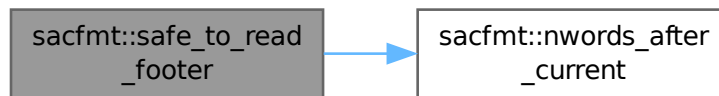
<i>io_error</i>	If unsafe to read.
-----------------	--------------------

```

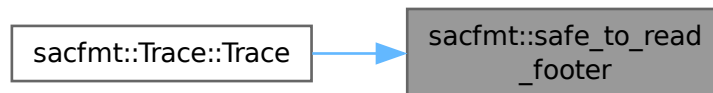
01708
01709     // doubles are two words long
01710     const read_spec spec{static_cast<size_t>(num_footer) * 2,
01711                         static_cast<size_t>(sac->tellg())};
01712     if (!nwords_after_current(sac, spec)) {
01713         throw io_error("Insufficient filesize for footer.");
01714     }
01715 }

```

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

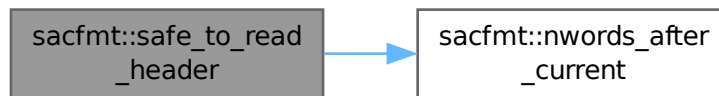
in	sac	std::ifstream* SAC-file to read.
----	-----	----------------------------------

Exceptions

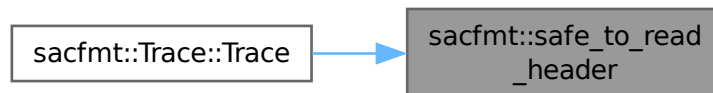
<i>io_error</i>	If unsafe to read.
---------------------------------	--------------------

```
01692                                     {
01693     const read_spec spec{data_word, 0};
01694     if (!nwords_after_current(sac, spec)) {
01695         throw io_error("Insufficient filesize for header.");
01696     }
01697 }
```

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.

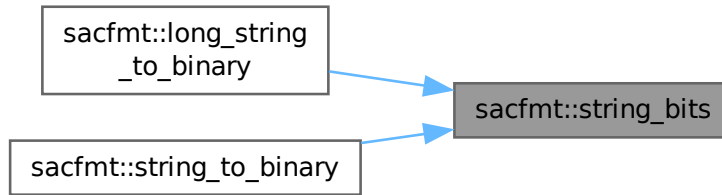
Parameters

out	<i>bits</i>	Destintation bitset for the string (result).
in	<i>str</i>	String to undergo conversion.
in	<i>str_size</i>	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

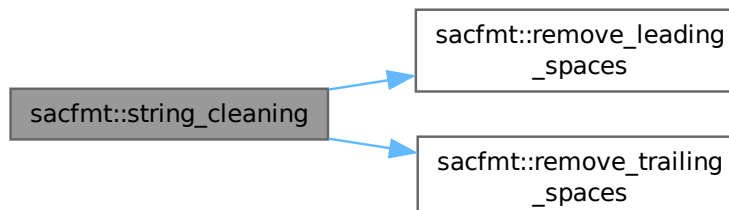
in	<i>str</i>	std::string String to be cleaned.
----	------------	-----------------------------------

Returns

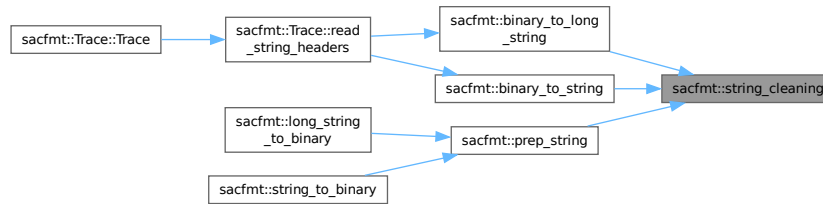
std::string Cleaned string.

```
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

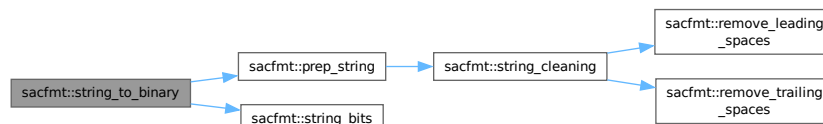
in	<i>str</i>	String to be converted to a bitset.
----	------------	-------------------------------------

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	<i>num</i>	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	<i>word_number</i>	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.

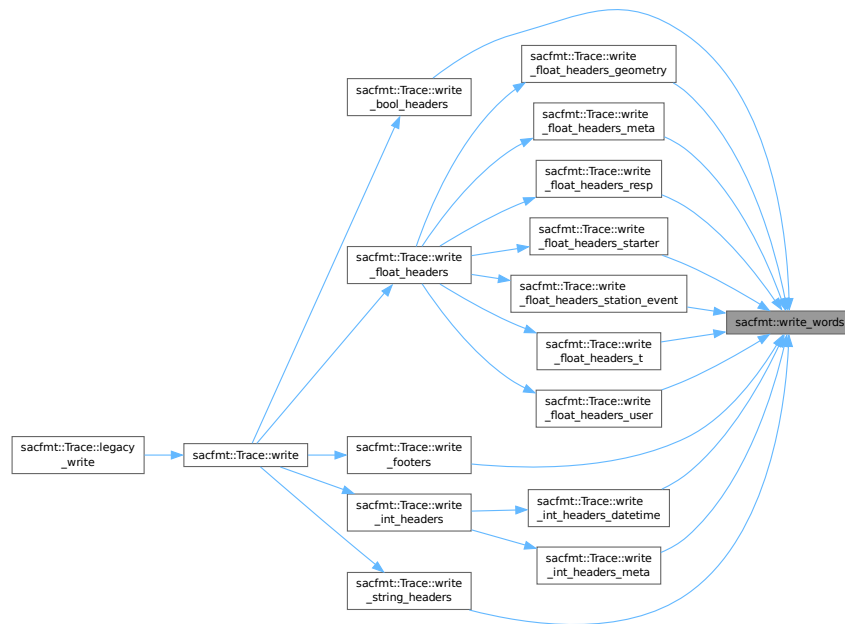
Note that this modifies the position of the writer within the stream (to the end of the written words).

Parameters

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

```
00510                                     {
00511     std::ofstream &sac = *sac_file;
00512     if (sac.is_open()) {
00513         std::for_each(input.begin(), input.end(), [&sac](const char &character) {
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-precision 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.

```
00946 {
00947     // Floats
00948     {name::depmin, 0},
00949     {name::depmax, 1},
00950     {name::odelta, 2},
00951     {name::resp0, 3},
00952     {name::resp1, 4},
00953     {name::resp2, 5},
00954     {name::resp3, 6},
00955     {name::resp4, 7},
00956     {name::resp5, 8},
00957     {name::resp6, 9},
00958     {name::resp7, 10},
00959     {name::resp8, 11},
00960     {name::resp9, 12},
00961     {name::stel, 13},
00962     {name::stdp, 14},
00963     {name::evel, 15},
00964     {name::evdp, 16},
00965     {name::mag, 17},
00966     {name::user0, 18},
00967     {name::user1, 19},
00968     {name::user2, 20},
00969     {name::user3, 21},
00970     {name::user4, 22},
00971     {name::user5, 23},
00972     {name::user6, 24},
00973     {name::user7, 25},
00974     {name::user8, 26},
00975     {name::user9, 27},
00976     {name::dist, 28},
00977     {name::az, 29},
00978     {name::baz, 30},
00979     {name::gcarc, 31},
00980     {name::depmen, 32},
00981     {name::cmpaz, 33},
00982     {name::cmpinc, 34},
00983     {name::xminimum, 35},
00984     {name::xmaximum, 36},
00985     {name::yminimum, 37},
00986     {name::ymaximum, 38},
00987     // Doubles
00988     {name::delta, 0},
00989     {name::b, 1},
00990     {name::e, 2},
00991     {name::o, 3},
00992     {name::a, 4},
00993     {name::t0, 5},
00994     {name::t1, 6},
00995     {name::t2, 7},
00996     {name::t3, 8},
00997     {name::t4, 9},
00998     {name::t5, 10},
00999     {name::t6, 11},
01000     {name::t7, 12},
01001     {name::t8, 13},
01002     {name::t9, 14},
01003     {name::f, 15},
01004     {name::stla, 16},
01005     {name::stlo, 17},
01006     {name::evla, 18},
01007     {name::evlo, 19},
01008     {name::sb, 20},
01009     {name::sdelta, 21},
```

```

01010 // Ints
01011 {name::nzyear, 0},
01012 {name::nzjday, 1},
01013 {name::nzhour, 2},
01014 {name::nzmin, 3},
01015 {name::nzsec, 4},
01016 {name::nzmsec, 5},
01017 {name::nvhdr, 6},
01018 {name::norid, 7},
01019 {name::nevid, 8},
01020 {name::npts, 9},
01021 {name::nsnpts, 10},
01022 {name::nwfid, 11},
01023 {name::nxsize, 12},
01024 {name::nysize, 13},
01025 {name::iftype, 14},
01026 {name::idep, 15},
01027 {name::iztype, 16},
01028 {name::iinst, 17},
01029 {name::istreg, 18},
01030 {name::ievreg, 19},
01031 {name::ievtyp, 20},
01032 {name::igual, 21},
01033 {name::isynth, 22},
01034 {name::imagtyp, 23},
01035 {name::imgsrc, 24},
01036 {name::ibody, 25},
01037 // Bools
01038 {name::leven, 0},
01039 {name::lpspol, 1},
01040 {name::lovrok, 2},
01041 {name::lcalda, 3},
01042 // Strings
01043 {name::kstnm, 0},
01044 {name::kevn, 1},
01045 {name::khole, 2},
01046 {name::ko, 3},
01047 {name::ka, 4},
01048 {name::kt0, 5},
01049 {name::kt1, 6},
01050 {name::kt2, 7},
01051 {name::kt3, 8},
01052 {name::kt4, 9},
01053 {name::kt5, 10},
01054 {name::kt6, 11},
01055 {name::kt7, 12},
01056 {name::kt8, 13},
01057 {name::kt9, 14},
01058 {name::kf, 15},
01059 {name::kuser0, 16},
01060 {name::kuser1, 17},
01061 {name::kuser2, 18},
01062 {name::kcmpnm, 19},
01063 {name::knetwk, 20},
01064 {name::kdatrd, 21},
01065 {name::kinst, 22},
01066 // Data
01067 {name::data1, 0},
01068 {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::coord Class Reference

Defines a geographic coordinant (degrees/radians)

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

Public Member Functions

- [coord \(\) noexcept](#)
Default coordinate constructor.
- [coord \(double value, bool degrees=true\) noexcept](#)
Coordinate constructor.
- [double degrees \(\) const noexcept](#)
Get coordinate value in decimal degrees.
- [double radians \(\) const noexcept](#)
Get coordinate value in radians.
- [void degrees \(double value\) noexcept](#)
Set coordinate value using decimal degrees.
- [void radians \(double value\) noexcept](#)
Set coordainate value using radians.

Private Attributes

- [double deg {}](#)
coordinate value in decimal degrees.
- [double rad {}](#)
coordinate value in radians.

11.1.1 Detailed Description

Defines a geographic coordinant (degrees/radians)

11.1.2 Constructor & Destructor Documentation

11.1.2.1 coord() [1/2]

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

Default coordinate constructor.

11.1.2.2 coord() [2/2]

```
sacfmt::coord::coord (
    double value,
    bool degrees = true ) [explicit], [noexcept]
```

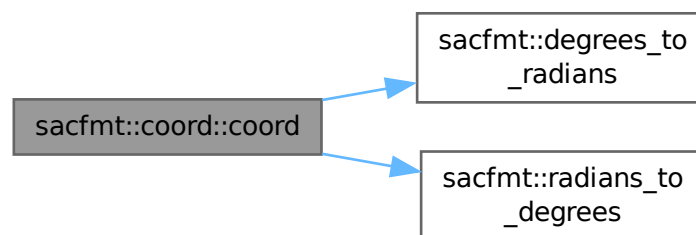
Coordinate constructor.

Parameters

in	<i>value</i>	Double value of coordinate
in	<i>degrees</i>	Boolean value, true if degrees (false = radians).

```
00683                                     {
00684     if (degrees) {
00685         deg = value;
00686         rad = degrees_to_radians(value);
00687     } else {
00688         rad = value;
00689         deg = radians_to_degrees(value);
00690     }
00691 }
```

Here is the call graph for this function:



11.1.3 Member Function Documentation

11.1.3.1 degrees() [1/2]

```
double sacfmt::coord::degrees ( ) const [inline], [noexcept]
```

Get coordinate value in decimal degrees.

```
00269 { return deg; }
```

11.1.3.2 degrees() [2/2]

```
void sacfmt::coord::degrees (
    double value ) [noexcept]
```

Set coordinate value using decimal degrees.

Parameters

in	<i>value</i>	double coordinate in decimal degrees.
----	--------------	---------------------------------------

```
00698 {
00699     deg = value;
00700     rad = degrees_to_radians(value);
00701 }
```

Here is the call graph for this function:



11.1.3.3 radians() [1/2]

```
double sacfmt::coord::radians ( ) const [inline], [noexcept]
```

Get coordinate value in radians.

```
00271 { return rad; }
```

11.1.3.4 radians() [2/2]

```
void sacfmt::coord::radians (
    double value ) [noexcept]
```

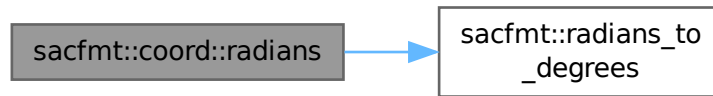
Set coordinate value using radians.

Parameters

in	<i>value</i>	double coordinate in radians.
----	--------------	-------------------------------

```
00708 {
00709     rad = value;
00710     deg = radians_to_degrees(value);
00711 }
```

Here is the call graph for this function:



11.1.4 Member Data Documentation

11.1.4.1 deg

```
double sacfmt::coord::deg {} [private]
```

coordinate value in decimal degrees.

```
00278 {};
```

11.1.4.2 rad

```
double sacfmt::coord::rad {} [private]
```

coordinate value in radians.

```
00280 {};
```

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

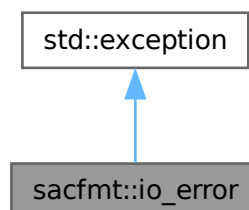
- include/sac-format/sac_format.hpp
- src/sac_format.cpp

11.2 sacfmt::io_error Class Reference

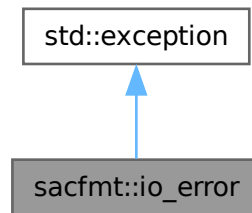
Class for generic I/O exceptions.

```
#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.

11.2.1 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.

11.2.2 Constructor & Destructor Documentation

11.2.2.1 io_error()

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

[io_error](#) Constructor

Parameters

in	<i>msg</i>	std::string Error message.
----	------------	----------------------------

```
01435 : message(std::move(msg)) {}
```

11.2.3 Member Function Documentation

11.2.3.1 what()

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

Error message delivery.

Returns

what char* Error message.

```
01441                                     {
01442     return message.c_str();
01443 }
```

11.2.4 Member Data Documentation

11.2.4.1 message

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

Error message.

```
01427 {};
```

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

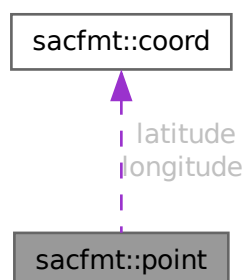
- include/sac-format/sac_format.hpp

11.3 sacfmt::point Struct Reference

Defines a geographic point (latitude, longitude)

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

Collaboration diagram for sacfmt::point:



Public Member Functions

- [point](#) ([coord lat](#), [coord lon](#)) `noexcept`
Construct point from latitude and longitude.

Public Attributes

- [coord latitude](#) {}
Latitude of point.
- [coord longitude](#) {}
Longitude of point.

11.3.1 Detailed Description

Defines a geographic point (latitude, longitude)

11.3.2 Constructor & Destructor Documentation

11.3.2.1 point()

```
sacfmt::point::point (
    coord lat,
    coord lon ) [inline], [noexcept]
```

Construct point from latitude and longitude.

Parameters

in	<i>lat</i>	coord latitude of point.
in	<i>lon</i>	coord longitude of point.

```
00295 : latitude(lat), longitude(lon) {}
```

11.3.3 Member Data Documentation

11.3.3.1 latitude

```
coord sacfmt::point::latitude {}
```

Latitude of point.

```
00286 {};
```

11.3.3.2 longitude

```
coord sacfmt::point::longitude {}
```

Longitude of point.

```
00287 {};
```

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

- include/sac-format/sac_format.hpp

11.4 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.

11.4.1 Detailed Description

Struct that specifies parameters for reading.

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

11.4.2 Member Data Documentation

11.4.2.1 num_words

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

Number of words to read.

```
00211 {};
```

11.4.2.2 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

11.5 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 kevnrm () 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 ival (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`

Static Public Member Functions

- `static void write_data (std::ofstream *sac_file, const std::vector< double > &data_vec)`
Writes data vectors.

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.
- `void read_float_headers_starter (std::ifstream *sac_file)`
Reads SAC-headers from words 000–009.

- [void read_float_headers_t](#) (std::ifstream *sac_file)
Reads SAC-headers from words 010–020.
- [void read_float_headers_resp](#) (std::ifstream *sac_file)
Reads SAC-headers from words 021–030.
- [void read_float_headers_station_event](#) (std::ifstream *sac_file)
Reads SAC-headers from words 031–039.
- [void read_float_headers_user](#) (std::ifstream *sac_file)
Reads SAC-headers from words 040–049.
- [void read_float_headers_geometry](#) (std::ifstream *sac_file)
Reads SAC-headers from words 050–053.
- [void read_float_headers_meta](#) (std::ifstream *sac_file)
Reads SAC-headers from words 054–069.
- [void read_float_headers](#) (std::ifstream *sac_file)
Reads SAC-headers from words 000–069.
- [void read_int_headers_datetime](#) (std::ifstream *sac_file)
Reads SAC-headers from words 070–075.
- [void read_int_headers_meta](#) (std::ifstream *sac_file)
Reads SAC-headers from words 076–104.
- [void read_int_headers](#) (std::ifstream *sac_file)
Reads SAC-headers from words 070–104.
- [void read_bool_headers](#) (std::ifstream *sac_file)
Reads SAC-headers from words 105–109.
- [void read_string_headers](#) (std::ifstream *sac_file)
Reads SAC-headers from words 110–157.
- [void read_datas](#) (std::ifstream *sac_file)
Reads data vectors.
- [void read_footers](#) (std::ifstream *sac_file)
Reads SAC-footers (post-data words 00–43).
- [void write_float_headers_starter](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 000–009.
- [void write_float_headers_t](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 010–020.
- [void write_float_headers_resp](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 021–030.
- [void write_float_headers_station_event](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 031–039.
- [void write_float_headers_user](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 040–049.
- [void write_float_headers_geometry](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 050–053.
- [void write_float_headers_meta](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 054–069.
- [void write_float_headers](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 000–069.
- [void write_int_headers_datetime](#) (std::ofstream *sac_file) const
Writes SAC-headers from words 070–075.
- [void write_int_headers_meta](#) (std::ofstream *sac_file, int hdr_ver) const
Writes SAC-headers from words 076–104.
- [void write_int_headers](#) (std::ofstream *sac_file, int hdr_ver) const
Writes SAC-headers from words 070–104.
- [void write_bool_headers](#) (std::ofstream *sac_file) const

- Writes SAC-headers from words 105–109.*

 - `void write_string_headers (std::ofstream *sac_file) const`

Writes SAC-headers from words 110–157.
- `void write_footers (std::ofstream *sac_file) const`

Writes SAC-footers (post-data words 00–43).
- `bool geometry_set () const noexcept`

Determine if locations are set for geometry calculation.
- `point station_location () const noexcept`

Return station location as a point.
- `point event_location () const noexcept`

Return even location as a point.
- `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.

11.5.1 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.

11.5.2 Constructor & Destructor Documentation

11.5.2.1 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.

```
00861     {
00862     std::fill(floats.begin(), floats.end(), unset_float);
00863     std::fill(doubles.begin(), doubles.end(), unset_double);
00864     std::fill(ints.begin(), ints.end(), unset_int);
00865     std::fill(bools.begin(), bools.end(), unset_bool);
00866     std::fill(strings.begin(), strings.end(), unset_word);
00867 }
```

11.5.2.2 Trace() [2/2]

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

Binary SAC-file reader.

Parameters

in	<i>path</i>	std::filesystem::path SAC-file to be read.
----	-------------	--

Returns

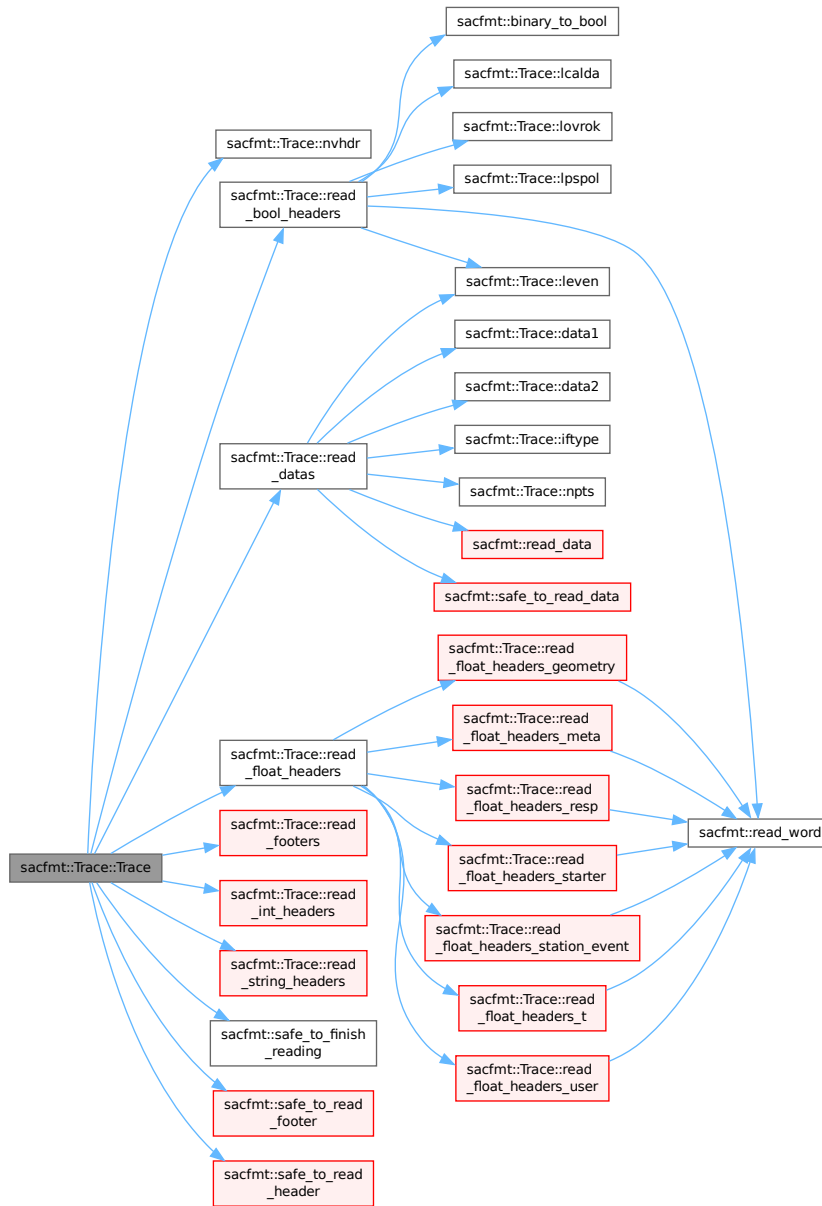
[Trace](#) read in-file.

Exceptions

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

```
02186     {
02187     std::ifstream file(path, std::ifstream::binary);
02188     if (!file) {
02189         throw io_error(path.string() + " cannot be opened to read.");
02190     }
02191     safe_to_read_header(&file); // throws io_error if not safe
02192     read_float_headers(&file);
02193     read_int_headers(&file);
02194     read_bool_headers(&file);
02195     read_string_headers(&file);
02196     read_datas(&file);
02197     if (nvhdr() == modern_hdr_version) {
02198         safe_to_read_footer(&file); // throws io_error if not safe
02199         read_footers(&file);
02200     }
02201     safe_to_finish_reading(&file); // throws io_error if the file isn't finished
02202     file.close();
02203 }
```

Here is the call graph for this function:

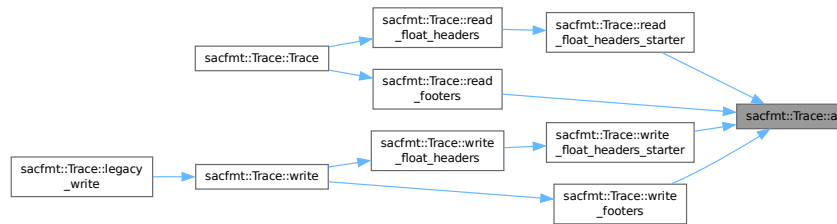


11.5.3 Member Function Documentation

11.5.3.1 a() [1/2]

```
double sacfmt::Trace::a ( ) const [noexcept]
01091 { return doubles[sac_map.at(name::a)]; }
```

Here is the caller graph for this function:



11.5.3.2 a() [2/2]

```

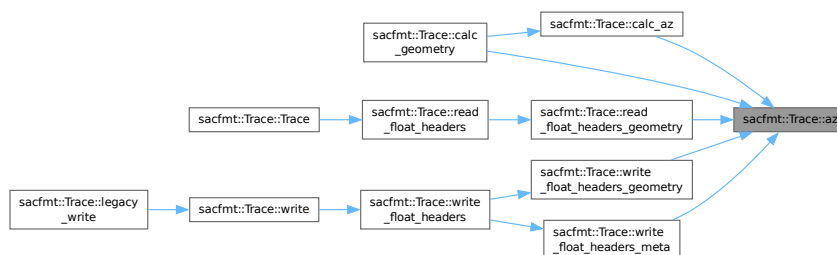
void sacfmt::Trace::a (
    double input ) [noexcept]
{
01346     doubles[sac_map.at(name::a)] = input;
01347 }
01348
  
```

11.5.3.3 az() [1/2]

```

float sacfmt::Trace::az ( ) const [noexcept]
{
01062     return floats[sac_map.at(name::az)];
}
  
```

Here is the caller graph for this function:



11.5.3.4 az() [2/2]

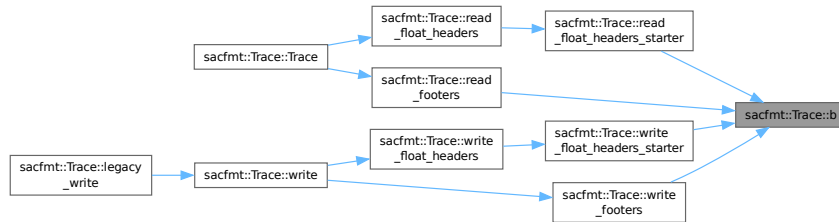
```

void sacfmt::Trace::az (
    float input ) [noexcept]
{
01303     floats[sac_map.at(name::az)] = input;
01304 }
01305
  
```

11.5.3.5 b() [1/2]

```
double sacfmt::Trace::b ( ) const [noexcept]
01088 { return doubles[sac_map.at(name::b)]; }
```

Here is the caller graph for this function:



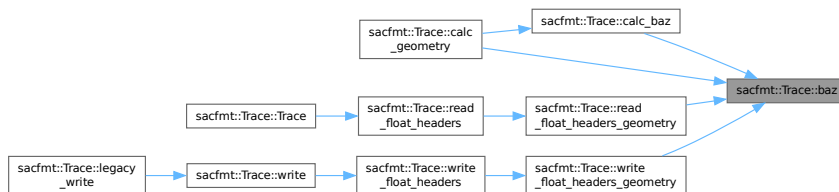
11.5.3.6 b() [2/2]

```
void sacfmt::Trace::b (
    double input ) [noexcept]
01337     {
01338     doubles[sac_map.at(name::b)] = input;
01339 }
```

11.5.3.7 baz() [1/2]

```
float sacfmt::Trace::baz ( ) const [noexcept]
01063 { return floats[sac_map.at(name::baz)]; }
```

Here is the caller graph for this function:



11.5.3.8 baz() [2/2]

```
void sacfmt::Trace::baz (
    float input ) [noexcept]
01306     {
01307     floats[sac_map.at(name::baz)] = input;
01308 }
```

11.5.3.9 calc_az()

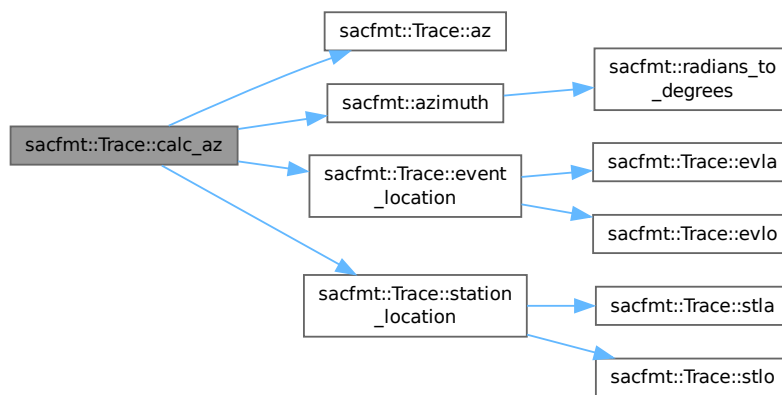
```
void sacfmt::Trace::calc_az ( ) [private], [noexcept]
```

Calculate azimuth.

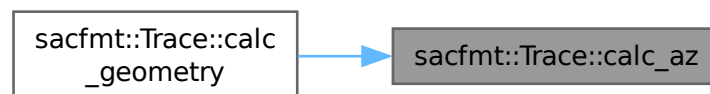
Station → Event

```
00971         {
00972     az(static_cast<float>(azimuth(event_location(), station_location())));
00973 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.10 calc_baz()

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

Calculate back-azimuth.

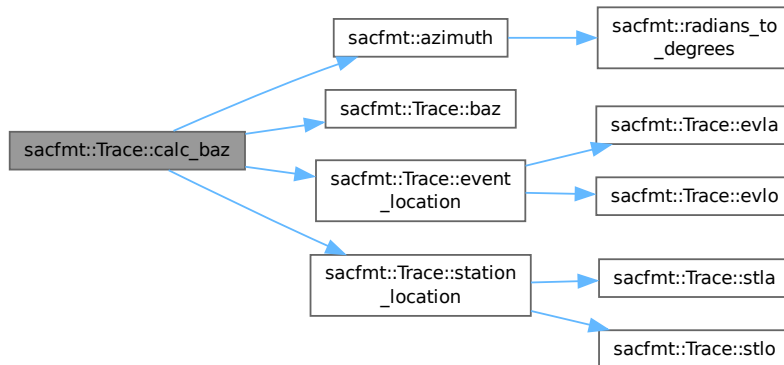
Event → Station

```

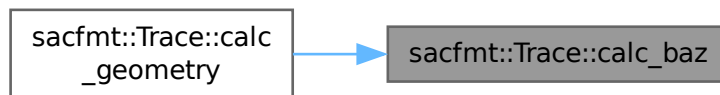
00982         {
00983     baz(static_cast<float>(azimuth(station_location(), event_location())));
00984 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.11 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 planetary bodies).

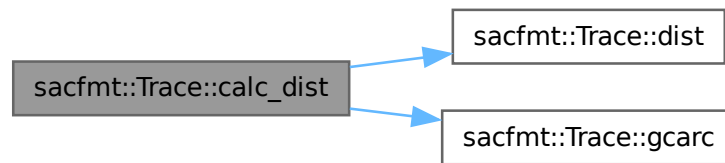
$$d = r_E \cdot \Delta$$

```

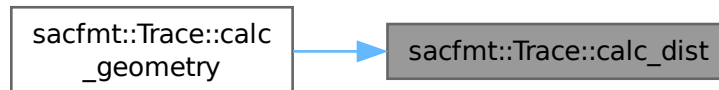
00960         {
00961     dist(static_cast<float>(earth_radius * rad_per_deg * gcarc()));
00962 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



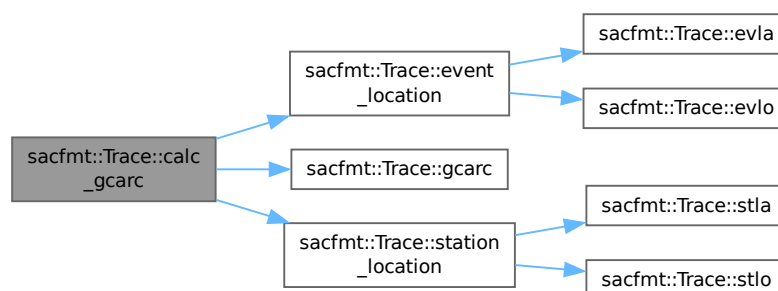
11.5.3.12 calc_gcarc()

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

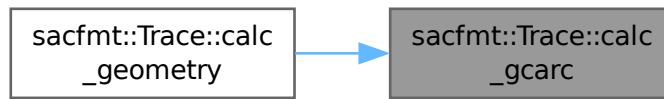
Calculate great-circle arc-distance (gcarc).

```
00945     {
00946     Trace::gcarc(
00947         static_cast<float>(sacfmt::gcarc(station_location(), event_location())));
00948 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.13 calc_geometry()

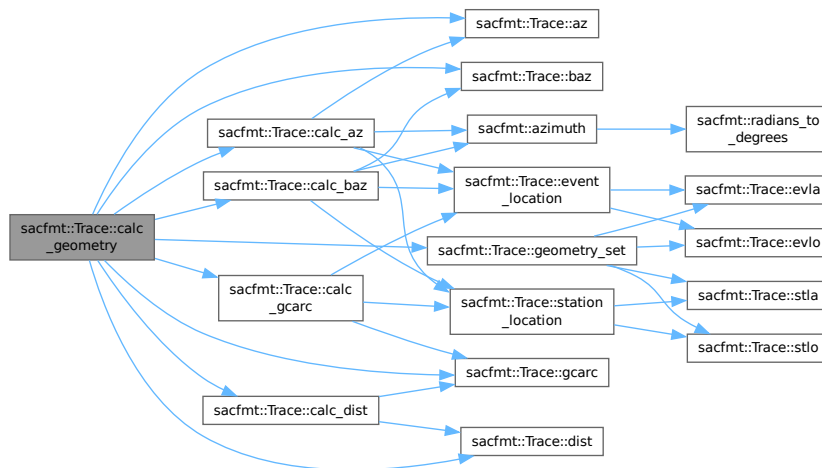
```
void sacfmt::Trace::calc_geometry ( ) [noexcept]
```

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

```

00901     {
00902     if (geometry_set()) {
00903         calc_gcarc();
00904         calc_dist();
00905         calc_az();
00906         calc_baz();
00907     } else {
00908         gcarc(unset_double);
00909         dist(unset_double);
00910         az(unset_double);
00911         baz(unset_double);
00912     }
00913 }
  
```

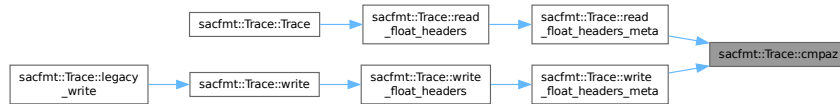
Here is the call graph for this function:



11.5.3.14 cmpaz() [1/2]

```
float sacfmt::Trace::cmpaz ( ) const [noexcept]
01068 { return floats[sac_map.at(name::cmpaz)]; }
```

Here is the caller graph for this function:



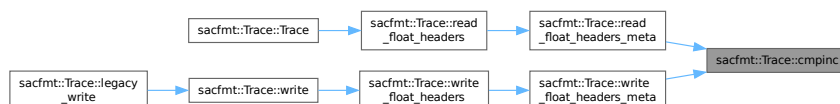
11.5.3.15 cmpaz() [2/2]

```
void sacfmt::Trace::cmpaz (
    float input ) [noexcept]
01315 {
01316     floats[sac_map.at(name::cmpaz)] = input;
01317 }
```

11.5.3.16 cmpinc() [1/2]

```
float sacfmt::Trace::cmpinc ( ) const [noexcept]
01069 {
01070     return floats[sac_map.at(name::cmpinc)];
01071 }
```

Here is the caller graph for this function:



11.5.3.17 cmpinc() [2/2]

```
void sacfmt::Trace::cmpinc (
    float input ) [noexcept]
01318 {
01319     floats[sac_map.at(name::cmpinc)] = input;
01320 }
```

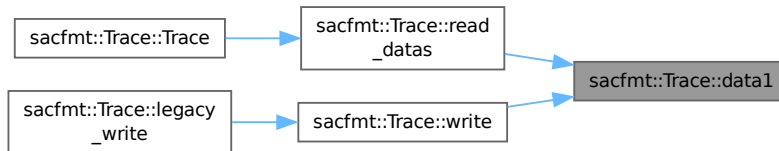
11.5.3.18 data1() [1/2]

```

std::vector< double > sacfmt::Trace::data1 ( ) const [noexcept]
01208     {
01209     return data[sac_map.at(name::data1)];
01210     }

```

Here is the caller graph for this function:



11.5.3.19 data1() [2/2]

```

void sacfmt::Trace::data1 (
    const std::vector< double > & input ) [noexcept]
01596     {
01597     data[sac_map.at(name::data1)] = input;
01598     // Propagate change as needed
01599     int size{static_cast<int>(data1().size())};
01600     size = ((size == 0) && (npts() == unset_int)) ? unset_int : size;
01601     if (size != npts()) {
01602         npts(size);
01603     }
01604     }

```

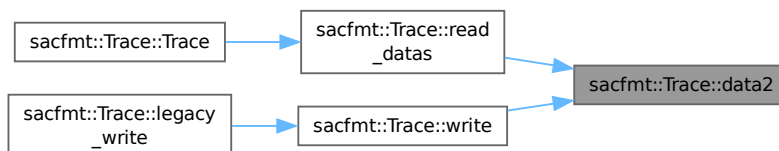
11.5.3.20 data2() [1/2]

```

std::vector< double > sacfmt::Trace::data2 ( ) const [noexcept]
01211     {
01212     return data[sac_map.at(name::data2)];
01213     }

```

Here is the caller graph for this function:



11.5.3.21 data2() [2/2]

```

void sacfmt::Trace::data2 (
    const std::vector< double > & input ) [noexcept]
{
01606     data[sac_map.at(name::data2)] = input;
01607     // Proagate change as needed
01608     int size{static_cast<int>(data2().size())};
01609     size = ((size == 0) && (npts() == unset_int)) ? unset_int : size);
01610     // Need to make sure this is legal
01611     // If positive size and not-legal, make spectral
01612     if (size > 0) {
01613         // If not legal, make spectral
01614         if (leven() && (iftype() <= 1)) {
01615             iftype(2);
01616         }
01617         // If legal and different from npts, update npts
01618         if ((!leven() || (iftype() > 1)) && (size != npts())) {
01619             npts(size);
01620         }
01621     }
01622 }
01623 }

```

11.5.3.22 date()

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

Get date string.

Returns

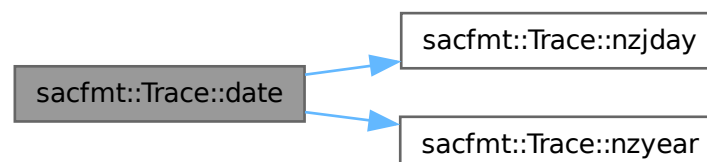
std::string Date (YYYY-JJJ).

```

00991     {
00992     // Require all to be set
00993     if ((nzyear() == unset_int) || (nzjday() == unset_int)) {
00994         return unset_word;
00995     }
00996     std::ostringstream oss{};
00997     oss << nzyear();
00998     oss << '-';
00999     oss << nzjday();
01000     return oss.str();
01001 }

```

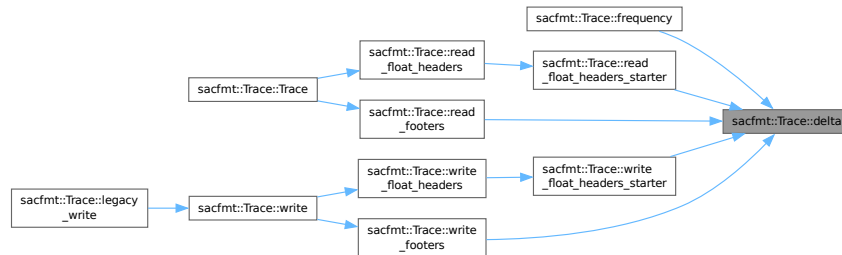
Here is the call graph for this function:



11.5.3.23 delta() [1/2]

```
double sacfmt::Trace::delta ( ) const [noexcept]
01085     {
01086     return doubles[sac_map.at(name::delta)];
01087     }
```

Here is the caller graph for this function:



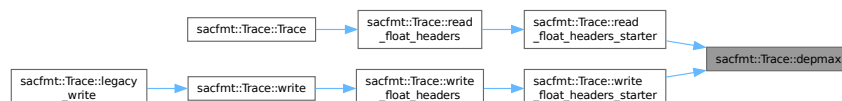
11.5.3.24 delta() [2/2]

```
void sacfmt::Trace::delta (
    double input ) [noexcept]
01334     {
01335     doubles[sac_map.at(name::delta)] = input;
01336     }
```

11.5.3.25 depmax() [1/2]

```
float sacfmt::Trace::depmax ( ) const [noexcept]
01030     {
01031     return floats[sac_map.at(name::depmax)];
01032     }
```

Here is the caller graph for this function:



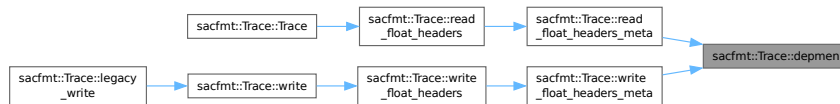
11.5.3.26 depmax() [2/2]

```
void sacfmt::Trace::depmax (
    float input ) [noexcept]
01219     {
01220     floats[sac_map.at(name::depmax)] = input;
01221     }
```

11.5.3.27 depmen() [1/2]

```
float sacfmt::Trace::depmen ( ) const [noexcept]
01065     {
01066     return floats[sac_map.at(name::depmen)];
01067 }
```

Here is the caller graph for this function:



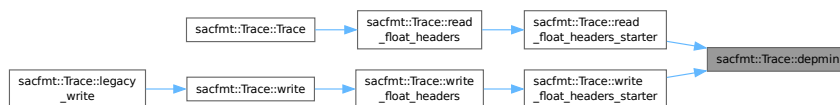
11.5.3.28 depmen() [2/2]

```
void sacfmt::Trace::depmen (
    float input ) [noexcept]
01312     {
01313     floats[sac_map.at(name::depmen)] = input;
01314 }
```

11.5.3.29 depmin() [1/2]

```
float sacfmt::Trace::depmin ( ) const [noexcept]
01027     {
01028     return floats[sac_map.at(name::depmin)];
01029 }
```

Here is the caller graph for this function:



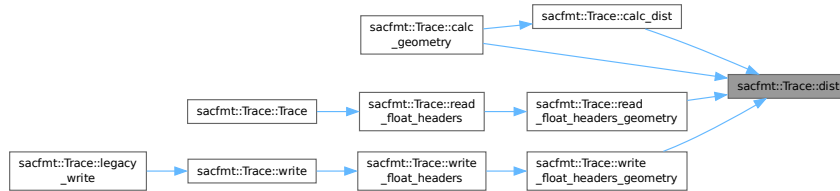
11.5.3.30 depmin() [2/2]

```
void sacfmt::Trace::depmin (
    float input ) [noexcept]
01216     {
01217     floats[sac_map.at(name::depmin)] = input;
01218 }
```

11.5.3.31 `dist()` [1/2]

```
float sacfmt::Trace::dist ( ) const [noexcept]
01061 { return floats[sac_map.at(name::dist)]; }
```

Here is the caller graph for this function:

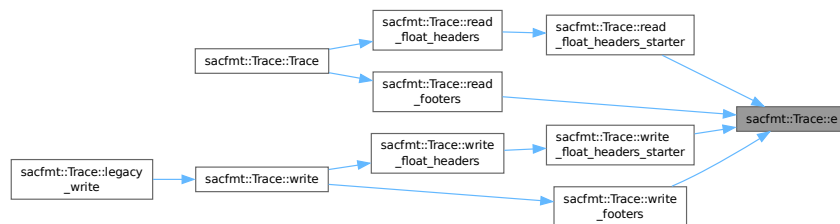
11.5.3.32 `dist()` [2/2]

```
void sacfmt::Trace::dist (
    float input ) [noexcept]
01300 {
01301     floats[sac_map.at(name::dist)] = input;
01302 }
```

11.5.3.33 `e()` [1/2]

```
double sacfmt::Trace::e ( ) const [noexcept]
01089 { return doubles[sac_map.at(name::e)]; }
```

Here is the caller graph for this function:

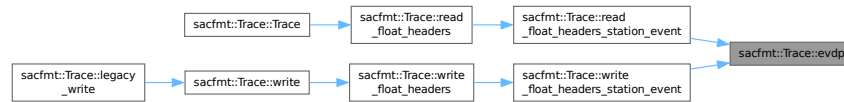
11.5.3.34 `e()` [2/2]

```
void sacfmt::Trace::e (
    double input ) [noexcept]
01340 {
01341     doubles[sac_map.at(name::e)] = input;
01342 }
```

11.5.3.35 evdp() [1/2]

```
float sacfmt::Trace::evdp ( ) const [noexcept]
01049 { return floats[sac_map.at(name::evdp)]; }
```

Here is the caller graph for this function:



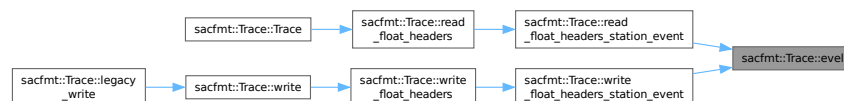
11.5.3.36 evdp() [2/2]

```
void sacfmt::Trace::evdp (
    float input ) [noexcept]
01264 {
01265     floats[sac_map.at(name::evdp)] = input;
01266 }
```

11.5.3.37 evel() [1/2]

```
float sacfmt::Trace::evel ( ) const [noexcept]
01048 { return floats[sac_map.at(name::evel)]; }
```

Here is the caller graph for this function:



11.5.3.38 evel() [2/2]

```
void sacfmt::Trace::evel (
    float input ) [noexcept]
01261 {
01262     floats[sac_map.at(name::evel)] = input;
01263 }
```

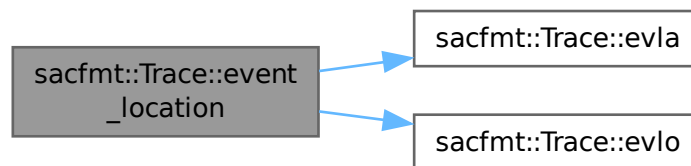

11.5.3.39 event_location()

```
point sacfmt::Trace::event_location ( ) const [inline], [private], [noexcept]
```

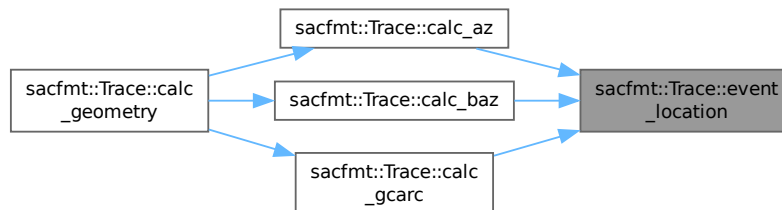
Return even location as a point.

```
01392 {
01393     return point{coord{evla(), true}, coord{evlo(), true}};
01394 }
```

Here is the call graph for this function:



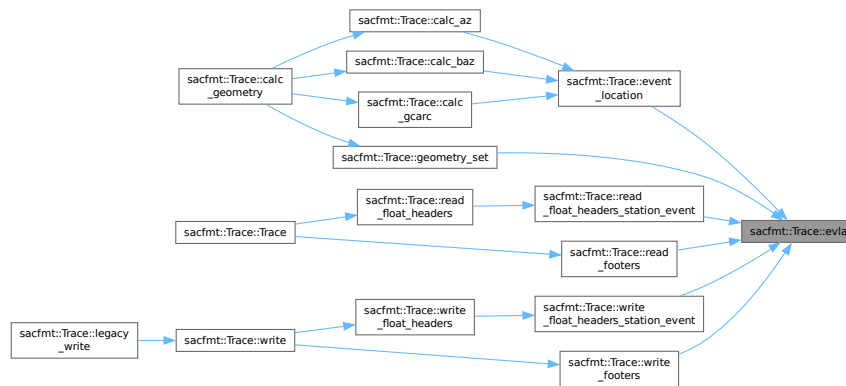
Here is the caller graph for this function:



11.5.3.40 evla() [1/2]

```
double sacfmt::Trace::evla ( ) const [noexcept]
01105 { return doubles[sac_map.at(name::evla)]; }
```

Here is the caller graph for this function:



11.5.3.41 evla() [2/2]

```

void sacfmt::Trace::evla (
    double input ) [noexcept]
{
01396     double clean_input{input};
01397     if (clean_input != unset_double) {
01398         clean_input = limit_90(clean_input);
01399     }
01400     doubles[sac_map.at(name::evla)] = clean_input;
01401 }
01402 }
  
```

Here is the call graph for this function:

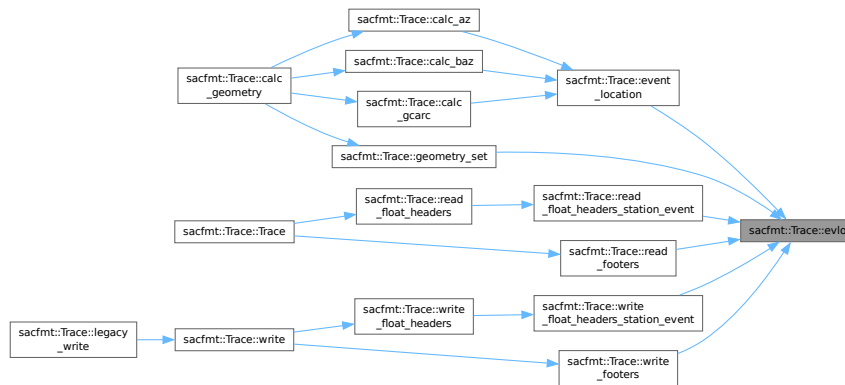


11.5.3.42 evlo() [1/2]

```

double sacfmt::Trace::evlo ( ) const [noexcept]
01106 { return doubles[sac_map.at(name::evlo)]; }
  
```

Here is the caller graph for this function:



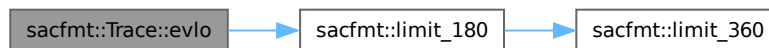
11.5.3.43 evlo() [2/2]

```

void sacfmt::Trace::evlo (
    double input ) [noexcept]
{
01403     double clean_input{input};
01404     if (clean_input != unset_double) {
01405         clean_input = limit_180(clean_input);
01406     }
01407     doubles[sac_map.at(name::evlo)] = clean_input;
01408 }
01409

```

Here is the call graph for this function:



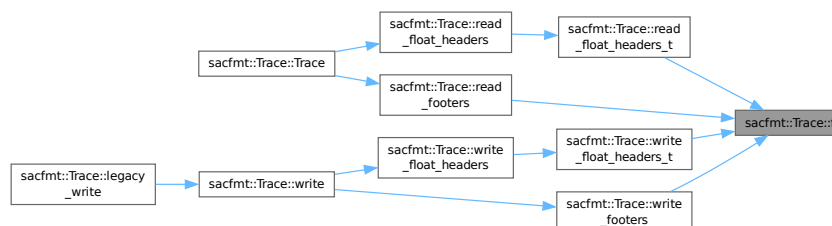
11.5.3.44 f() [1/2]

```

double sacfmt::Trace::f ( ) const [noexcept]
01102 { return doubles[sac_map.at(name::f)]; }

```

Here is the caller graph for this function:



11.5.3.45 f() [2/2]

```
void sacfmt::Trace::f (
    double input ) [noexcept]
01379
01380     doubles[sac_map.at(name::f)] = input;
01381 }
```

11.5.3.46 frequency()

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

Calculate frequency from delta.

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

Returns

double Frequency.

```
00924
00925     const double delta_val{delta()};
00926     if ((delta_val == unset_double) || (delta_val <= 0)) {
00927         return unset_double;
00928     }
00929     return 1.0 / delta_val;
00930 }
```

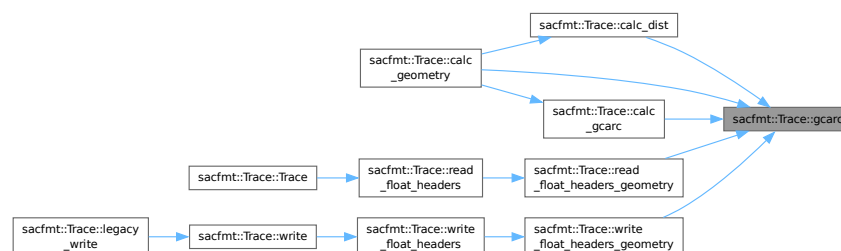
Here is the call graph for this function:



11.5.3.47 gcarc() [1/2]

```
float sacfmt::Trace::gcarc ( ) const [noexcept]
01064 { return floats[sac_map.at(name::gcarc)]; }
```

Here is the caller graph for this function:



11.5.3.48 gcarc() [2/2]

```

void sacfmt::Trace::gcarc (
    float input ) [noexcept]
01309     {
01310     floats[sac_map.at(name::gcarc)] = input;
01311     }

```

11.5.3.49 geometry_set()

```

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

```

Determine if locations are set for geometry calculation.

Returns

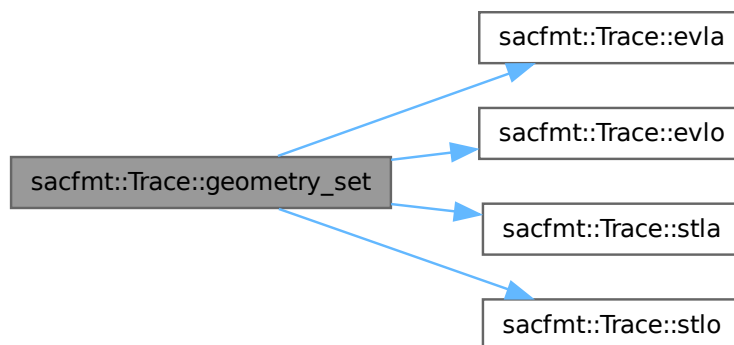
bool True if able to calculate geometry.

```

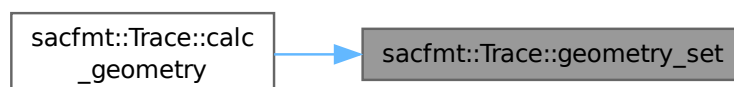
00937     {
00938     return (stla() != unset_double) && (stlo() != unset_double) &&
00939           (evla() != unset_double) && (evlo() != unset_double);
00940     }

```

Here is the call graph for this function:



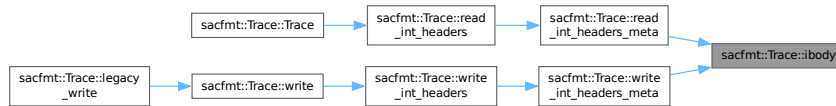
Here is the caller graph for this function:



11.5.3.50 `ibody()` [1/2]

```
int sacfmt::Trace::ibody ( ) const [noexcept]
01137 { return ints[sac_map.at(name::ibody)]; }
```

Here is the caller graph for this function:



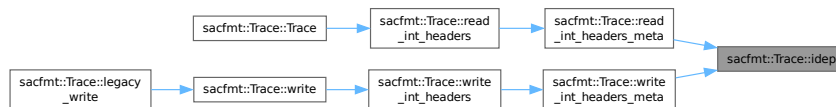
11.5.3.51 `ibody()` [2/2]

```
void sacfmt::Trace::ibody (
    int input ) [noexcept]
01502 {
01503     ints[sac_map.at(name::ibody)] = input;
01504 }
```

11.5.3.52 `idep()` [1/2]

```
int sacfmt::Trace::idep ( ) const [noexcept]
01127 { return ints[sac_map.at(name::idep)]; }
```

Here is the caller graph for this function:



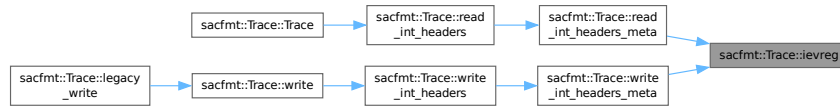
11.5.3.53 `idep()` [2/2]

```
void sacfmt::Trace::idep (
    int input ) [noexcept]
01472 {
01473     ints[sac_map.at(name::idep)] = input;
01474 }
```

11.5.3.54 ievreg() [1/2]

```
int sacfmt::Trace::ievreg ( ) const [noexcept]
01131 { return ints[sac_map.at(name::ievreg)]; }
```

Here is the caller graph for this function:



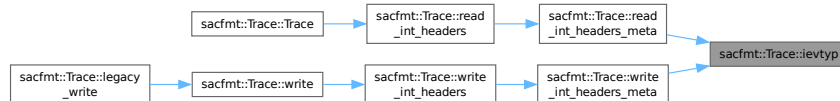
11.5.3.55 ievreg() [2/2]

```
void sacfmt::Trace::ievreg (
    int input ) [noexcept]
01484 {
01485     ints[sac_map.at(name::ievreg)] = input;
01486 }
```

11.5.3.56 ievtyp() [1/2]

```
int sacfmt::Trace::ievtyp ( ) const [noexcept]
01132 { return ints[sac_map.at(name::ievtyp)]; }
```

Here is the caller graph for this function:



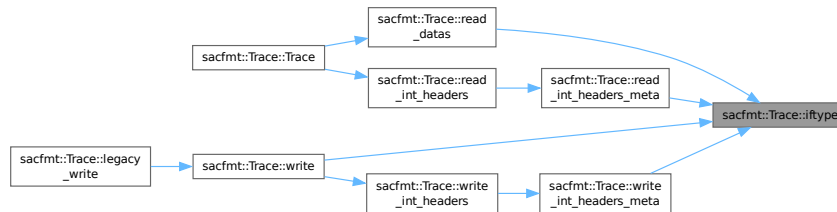
11.5.3.57 ievtyp() [2/2]

```
void sacfmt::Trace::ievtyp (
    int input ) [noexcept]
01487 {
01488     ints[sac_map.at(name::ievtyp)] = input;
01489 }
```

11.5.3.58 iftype() [1/2]

```
int sacfmt::Trace::iftype ( ) const [noexcept]
01126 { return ints[sac_map.at(name::iftype)]; }
```

Here is the caller graph for this function:



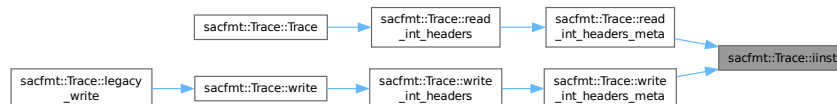
11.5.3.59 iftype() [2/2]

```
void sacfmt::Trace::iftype (
    int input ) [noexcept]
01463     {
01464     ints[sac_map.at(name::iftype)] = input;
01465     const size_t size{npts() >= 0 ? static_cast<size_t>(npts()) : 0};
01466     // Uneven 2D data not supported as not in specification
01467     if ((input > 1) && !leven()) {
01468         leven(true);
01469     }
01470     resize_data2(size);
01471 }
```

11.5.3.60 iinst() [1/2]

```
int sacfmt::Trace::iinst ( ) const [noexcept]
01129 { return ints[sac_map.at(name::iinst)]; }
```

Here is the caller graph for this function:



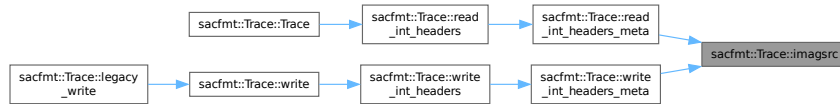
11.5.3.61 iinst() [2/2]

```
void sacfmt::Trace::iinst (
    int input ) [noexcept]
01478     {
01479     ints[sac_map.at(name::iinst)] = input;
01480 }
```


11.5.3.62 `imgsrc()` [1/2]

```
int sacfmt::Trace::imgsrc ( ) const [noexcept]
01136 { return ints[sac_map.at(name::imgsrc)]; }
```

Here is the caller graph for this function:

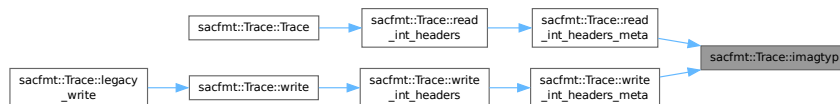
11.5.3.63 `imgsrc()` [2/2]

```
void sacfmt::Trace::imgsrc (
    int input ) [noexcept]
01499 {
01500     ints[sac_map.at(name::imgsrc)] = input;
01501 }
```

11.5.3.64 `imagtyp()` [1/2]

```
int sacfmt::Trace::imagtyp ( ) const [noexcept]
01135 { return ints[sac_map.at(name::imagtyp)]; }
```

Here is the caller graph for this function:

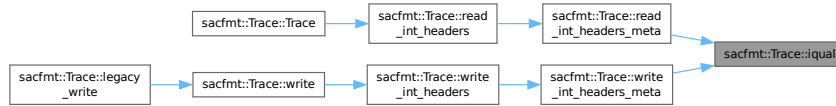
11.5.3.65 `imagtyp()` [2/2]

```
void sacfmt::Trace::imagtyp (
    int input ) [noexcept]
01496 {
01497     ints[sac_map.at(name::imagtyp)] = input;
01498 }
```

11.5.3.66 `equal()` [1/2]

```
int sacfmt::Trace::equal ( ) const [noexcept]
01133 { return ints[sac_map.at(name::equal)]; }
```

Here is the caller graph for this function:



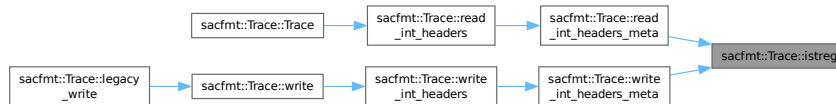
11.5.3.67 `equal()` [2/2]

```
void sacfmt::Trace::equal (
    int input ) [noexcept]
01490 {
01491     ints[sac_map.at(name::equal)] = input;
01492 }
```

11.5.3.68 `istreg()` [1/2]

```
int sacfmt::Trace::istreg ( ) const [noexcept]
01130 { return ints[sac_map.at(name::istreg)]; }
```

Here is the caller graph for this function:



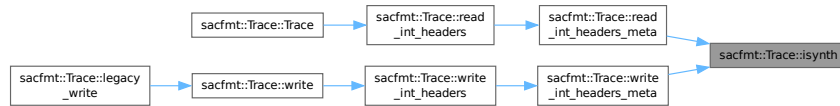
11.5.3.69 `istreg()` [2/2]

```
void sacfmt::Trace::istreg (
    int input ) [noexcept]
01481 {
01482     ints[sac_map.at(name::istreg)] = input;
01483 }
```

11.5.3.70 isynth() [1/2]

```
int sacfmt::Trace::isynt ( ) const [noexcept]
01134 { return ints[sac_map.at(name::isynt)]; }
```

Here is the caller graph for this function:



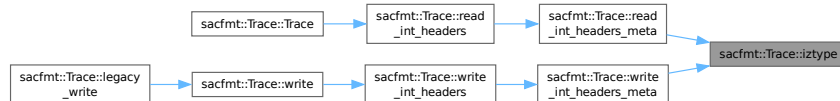
11.5.3.71 isynth() [2/2]

```
void sacfmt::Trace::isynt (
    int input ) [noexcept]
01493 {
01494     ints[sac_map.at(name::isynt)] = input;
01495 }
```

11.5.3.72 iztype() [1/2]

```
int sacfmt::Trace::iztype ( ) const [noexcept]
01128 { return ints[sac_map.at(name::iztype)]; }
```

Here is the caller graph for this function:



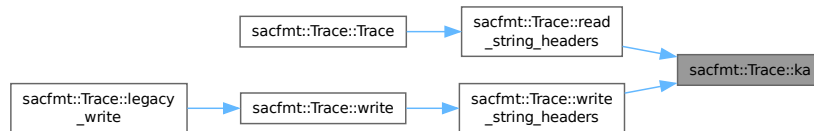
11.5.3.73 iztype() [2/2]

```
void sacfmt::Trace::iztype (
    int input ) [noexcept]
01475 {
01476     ints[sac_map.at(name::iztype)] = input;
01477 }
```

11.5.3.74 ka() [1/2]

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

Here is the caller graph for this function:

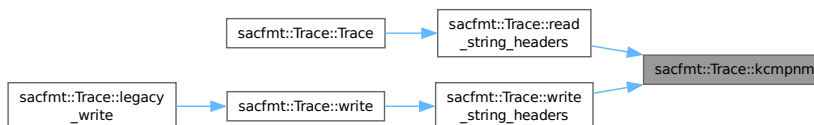
**11.5.3.75 ka()** [2/2]

```
void sacfmt::Trace::ka (
    const std::string & input ) [noexcept]
01537     {
01538     strings[sac_map.at(name::ka)] = input;
01539 }
```

11.5.3.76 kcmpnm() [1/2]

```
std::string sacfmt::Trace::kcmpnm ( ) const [noexcept]
01195     {
01196     return strings[sac_map.at(name::kcmpnm)];
01197 }
```

Here is the caller graph for this function:

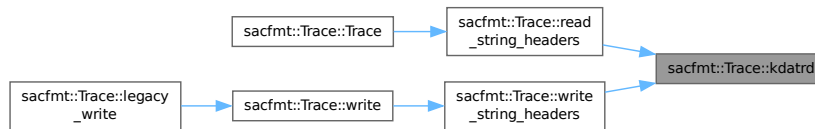
**11.5.3.77 kcmpnm()** [2/2]

```
void sacfmt::Trace::kcmpnm (
    const std::string & input ) [noexcept]
01582     {
01583     strings[sac_map.at(name::kcmpnm)] = input;
01584 }
```

11.5.3.78 kdatrd() [1/2]

```
std::string sacfmt::Trace::kdatrd ( ) const [noexcept]
01201     {
01202     return strings[sac_map.at(name::kdatrd)];
01203     }
```

Here is the caller graph for this function:



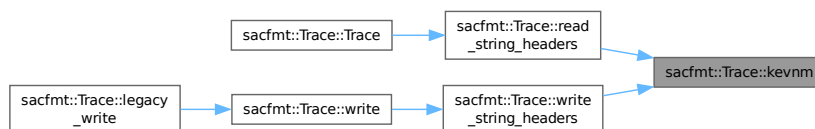
11.5.3.79 kdatrd() [2/2]

```
void sacfmt::Trace::kdatrd (
    const std::string & input ) [noexcept]
01588     {
01589     strings[sac_map.at(name::kdatrd)] = input;
01590     }
```

11.5.3.80 kevnm() [1/2]

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

Here is the caller graph for this function:



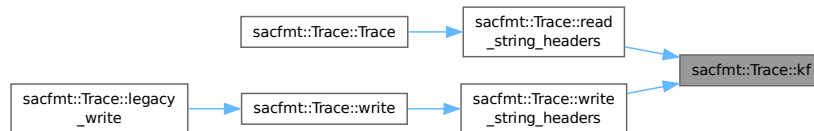
11.5.3.81 kevnm() [2/2]

```
void sacfmt::Trace::kevnm (
    const std::string & input ) [noexcept]
01528     {
01529     strings[sac_map.at(name::kevnm)] = input;
01530     }
```

11.5.3.82 kf() [1/2]

```
std::string sacfmt::Trace::kf ( ) const [noexcept]
01185 { return strings[sac_map.at(name::kf)]; }
```

Here is the caller graph for this function:

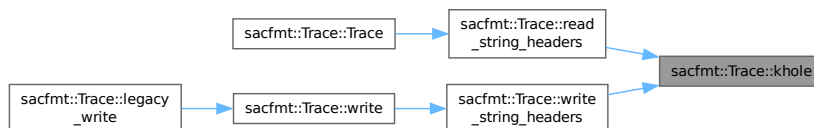
**11.5.3.83 kf()** [2/2]

```
void sacfmt::Trace::kf (
    const std::string & input ) [noexcept]
01570     {
01571     strings[sac_map.at(name::kf)] = input;
01572 }
```

11.5.3.84 khole() [1/2]

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

Here is the caller graph for this function:

**11.5.3.85 khole()** [2/2]

```
void sacfmt::Trace::khole (
    const std::string & input ) [noexcept]
01531     {
01532     strings[sac_map.at(name::khole)] = input;
01533 }
```

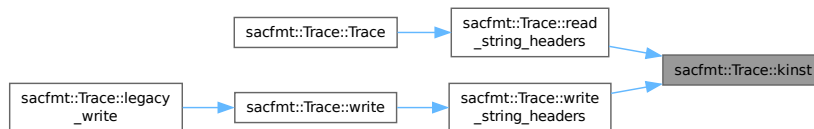
11.5.3.86 `kinst()` [1/2]

```

std::string sacfmt::Trace::kinst ( ) const [noexcept]
01204     {
01205     return strings[sac_map.at(name::kinst)];
01206     }

```

Here is the caller graph for this function:

11.5.3.87 `kinst()` [2/2]

```

void sacfmt::Trace::kinst (
    const std::string & input ) [noexcept]
01591     {
01592     strings[sac_map.at(name::kinst)] = input;
01593     }

```

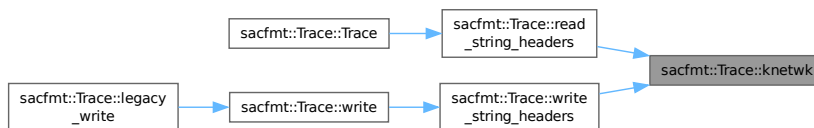
11.5.3.88 `knetwk()` [1/2]

```

std::string sacfmt::Trace::knetwk ( ) const [noexcept]
01198     {
01199     return strings[sac_map.at(name::knetwk)];
01200     }

```

Here is the caller graph for this function:

11.5.3.89 `knetwk()` [2/2]

```

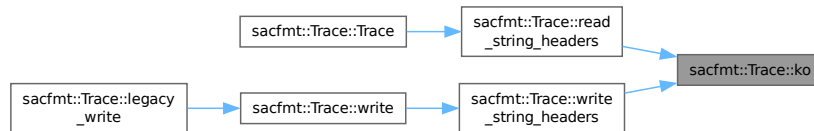
void sacfmt::Trace::knetwk (
    const std::string & input ) [noexcept]
01585     {
01586     strings[sac_map.at(name::knetwk)] = input;
01587     }

```

11.5.3.90 ko() [1/2]

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

Here is the caller graph for this function:

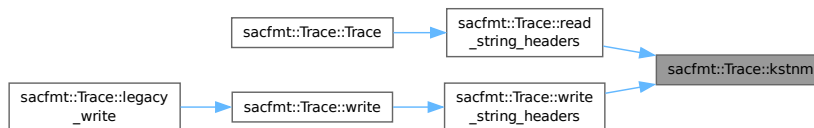
**11.5.3.91 ko()** [2/2]

```
void sacfmt::Trace::ko (
    const std::string & input ) [noexcept]
01534     {
01535     strings[sac_map.at(name::ko)] = input;
01536 }
```

11.5.3.92 kstnm() [1/2]

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

Here is the caller graph for this function:

**11.5.3.93 kstnm()** [2/2]

```
void sacfmt::Trace::kstnm (
    const std::string & input ) [noexcept]
01525     {
01526     strings[sac_map.at(name::kstnm)] = input;
01527 }
```

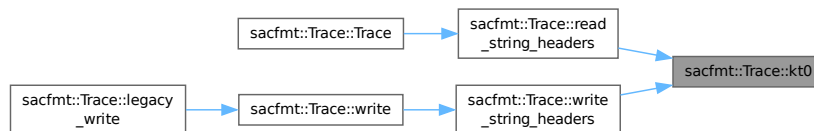

11.5.3.94 kt0() [1/2]

```

std::string sacfmt::Trace::kt0 ( ) const [noexcept]
01155     {
01156     return strings[sac_map.at(name::kt0)];
01157     }

```

Here is the caller graph for this function:



11.5.3.95 kt0() [2/2]

```

void sacfmt::Trace::kt0 (
    const std::string & input ) [noexcept]
01540     {
01541     strings[sac_map.at(name::kt0)] = input;
01542     }

```

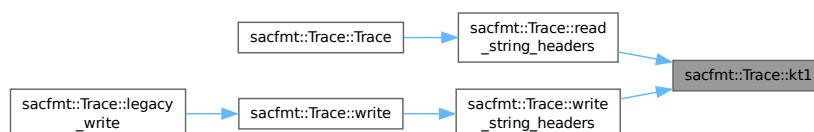
11.5.3.96 kt1() [1/2]

```

std::string sacfmt::Trace::kt1 ( ) const [noexcept]
01158     {
01159     return strings[sac_map.at(name::kt1)];
01160     }

```

Here is the caller graph for this function:



11.5.3.97 kt1() [2/2]

```

void sacfmt::Trace::kt1 (
    const std::string & input ) [noexcept]
01543     {
01544     strings[sac_map.at(name::kt1)] = input;
01545     }

```

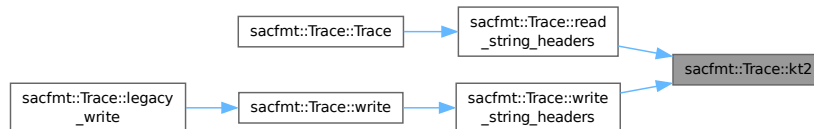
11.5.3.98 kt2() [1/2]

```

std::string sacfmt::Trace::kt2 ( ) const [noexcept]
01161     {
01162     return strings[sac_map.at(name::kt2)];
01163     }

```

Here is the caller graph for this function:

**11.5.3.99 kt2()** [2/2]

```

void sacfmt::Trace::kt2 (
    const std::string & input ) [noexcept]
01546     {
01547     strings[sac_map.at(name::kt2)] = input;
01548     }

```

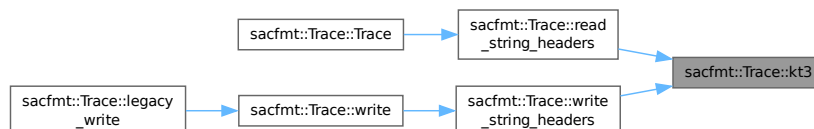
11.5.3.100 kt3() [1/2]

```

std::string sacfmt::Trace::kt3 ( ) const [noexcept]
01164     {
01165     return strings[sac_map.at(name::kt3)];
01166     }

```

Here is the caller graph for this function:

**11.5.3.101 kt3()** [2/2]

```

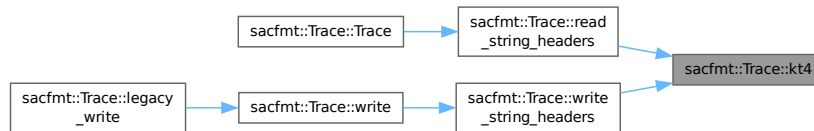
void sacfmt::Trace::kt3 (
    const std::string & input ) [noexcept]
01549     {
01550     strings[sac_map.at(name::kt3)] = input;
01551     }

```

11.5.3.102 kt4() [1/2]

```
std::string sacfmt::Trace::kt4 ( ) const [noexcept]
01167     {
01168     return strings[sac_map.at(name::kt4)];
01169     }
```

Here is the caller graph for this function:



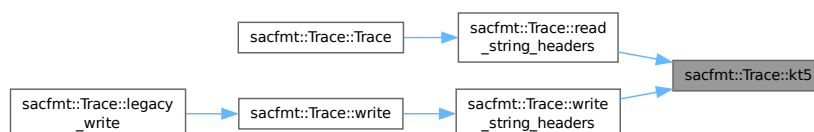
11.5.3.103 kt4() [2/2]

```
void sacfmt::Trace::kt4 (
    const std::string & input ) [noexcept]
01552     {
01553     strings[sac_map.at(name::kt4)] = input;
01554     }
```

11.5.3.104 kt5() [1/2]

```
std::string sacfmt::Trace::kt5 ( ) const [noexcept]
01170     {
01171     return strings[sac_map.at(name::kt5)];
01172     }
```

Here is the caller graph for this function:



11.5.3.105 kt5() [2/2]

```
void sacfmt::Trace::kt5 (
    const std::string & input ) [noexcept]
01555     {
01556     strings[sac_map.at(name::kt5)] = input;
01557     }
```

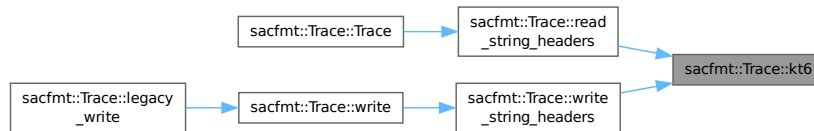
11.5.3.106 kt6() [1/2]

```

std::string sacfmt::Trace::kt6 ( ) const [noexcept]
01173     {
01174     return strings[sac_map.at(name::kt6)];
01175     }

```

Here is the caller graph for this function:

**11.5.3.107 kt6()** [2/2]

```

void sacfmt::Trace::kt6 (
    const std::string & input ) [noexcept]
01558     {
01559     strings[sac_map.at(name::kt6)] = input;
01560     }

```

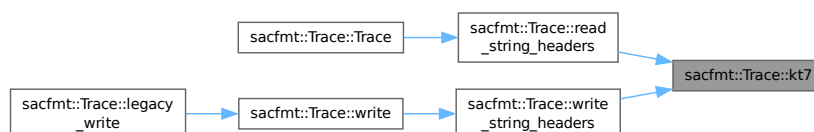
11.5.3.108 kt7() [1/2]

```

std::string sacfmt::Trace::kt7 ( ) const [noexcept]
01176     {
01177     return strings[sac_map.at(name::kt7)];
01178     }

```

Here is the caller graph for this function:

**11.5.3.109 kt7()** [2/2]

```

void sacfmt::Trace::kt7 (
    const std::string & input ) [noexcept]
01561     {
01562     strings[sac_map.at(name::kt7)] = input;
01563     }

```

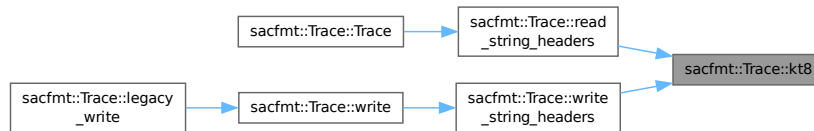
11.5.3.110 kt8() [1/2]

```

std::string sacfmt::Trace::kt8 ( ) const [noexcept]
01179     {
01180     return strings[sac_map.at(name::kt8)];
01181     }

```

Here is the caller graph for this function:



11.5.3.111 kt8() [2/2]

```

void sacfmt::Trace::kt8 (
    const std::string & input ) [noexcept]
01564     {
01565     strings[sac_map.at(name::kt8)] = input;
01566     }

```

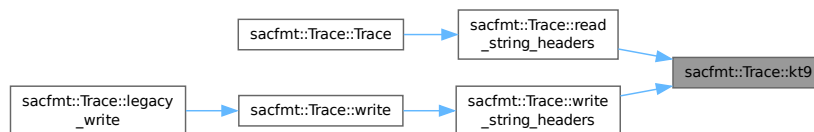
11.5.3.112 kt9() [1/2]

```

std::string sacfmt::Trace::kt9 ( ) const [noexcept]
01182     {
01183     return strings[sac_map.at(name::kt9)];
01184     }

```

Here is the caller graph for this function:



11.5.3.113 kt9() [2/2]

```

void sacfmt::Trace::kt9 (
    const std::string & input ) [noexcept]
01567     {
01568     strings[sac_map.at(name::kt9)] = input;
01569     }

```

11.5.3.114 kuser0() [1/2]

```
std::string sacfmt::Trace::kuser0 ( ) const [noexcept]
01186     {
01187     return strings[sac_map.at(name::kuser0)];
01188     }
```

Here is the caller graph for this function:

**11.5.3.115 kuser0()** [2/2]

```
void sacfmt::Trace::kuser0 (
    const std::string & input ) [noexcept]
01573     {
01574     strings[sac_map.at(name::kuser0)] = input;
01575     }
```

11.5.3.116 kuser1() [1/2]

```
std::string sacfmt::Trace::kuser1 ( ) const [noexcept]
01189     {
01190     return strings[sac_map.at(name::kuser1)];
01191     }
```

Here is the caller graph for this function:

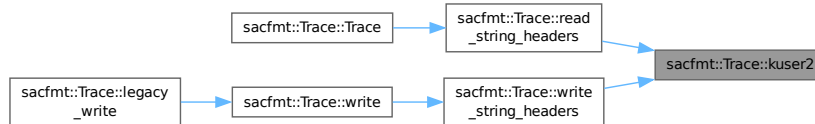
**11.5.3.117 kuser1()** [2/2]

```
void sacfmt::Trace::kuser1 (
    const std::string & input ) [noexcept]
01576     {
01577     strings[sac_map.at(name::kuser1)] = input;
01578     }
```

11.5.3.118 kuser2() [1/2]

```
std::string sacfmt::Trace::kuser2 ( ) const [noexcept]
01192     {
01193     return strings[sac_map.at(name::kuser2)];
01194     }
```

Here is the caller graph for this function:



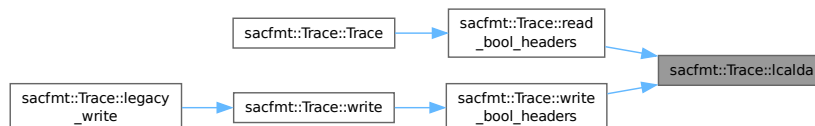
11.5.3.119 kuser2() [2/2]

```
void sacfmt::Trace::kuser2 (
    const std::string & input ) [noexcept]
01579     {
01580     strings[sac_map.at(name::kuser2)] = input;
01581     }
```

11.5.3.120 lcalda() [1/2]

```
bool sacfmt::Trace::lcalda ( ) const [noexcept]
01142 { return bools[sac_map.at(name::lcalda)]; }
```

Here is the caller graph for this function:



11.5.3.121 lcalda() [2/2]

```
void sacfmt::Trace::lcalda (
    bool input ) [noexcept]
01521     {
01522     bools[sac_map.at(name::lcalda)] = input;
01523     }
```

11.5.3.122 legacy_write()

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

Binary SAC-file legacy-write convenience function.

Parameters

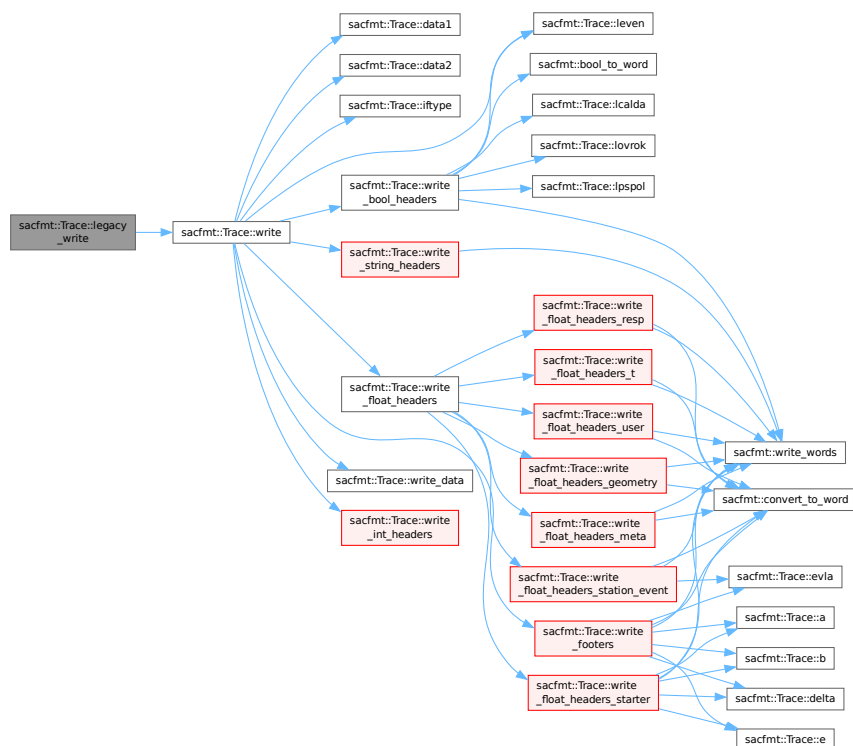
in	path	std::filesystem::path SAC-file to be written.
----	------	---

Exceptions

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

```
02718
02719     write(path, true);
02720 }
```

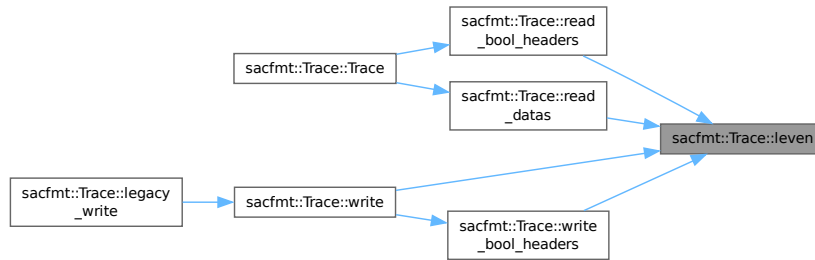
Here is the call graph for this function:



11.5.3.123 leven() [1/2]

```
bool sacfmt::Trace::leven ( ) const [noexcept]
01139 { return bools[sac_map.at (name::leven)]; }
```


Here is the caller graph for this function:



11.5.3.124 leven() [2/2]

```

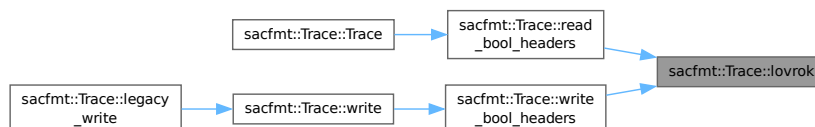
void sacfmt::Trace::leven (
    bool input ) [noexcept]
01506     {
01507     bools[sac_map.at(name::leven)] = input;
01508     const size_t size{npts() >= 0 ? static_cast<size_t>(npts()) : 0};
01509     // Uneven 2D data not supported since not in specification
01510     if (!input && (iftype() > 1)) {
01511         iftype(unset_int);
01512     }
01513     resize_data2(size);
01514 }
  
```

11.5.3.125 lovrok() [1/2]

```

bool sacfmt::Trace::lovrok ( ) const [noexcept]
01141 { return bools[sac_map.at(name::lovrok)]; }
  
```

Here is the caller graph for this function:



11.5.3.126 lovrok() [2/2]

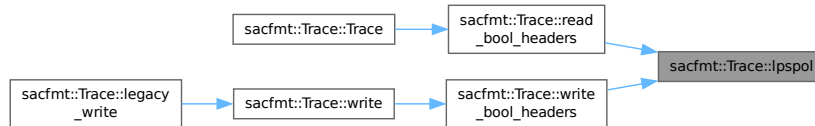
```

void sacfmt::Trace::lovrok (
    bool input ) [noexcept]
01518     {
01519     bools[sac_map.at(name::lovrok)] = input;
01520 }
  
```

11.5.3.127 lpspol() [1/2]

```
bool sacfmt::Trace::lpspol ( ) const [noexcept]
01140 { return bools[sac_map.at(name::lpspol)]; }
```

Here is the caller graph for this function:



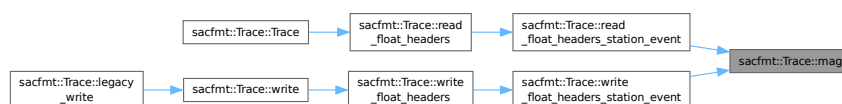
11.5.3.128 lpspol() [2/2]

```
void sacfmt::Trace::lpspol (
    bool input ) [noexcept]
01515     {
01516     bools[sac_map.at(name::lpspol)] = input;
01517 }
```

11.5.3.129 mag() [1/2]

```
float sacfmt::Trace::mag ( ) const [noexcept]
01050 { return floats[sac_map.at(name::mag)]; }
```

Here is the caller graph for this function:



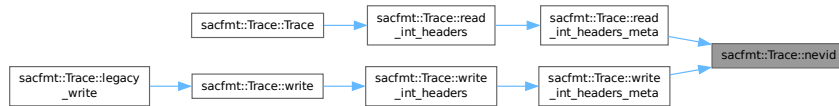
11.5.3.130 mag() [2/2]

```
void sacfmt::Trace::mag (
    float input ) [noexcept]
01267     {
01268     floats[sac_map.at(name::mag)] = input;
01269 }
```

11.5.3.131 nevid() [1/2]

```
int sacfmt::Trace::nevid ( ) const [noexcept]
01120 { return ints[sac_map.at(name::nevid)]; }
```

Here is the caller graph for this function:



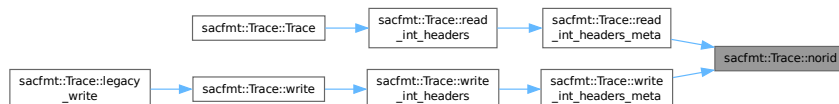
11.5.3.132 nevid() [2/2]

```
void sacfmt::Trace::nevid (
    int input ) [noexcept]
01441 {
01442     ints[sac_map.at(name::nevid)] = input;
01443 }
```

11.5.3.133 norid() [1/2]

```
int sacfmt::Trace::norid ( ) const [noexcept]
01119 { return ints[sac_map.at(name::norid)]; }
```

Here is the caller graph for this function:



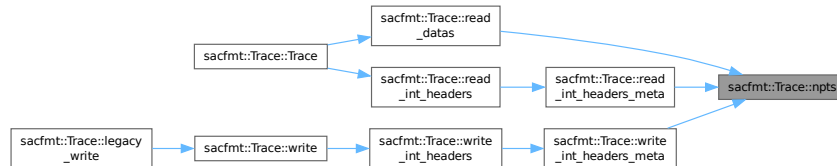
11.5.3.134 norid() [2/2]

```
void sacfmt::Trace::norid (
    int input ) [noexcept]
01438 {
01439     ints[sac_map.at(name::norid)] = input;
01440 }
```

11.5.3.135 npts() [1/2]

```
int sacfmt::Trace::npts ( ) const [noexcept]
01121 { return ints[sac_map.at(name::npts)]; }
```

Here is the caller graph for this function:



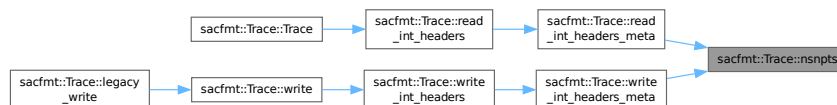
11.5.3.136 npts() [2/2]

```
void sacfmt::Trace::npts (
    int input ) [noexcept]
01444     {
01445     if ((input >= 0) || (input == unset_int)) {
01446         ints[sac_map.at(name::npts)] = input;
01447         const size_t size{static_cast<size_t>(input >= 0 ? input : 0)};
01448         resize_data(size);
01449     }
01450 }
```

11.5.3.137 nsnpts() [1/2]

```
int sacfmt::Trace::nsnpts ( ) const [noexcept]
01122 { return ints[sac_map.at(name::nsnpts)]; }
```

Here is the caller graph for this function:



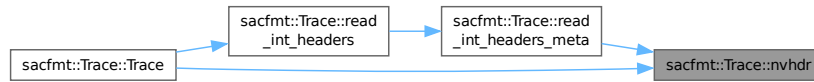
11.5.3.138 nsnpts() [2/2]

```
void sacfmt::Trace::nsnpts (
    int input ) [noexcept]
01451     {
01452     ints[sac_map.at(name::nsnpts)] = input;
01453 }
```

11.5.3.139 nvhdr() [1/2]

```
int sacfmt::Trace::nvhdr ( ) const [noexcept]
01118 { return ints[sac_map.at(name::nvhdr)]; }
```

Here is the caller graph for this function:



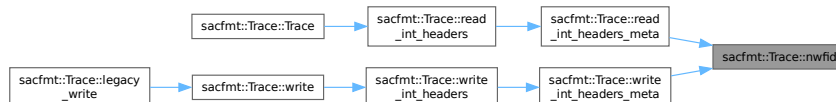
11.5.3.140 nvhdr() [2/2]

```
void sacfmt::Trace::nvhdr (
    int input ) [noexcept]
01435 {
01436     ints[sac_map.at(name::nvhdr)] = input;
01437 }
```

11.5.3.141 nwfid() [1/2]

```
int sacfmt::Trace::nwfid ( ) const [noexcept]
01123 { return ints[sac_map.at(name::nwfid)]; }
```

Here is the caller graph for this function:



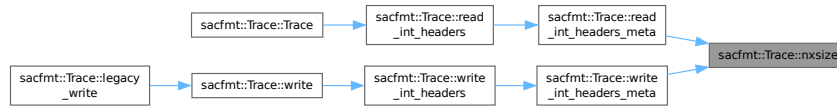
11.5.3.142 nwfid() [2/2]

```
void sacfmt::Trace::nwfid (
    int input ) [noexcept]
01454 {
01455     ints[sac_map.at(name::nwfid)] = input;
01456 }
```

11.5.3.143 nysize() [1/2]

```
int sacfmt::Trace::nysize ( ) const [noexcept]
01124 { return ints[sac_map.at(name::nysize)]; }
```

Here is the caller graph for this function:

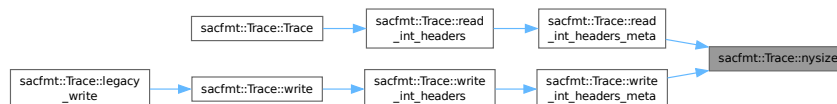
**11.5.3.144 nysize()** [2/2]

```
void sacfmt::Trace::nysize (
    int input ) [noexcept]
01457 {
01458     ints[sac_map.at(name::nysize)] = input;
01459 }
```

11.5.3.145 nysize() [1/2]

```
int sacfmt::Trace::nysize ( ) const [noexcept]
01125 { return ints[sac_map.at(name::nysize)]; }
```

Here is the caller graph for this function:

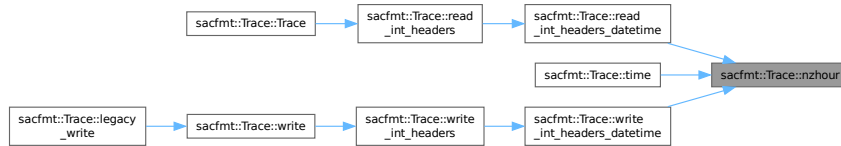
**11.5.3.146 nysize()** [2/2]

```
void sacfmt::Trace::nysize (
    int input ) [noexcept]
01460 {
01461     ints[sac_map.at(name::nysize)] = input;
01462 }
```

11.5.3.147 nzhour() [1/2]

```
int sacfmt::Trace::nzhour ( ) const [noexcept]
01114 { return ints[sac_map.at(name::nzhour)]; }
```

Here is the caller graph for this function:



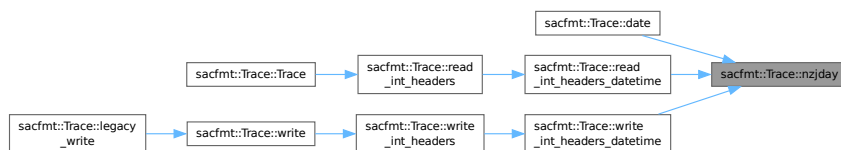
11.5.3.148 nzhour() [2/2]

```
void sacfmt::Trace::nzhour (
    int input ) [noexcept]
01423     {
01424     ints[sac_map.at(name::nzhour)] = input;
01425 }
```

11.5.3.149 nzjday() [1/2]

```
int sacfmt::Trace::nzjday ( ) const [noexcept]
01113 { return ints[sac_map.at(name::nzjday)]; }
```

Here is the caller graph for this function:



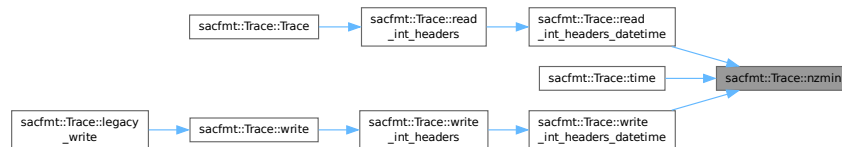
11.5.3.150 nzjday() [2/2]

```
void sacfmt::Trace::nzjday (
    int input ) [noexcept]
01420     {
01421     ints[sac_map.at(name::nzjday)] = input;
01422 }
```

11.5.3.151 nzmin() [1/2]

```
int sacfmt::Trace::nzmin ( ) const [noexcept]
01115 { return ints[sac_map.at(name::nzmin)]; }
```

Here is the caller graph for this function:



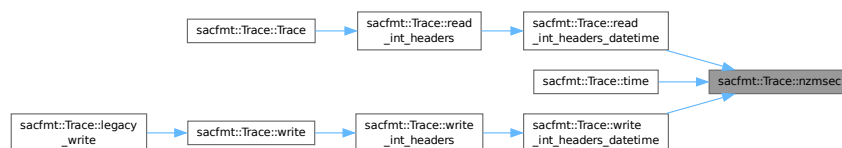
11.5.3.152 nzmin() [2/2]

```
void sacfmt::Trace::nzmin (
    int input ) [noexcept]
01426     {
01427     ints[sac_map.at(name::nzmin)] = input;
01428 }
```

11.5.3.153 nzmsec() [1/2]

```
int sacfmt::Trace::nzmsec ( ) const [noexcept]
01117 { return ints[sac_map.at(name::nzmsec)]; }
```

Here is the caller graph for this function:



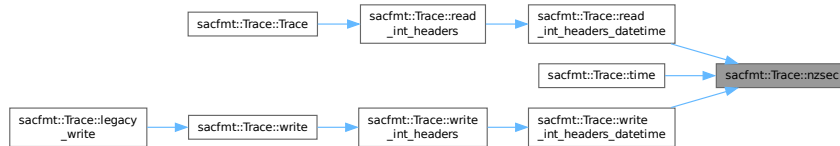
11.5.3.154 nzmsec() [2/2]

```
void sacfmt::Trace::nzmsec (
    int input ) [noexcept]
01432     {
01433     ints[sac_map.at(name::nzmsec)] = input;
01434 }
```


11.5.3.155 nzsec() [1/2]

```
int sacfmt::Trace::nzsec ( ) const [noexcept]
01116 { return ints[sac_map.at(name::nzsec)]; }
```

Here is the caller graph for this function:



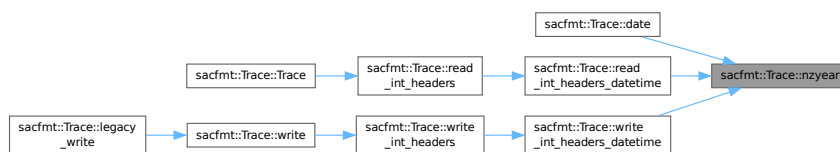
11.5.3.156 nzsec() [2/2]

```
void sacfmt::Trace::nzsec (
    int input ) [noexcept]
01429     {
01430     ints[sac_map.at(name::nzsec)] = input;
01431 }
```

11.5.3.157 nzyear() [1/2]

```
int sacfmt::Trace::nzyear ( ) const [noexcept]
01112 { return ints[sac_map.at(name::nzyear)]; }
```

Here is the caller graph for this function:



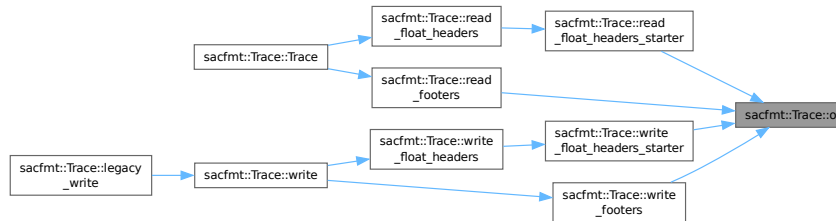
11.5.3.158 nzyear() [2/2]

```
void sacfmt::Trace::nzyear (
    int input ) [noexcept]
01417     {
01418     ints[sac_map.at(name::nzyear)] = input;
01419 }
```

11.5.3.159 o() [1/2]

```
double sacfmt::Trace::o ( ) const [noexcept]
01090 { return doubles[sac_map.at(name::o)]; }
```

Here is the caller graph for this function:

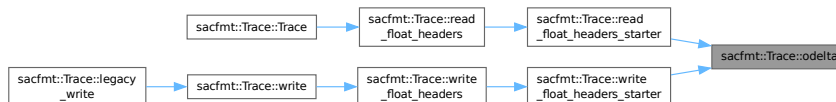
**11.5.3.160 o() [2/2]**

```
void sacfmt::Trace::o (
    double input ) [noexcept]
01343 {
01344     doubles[sac_map.at(name::o)] = input;
01345 }
```

11.5.3.161 odelta() [1/2]

```
float sacfmt::Trace::odelta ( ) const [noexcept]
01033 {
01034     return floats[sac_map.at(name::odelta)];
01035 }
```

Here is the caller graph for this function:

**11.5.3.162 odelta() [2/2]**

```
void sacfmt::Trace::odelta (
    float input ) [noexcept]
01222 {
01223     floats[sac_map.at(name::odelta)] = input;
01224 }
```

11.5.3.163 operator==()

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

[Trace](#) equality operator.

Parameters

in	<i>this</i>	First Trace in comparison (LHS).
in	<i>other</i>	Second Trace in comparison (RHS).

Returns

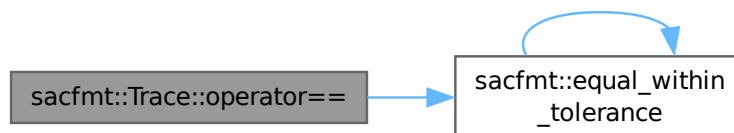
bool Truth value of equality.

```

00876                                     {
00877     if (floats != other.floats) {
00878         return false;
00879     }
00880     if (doubles != other.doubles) {
00881         return false;
00882     }
00883     if (ints != other.ints) {
00884         return false;
00885     }
00886     if (strings != other.strings) {
00887         return false;
00888     }
00889     if (!equal_within_tolerance(data[0], other.data[0])) {
00890         return false;
00891     }
00892     if (!equal_within_tolerance(data[1], other.data[1])) {
00893         return false;
00894     }
00895     return true;
00896 }

```

Here is the call graph for this function:



11.5.3.164 read_bool_headers()

```

void sacfmt::Trace::read_bool_headers (
    std::ifstream * sac_file ) [private]

```

Reads SAC-headers from words 105–109.

Note that this expects the position of the reader to be the beginning of word 105.

Note that this modifies the position of the reader to the end of word 109.

Loads all boolean headers.

Parameters

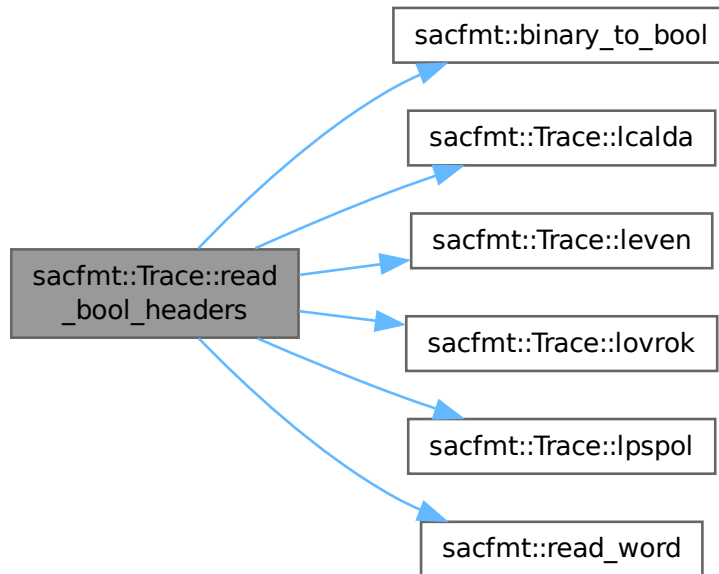
in, out	<i>sac_file</i>	std::ifstream* SAC-file to be read.
---------	-----------------	-------------------------------------

```

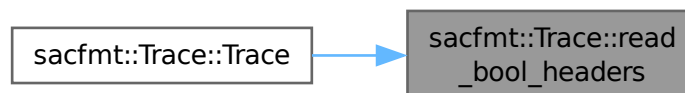
02062                                     {
02063     // Logical headers
02064     leven(binary_to_bool(read_word(sac_file))); // 105
02065     lpspol(binary_to_bool(read_word(sac_file))); // 106
02066     lovrok(binary_to_bool(read_word(sac_file))); // 107
02067     lcalda(binary_to_bool(read_word(sac_file))); // 108
02068     // Skip 'unused'
02069     read_word(sac_file); // 109
02070 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.165 read_datas()

```

void sacfmt::Trace::read_datas (
    std::ifstream * sac_file ) [private]

```

Reads data vectors.

Note that this modifies the position of the reader to the end of the data section(s).

For data1 reads words 158–(158 + npts).

For data2 reads words (158 + 1 + npts)–(159 + (2 * npts))

Parameters

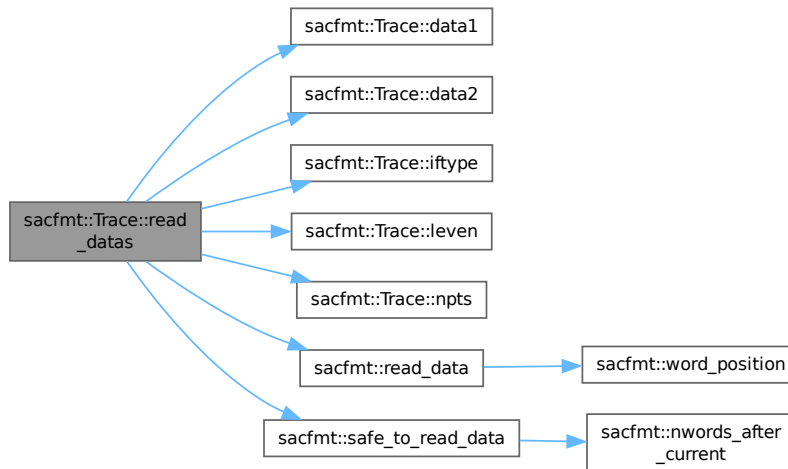
<code>in, out</code>	<code>sac_file</code>	<code>std::ifstream*</code> SAC-file to be read.
----------------------	-----------------------	--

```

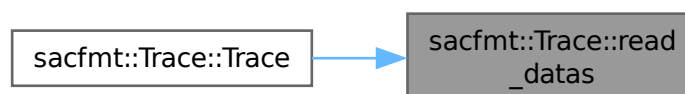
02125
02126     const bool is_data{npts() != unset_int}; {
02127     // data1
02128     const size_t n_words{static_cast<size_t>(npts())};
02129     if (is_data) {
02130         // false flags for data1
02131         safe_to_read_data(sac_file, n_words, false); // throws io_error if unsafe
02132         const read_spec spec{n_words, data_word};
02133         // Originally floats, read as doubles
02134         data1(read_data(sac_file, spec));
02135     }
02136     // data2 (uneven or spectral data)
02137     if (is_data && (!leven() || (iftype() > 1))) {
02138         // true flags for data2
02139         safe_to_read_data(sac_file, n_words, true); // throws io_error if unsafe
02140         const read_spec spec{n_words, data_word + static_cast<size_t>(npts())};
02141         data2(read_data(sac_file, spec));
02142     }
02143 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.166 read_float_headers()

```
void sacfmt::Trace::read_float_headers (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 000–069.

Note that this expects the position of the reader to be the beginning of word 000.

Note that this modifies the position of the reader to the end of word 069.

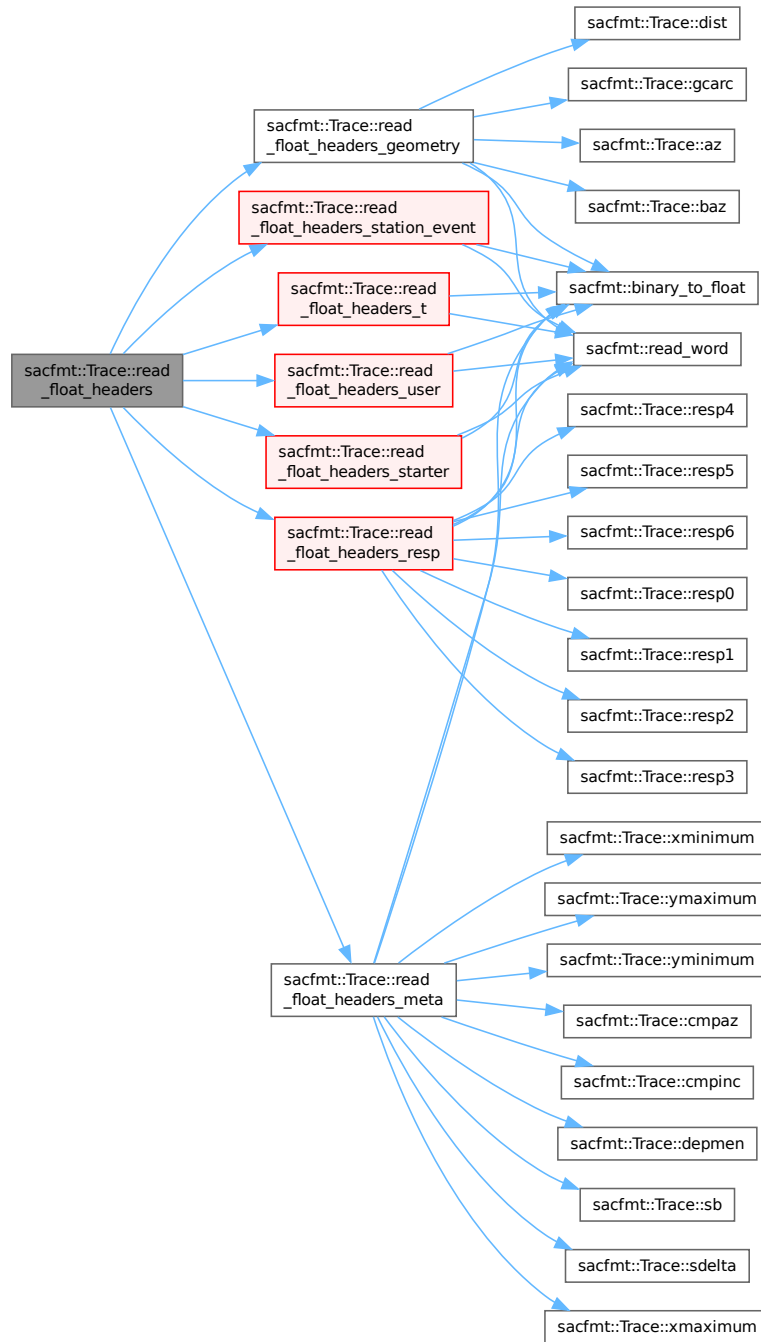
Loads all the float headers.

Parameters

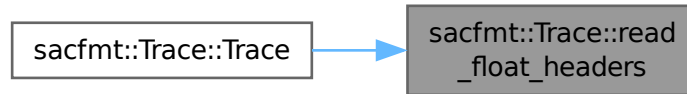
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
01957
01958   read_float_headers_starter(sac_file);           // 000-009
01959   read_float_headers_t(sac_file);                 // 010-020
01960   read_float_headers_resp(sac_file);              // 021-030
01961   read_float_headers_station_event(sac_file);     // 031-039
01962   read_float_headers_user(sac_file);              // 040-049
01963   read_float_headers_geometry(sac_file);          // 050-053
01964   read_float_headers_meta(sac_file);              // 054-069
01965 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.167 read_float_headers_geometry()

```
void sacfmt::Trace::read_float_headers_geometry (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 050–053.

Note that this expects the position of the reader to be the beginning of word 050.

Note that this modifies the position of the reader to the end of word 053.

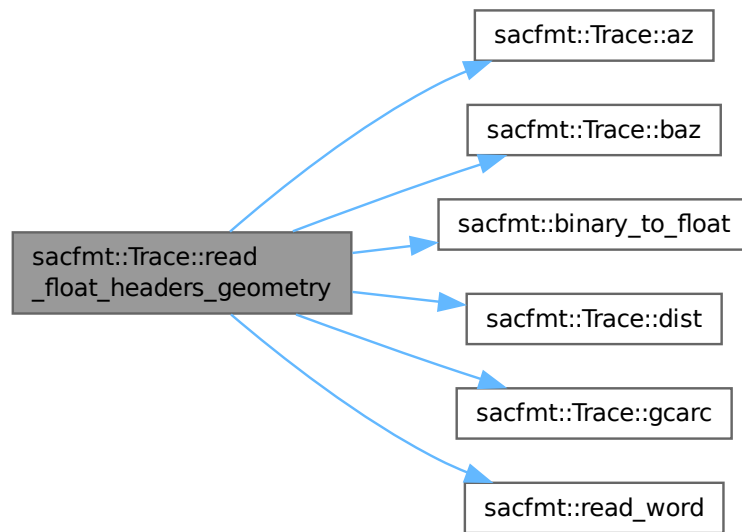
Headers loaded: dist, az, baz, and gcarc.

Parameters

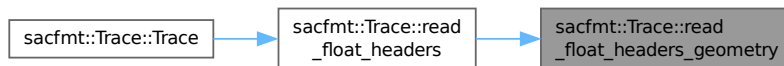
<i>in, out</i>	<i>sac_file</i>	std::ifstream* SAC-file to be read.
----------------	-----------------	-------------------------------------

```
01909
01910     dist(binary_to_float(read_word(sac_file))); // 050
01911     az(binary_to_float(read_word(sac_file))); // 051
01912     baz(binary_to_float(read_word(sac_file))); // 052
01913     gcarc(binary_to_float(read_word(sac_file))); // 053
01914 }
```


Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.168 read_float_headers_meta()

```
void sacfmt::Trace::read_float_headers_meta (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 054–069.

Note that this expects the position of the reader to be the beginning of word 054.

Note that this modifies the position of the reader to the end of word 069.

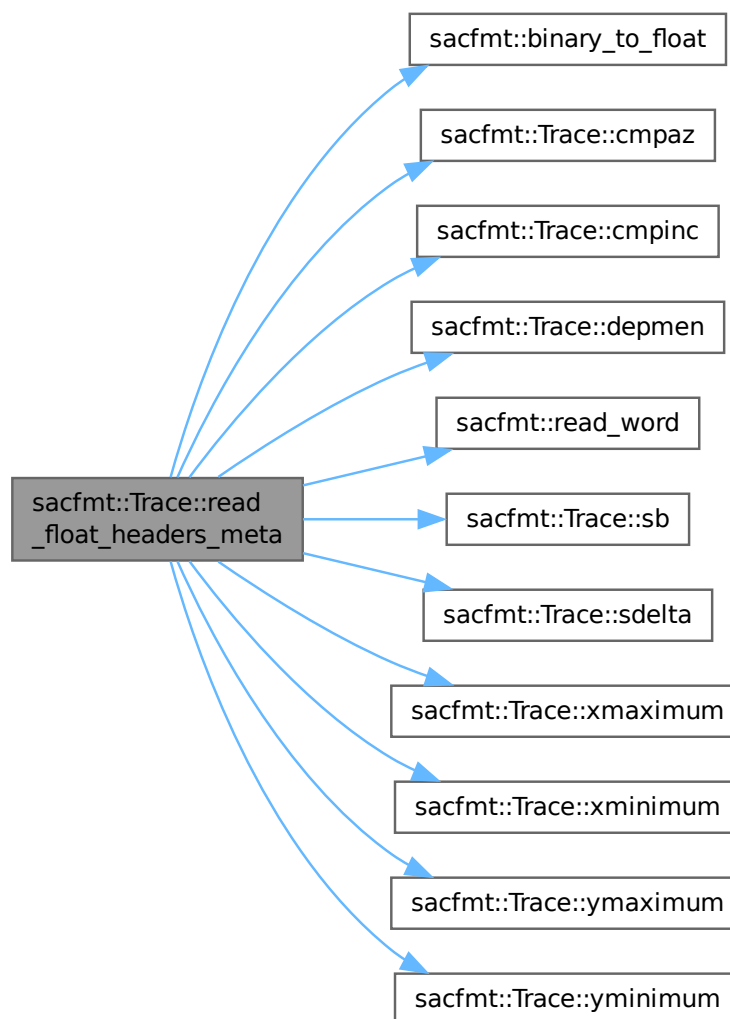
Headers loaded: sb, sdelta, depmen, cmpaz, cmpinc, xminimum, xmaximum, yminimum, and ymaximum.

Parameters

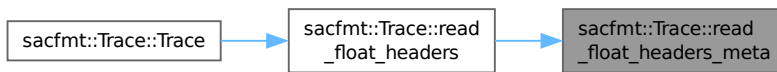
<i>in, out</i>	<i>sac_file</i>	std::ifstream* SAC-file to be read.
----------------	-----------------	-------------------------------------

```
01929 {
01930     sb(binary_to_float(read_word(sac_file))); // 054
01931     sdelta(binary_to_float(read_word(sac_file))); // 055
01932     depmen(binary_to_float(read_word(sac_file))); // 056
01933     cmpaz(binary_to_float(read_word(sac_file))); // 057
01934     cmpinc(binary_to_float(read_word(sac_file))); // 058
01935     xminimum(binary_to_float(read_word(sac_file))); // 059
01936     xmaximum(binary_to_float(read_word(sac_file))); // 060
01937     yminimum(binary_to_float(read_word(sac_file))); // 061
01938     ymaximum(binary_to_float(read_word(sac_file))); // 062
01939     // Skip 'unused' (xcommon_skip_num)
01940     for (int i{0}; i < common_skip_num; ++i) { // 063--069
01941         read_word(sac_file);
01942     }
01943 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.169 read_float_headers_resp()

```
void sacfmt::Trace::read_float_headers_resp (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 021–030.

Note that this expects the position of the reader to be the beginning of word 021.

Note that this modifies the position of the reader to the end of word 030.

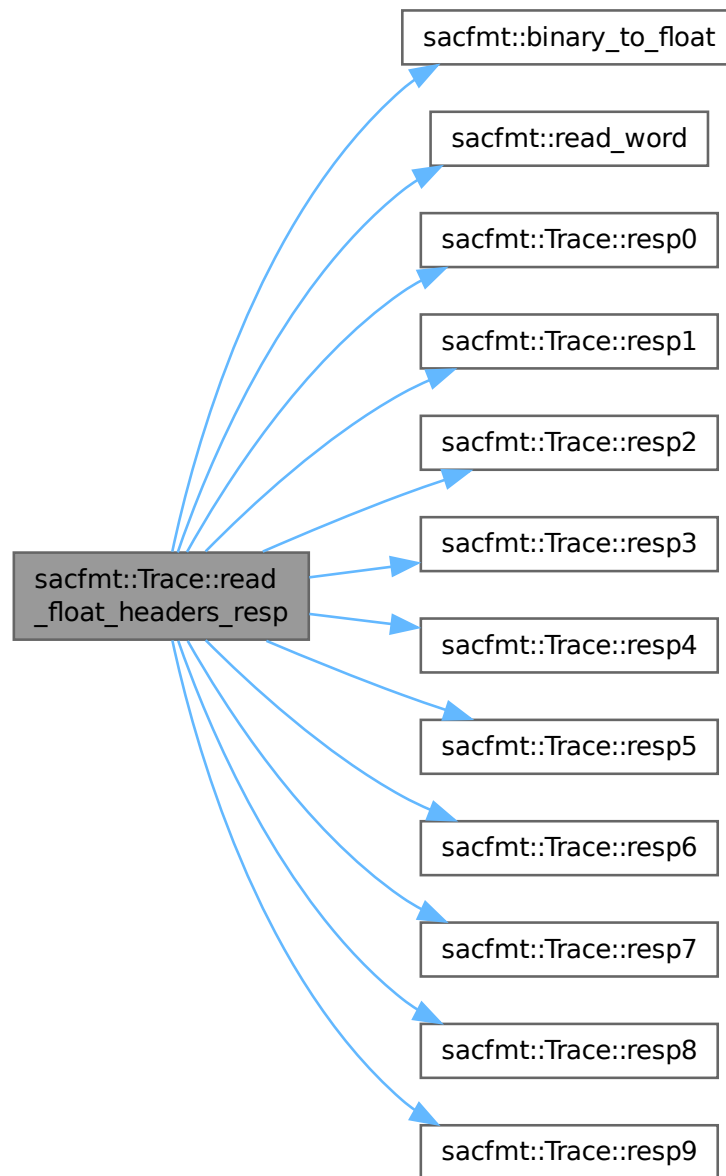
Headers loaded: resp0, resp1, resp2, resp3, resp4, resp5, resp6, resp7, resp8, and resp9.

Parameters

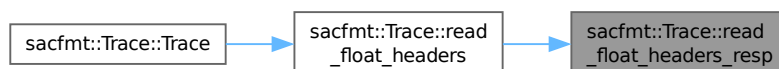
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
01832
01833     resp0(binary_to_float(read_word(sac_file))); // 021
01834     resp1(binary_to_float(read_word(sac_file))); // 022
01835     resp2(binary_to_float(read_word(sac_file))); // 023
01836     resp3(binary_to_float(read_word(sac_file))); // 024
01837     resp4(binary_to_float(read_word(sac_file))); // 025
01838     resp5(binary_to_float(read_word(sac_file))); // 026
01839     resp6(binary_to_float(read_word(sac_file))); // 027
01840     resp7(binary_to_float(read_word(sac_file))); // 028
01841     resp8(binary_to_float(read_word(sac_file))); // 029
01842     resp9(binary_to_float(read_word(sac_file))); // 030
01843 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.170 read_float_headers_starter()

```
void sacfmt::Trace::read_float_headers_starter (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 000–009.

Note that this expects the position of the reader to be the beginning of word 000.

Note that this modifies the position of the reader to the end of word 009.

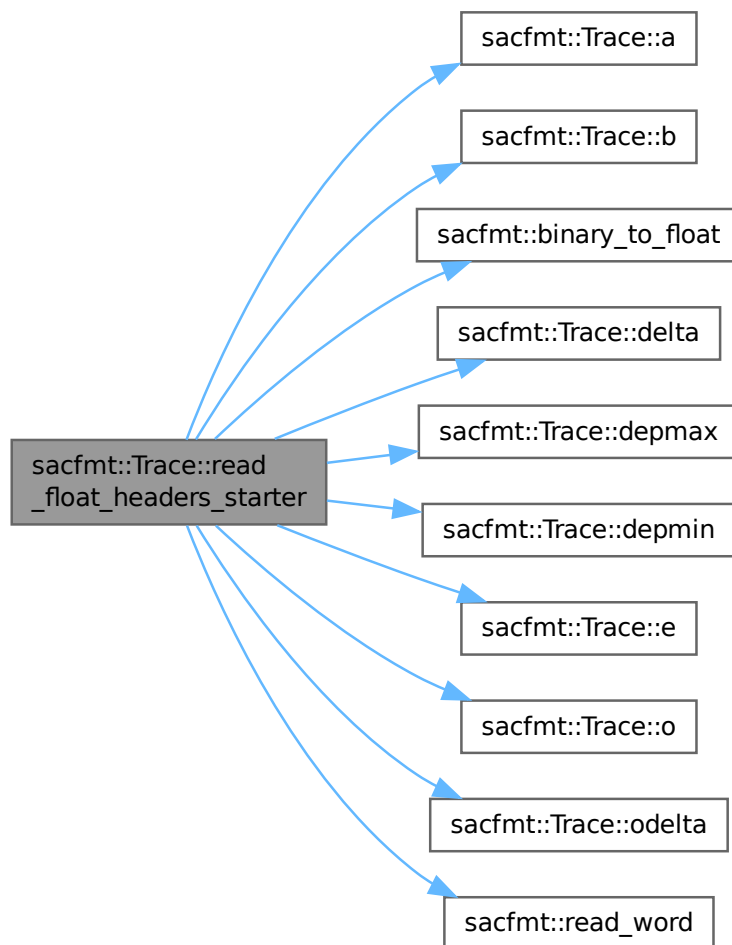
Headers loaded: delta, depmin, depmax, odelta, b, e, o, and a.

Parameters

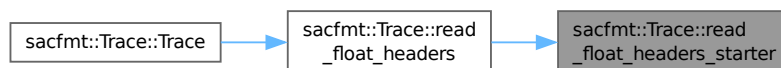
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
01778
01779     delta(binary_to_float(read_word(sac_file))); // 000
01780     depmin(binary_to_float(read_word(sac_file))); // 001
01781     depmax(binary_to_float(read_word(sac_file))); // 002
01782     // Skip 'unused'
01783     read_word(sac_file); // 003
01784     odelta(binary_to_float(read_word(sac_file))); // 004
01785     b(binary_to_float(read_word(sac_file))); // 005
01786     e(binary_to_float(read_word(sac_file))); // 006
01787     o(binary_to_float(read_word(sac_file))); // 007
01788     a(binary_to_float(read_word(sac_file))); // 008
01789     // Skip 'internal'
01790     read_word(sac_file); // 009
01791 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.171 read_float_headers_station_event()

```

void sacfmt::Trace::read_float_headers_station_event (
    std::ifstream * sac_file ) [private]
  
```

Reads SAC-headers from words 031–039.

Note that this expects the position of the reader to be the beginning of word 031.

Note that this modifies the position of the reader to the end of word 039.

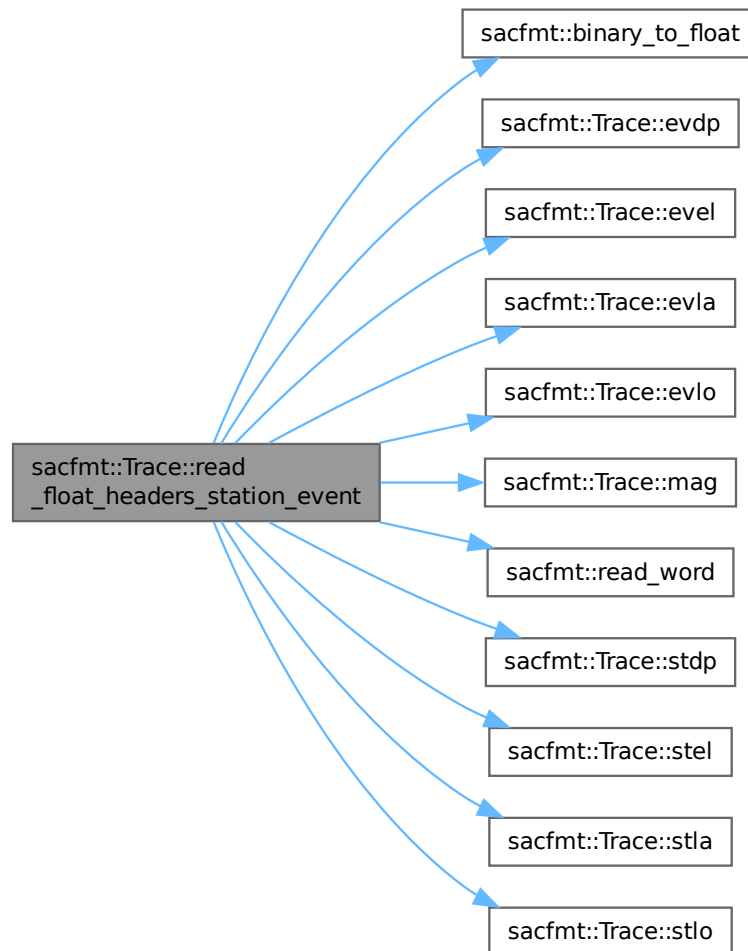
Headers loaded: stla, stlo, stel, stdp, evla, evlo, evel, evdp, and mag.

Parameters

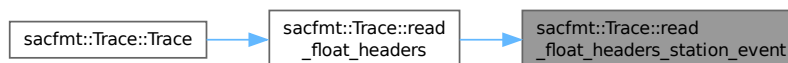
<code>in, out</code>	<code>sac_file</code>	<code>std::ifstream*</code> SAC-file to be read.
----------------------	-----------------------	--

```
01857                                     {
01858 // Station headers
01859 stla(binary_to_float(read_word(sac_file))); // 031
01860 stlo(binary_to_float(read_word(sac_file))); // 032
01861 stel(binary_to_float(read_word(sac_file))); // 033
01862 stdp(binary_to_float(read_word(sac_file))); // 034
01863 // Event headers
01864 evla(binary_to_float(read_word(sac_file))); // 035
01865 evlo(binary_to_float(read_word(sac_file))); // 036
01866 evel(binary_to_float(read_word(sac_file))); // 037
01867 evdp(binary_to_float(read_word(sac_file))); // 038
01868 mag(binary_to_float(read_word(sac_file))); // 039
01869 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.172 read_float_headers_t()

```

void sacfmt::Trace::read_float_headers_t (
    std::ifstream * sac_file ) [private]
  
```


Reads SAC-headers from words 010–020.

Note that this expects the position of the reader to be the beginning of word 010.

Note that this modifies the position of the reader to the end of word 020.

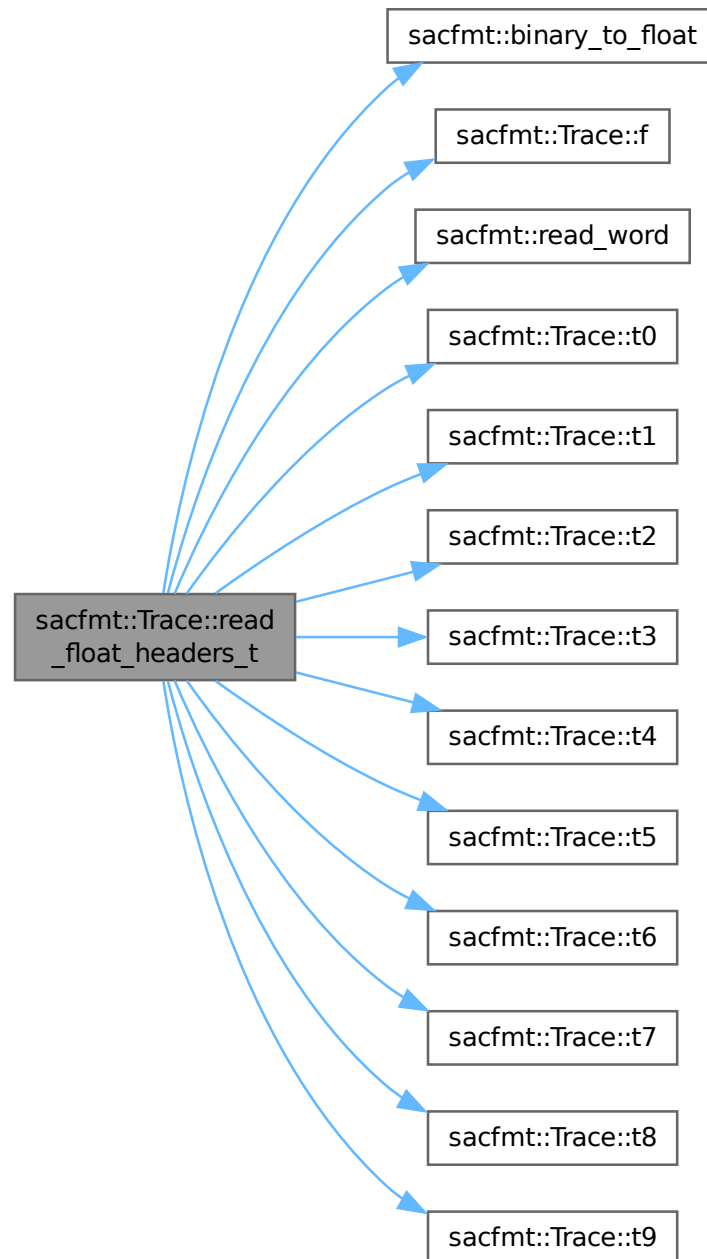
Headers loaded: t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, and f.

Parameters

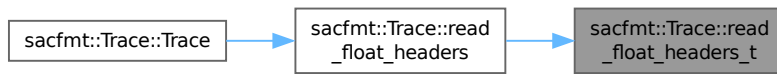
<i>in, out</i>	<i>sac_file</i>	std::ifstream* SAC-file to be read.
----------------	-----------------	-------------------------------------

```
01805
01806     t0(binary_to_float(read_word(sac_file))); // 010
01807     t1(binary_to_float(read_word(sac_file))); // 011
01808     t2(binary_to_float(read_word(sac_file))); // 012
01809     t3(binary_to_float(read_word(sac_file))); // 013
01810     t4(binary_to_float(read_word(sac_file))); // 014
01811     t5(binary_to_float(read_word(sac_file))); // 015
01812     t6(binary_to_float(read_word(sac_file))); // 016
01813     t7(binary_to_float(read_word(sac_file))); // 017
01814     t8(binary_to_float(read_word(sac_file))); // 018
01815     t9(binary_to_float(read_word(sac_file))); // 019
01816     f(binary_to_float(read_word(sac_file))); // 020
01817 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.173 read_float_headers_user()

```
void sacfmt::Trace::read_float_headers_user (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 040–049.

Note that this expects the position of the reader to be the beginning of word 040.

Note that this modifies the position of the reader to the end of word 049.

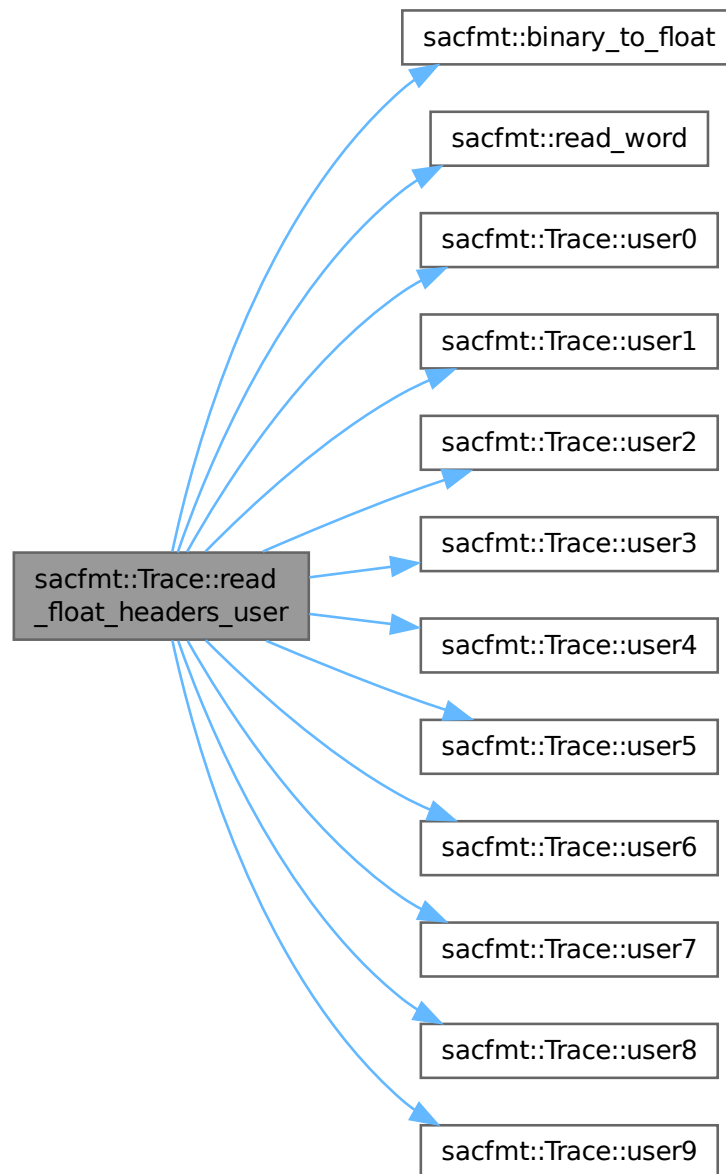
Headers loaded: user0, user1, user2, user3, user4, user5, user6, user7, user8, and user9.

Parameters

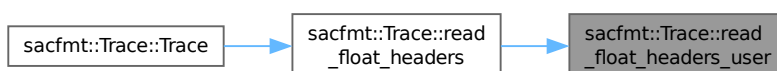
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
01884
01885     user0(binary_to_float(read_word(sac_file))); // 040
01886     user1(binary_to_float(read_word(sac_file))); // 041
01887     user2(binary_to_float(read_word(sac_file))); // 042
01888     user3(binary_to_float(read_word(sac_file))); // 043
01889     user4(binary_to_float(read_word(sac_file))); // 044
01890     user5(binary_to_float(read_word(sac_file))); // 045
01891     user6(binary_to_float(read_word(sac_file))); // 046
01892     user7(binary_to_float(read_word(sac_file))); // 047
01893     user8(binary_to_float(read_word(sac_file))); // 048
01894     user9(binary_to_float(read_word(sac_file))); // 049
01895 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.174 read_footers()

```
void sacfmt::Trace::read_footers (
    std::ifstream * sac_file ) [private]
```

Reads SAC-footers (post-data words 00–43).

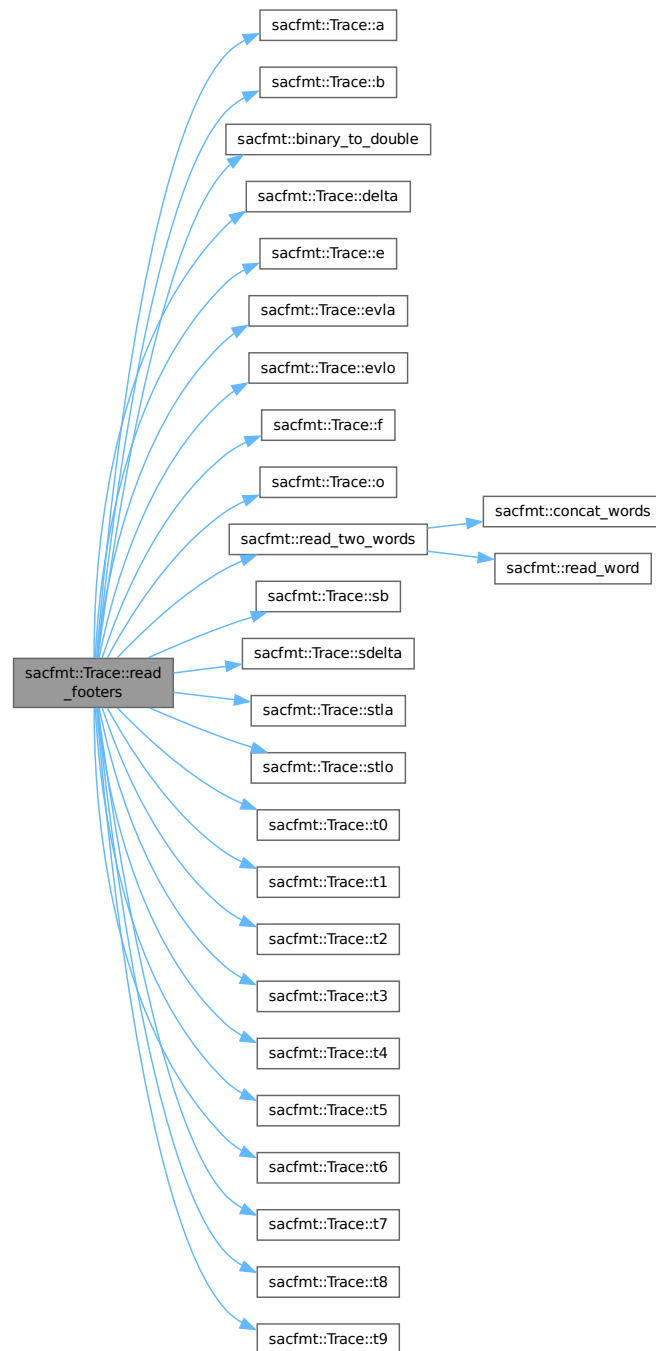
Note that this modifies the position of the reader to the end of the footer section.

Parameters

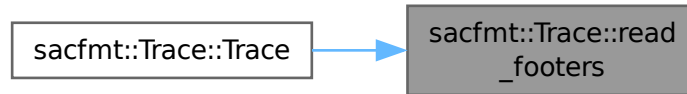
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
02153     {
02154         delta(binary_to_double(read_two_words(sac_file))); // 00-01
02155         b(binary_to_double(read_two_words(sac_file))); // 02-03
02156         e(binary_to_double(read_two_words(sac_file))); // 04-05
02157         o(binary_to_double(read_two_words(sac_file))); // 06-07
02158         a(binary_to_double(read_two_words(sac_file))); // 08-09
02159         t0(binary_to_double(read_two_words(sac_file))); // 10-11
02160         t1(binary_to_double(read_two_words(sac_file))); // 12-13
02161         t2(binary_to_double(read_two_words(sac_file))); // 14-15
02162         t3(binary_to_double(read_two_words(sac_file))); // 16-17
02163         t4(binary_to_double(read_two_words(sac_file))); // 18-19
02164         t5(binary_to_double(read_two_words(sac_file))); // 20-21
02165         t6(binary_to_double(read_two_words(sac_file))); // 22-23
02166         t7(binary_to_double(read_two_words(sac_file))); // 24-25
02167         t8(binary_to_double(read_two_words(sac_file))); // 26-27
02168         t9(binary_to_double(read_two_words(sac_file))); // 28-29
02169         f(binary_to_double(read_two_words(sac_file))); // 30-31
02170         evlo(binary_to_double(read_two_words(sac_file))); // 32-33
02171         evla(binary_to_double(read_two_words(sac_file))); // 34-35
02172         stlo(binary_to_double(read_two_words(sac_file))); // 36-37
02173         stla(binary_to_double(read_two_words(sac_file))); // 38-39
02174         sb(binary_to_double(read_two_words(sac_file))); // 40-41
02175         sdelta(binary_to_double(read_two_words(sac_file))); // 42-43
02176     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.175 read_int_headers()

```
void sacfmt::Trace::read_int_headers (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 070–104.

Note that this expects the position of the reader to be the beginning of word 070.

Note that this modifies the position of the reader to the end of word 104.

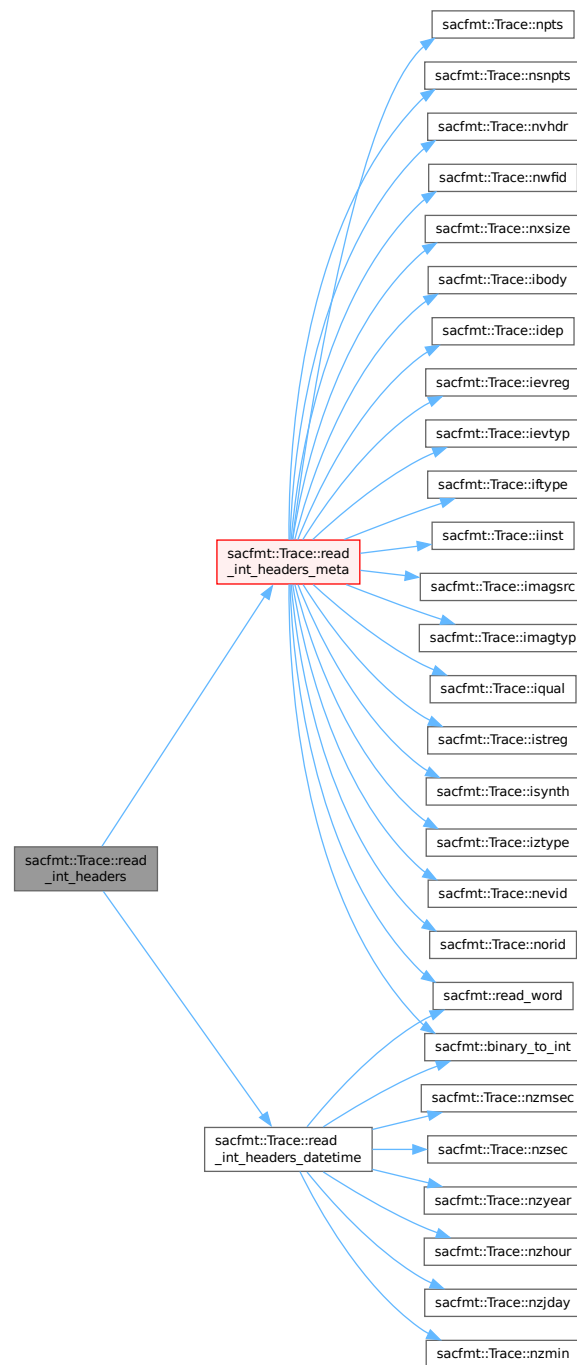
Loads all integer headers.

Parameters

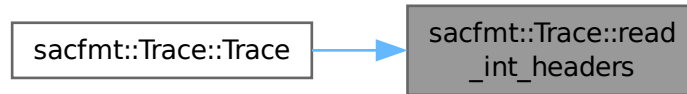
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
02045     {
02046     read_int_headers_datetime(sac_file); // 070--075
02047     read_int_headers_meta(sac_file);    // 076--104
02048 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.176 read_int_headers_datetime()

```
void sacfmt::Trace::read_int_headers_datetime (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 070–075.

Note that this expects the position of the reader to be the beginning of word 070.

Note that this modifies the position of the reader to the end of word 075.

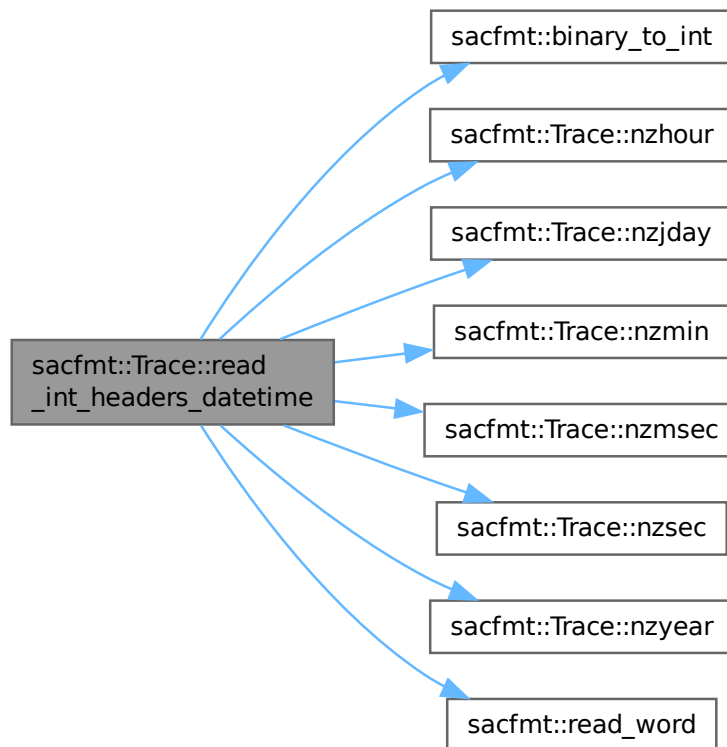
Headers loaded: nzyear, nzjday, nzhour, nzmin, nzsec, and nzmsec.

Parameters

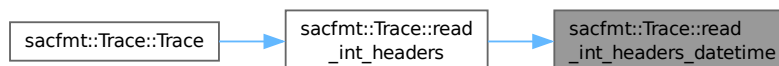
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
01979
01980  nzyear(binary_to_int(read_word(sac_file))); // 070
01981  nzjday(binary_to_int(read_word(sac_file))); // 071
01982  nzhour(binary_to_int(read_word(sac_file))); // 072
01983  nzmin(binary_to_int(read_word(sac_file))); // 073
01984  nzsec(binary_to_int(read_word(sac_file))); // 074
01985  nzmsec(binary_to_int(read_word(sac_file))); // 075
01986 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.177 read_int_headers_meta()

```
void sacfmt::Trace::read_int_headers_meta (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 076–104.

Note that this expects the position of the reader to be the beginning of word 076.

Note that this modifies the position of the reader to the end of word 104.

Headers loaded: `nvhdr`, `norid`, `nevid`, `npts`, `nsnpts`, `nwfid`, `nxsize`, `nysize`, `iftype`, `idep`, `iztype`, `iinst`, `istreg`, `ievreg`, `ievtyp`, `igual`, `isynth`, `imagtyp`, `imagsrc`, and `ibody`.

Parameters

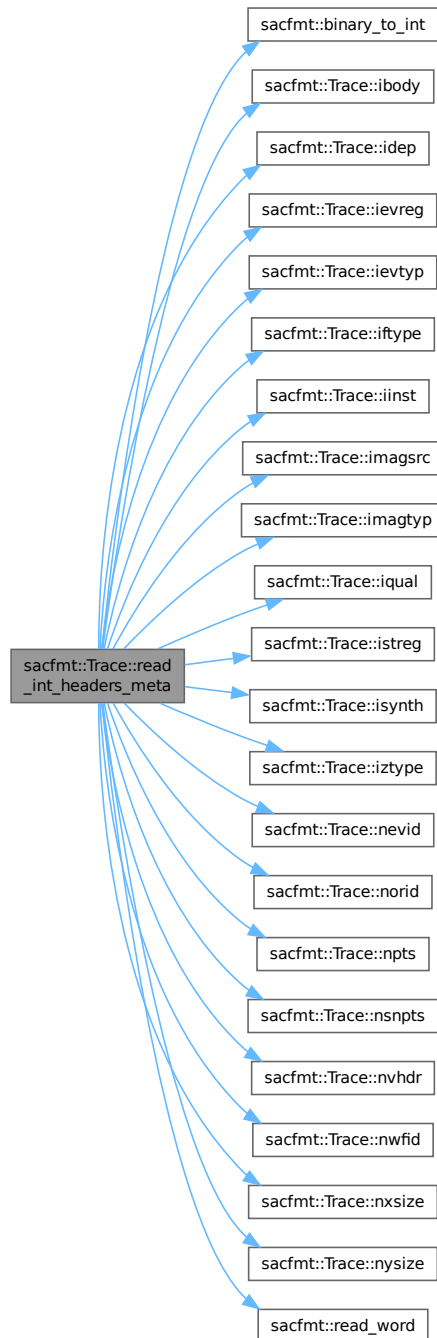
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```

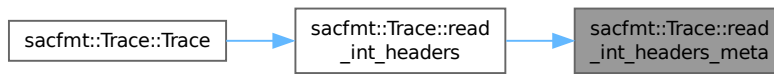
02002
02003     nvhdr(binary_to_int(read_word(sac_file))); // 076
02004     norid(binary_to_int(read_word(sac_file))); // 077
02005     nevid(binary_to_int(read_word(sac_file))); // 078
02006     npts(binary_to_int(read_word(sac_file))); // 079
02007     nsnpts(binary_to_int(read_word(sac_file))); // 080
02008     nwfid(binary_to_int(read_word(sac_file))); // 081
02009     nxsize(binary_to_int(read_word(sac_file))); // 082
02010     nysize(binary_to_int(read_word(sac_file))); // 083
02011     // Skip 'unused'
02012     read_word(sac_file); // 084
02013     iftype(binary_to_int(read_word(sac_file))); // 085
02014     idep(binary_to_int(read_word(sac_file))); // 086
02015     iztype(binary_to_int(read_word(sac_file))); // 087
02016     // Skip 'unused'
02017     read_word(sac_file); // 088
02018     iinst(binary_to_int(read_word(sac_file))); // 089
02019     istreg(binary_to_int(read_word(sac_file))); // 090
02020     ievreg(binary_to_int(read_word(sac_file))); // 091
02021     ievtyp(binary_to_int(read_word(sac_file))); // 092
02022     iqual(binary_to_int(read_word(sac_file))); // 093
02023     isynth(binary_to_int(read_word(sac_file))); // 094
02024     imagtyp(binary_to_int(read_word(sac_file))); // 095
02025     imagsrc(binary_to_int(read_word(sac_file))); // 096
02026     ibody(binary_to_int(read_word(sac_file))); // 097
02027     // Skip 'unused' (xcommon_skip_num)
02028     for (int i{0}; i < common_skip_num; ++i) { // 098--104
02029         read_word(sac_file);
02030     }
02031 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.178 read_string_headers()

```
void sacfmt::Trace::read_string_headers (
    std::ifstream * sac_file ) [private]
```

Reads SAC-headers from words 110–157.

Note that this expects the position of the reader to be the beginning of word 110.

Note that this modifies the position of the reader to the end of word 157.

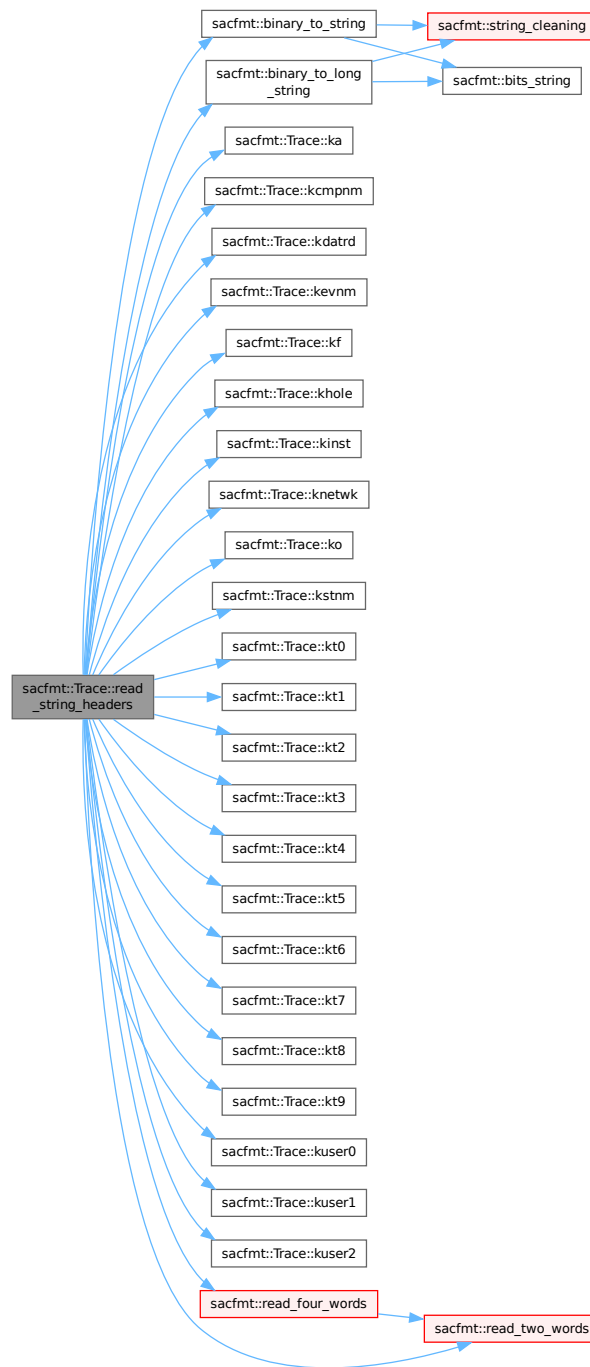
Loads all string headers.

Parameters

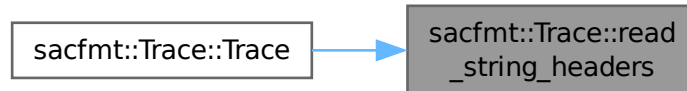
in, out	sac_file	std::ifstream* SAC-file to be read.
---------	----------	-------------------------------------

```
02084         {
02085         // KSTNM is 2 words (normal)
02086         kstnm(binary_to_string(read_two_words(sac_file))); // 110-111
02087         // KEVNM is 4 words long (unique!)
02088         kevnm(binary_to_long_string(read_four_words(sac_file))); // 112-115
02089         // All other 'K' headers are 2 words
02090         khole(binary_to_string(read_two_words(sac_file))); // 116-117
02091         ko(binary_to_string(read_two_words(sac_file))); // 118-119
02092         ka(binary_to_string(read_two_words(sac_file))); // 120-121
02093         kt0(binary_to_string(read_two_words(sac_file))); // 122-123
02094         kt1(binary_to_string(read_two_words(sac_file))); // 124-125
02095         kt2(binary_to_string(read_two_words(sac_file))); // 126-127
02096         kt3(binary_to_string(read_two_words(sac_file))); // 128-129
02097         kt4(binary_to_string(read_two_words(sac_file))); // 130-131
02098         kt5(binary_to_string(read_two_words(sac_file))); // 132-133
02099         kt6(binary_to_string(read_two_words(sac_file))); // 134-135
02100         kt7(binary_to_string(read_two_words(sac_file))); // 136-137
02101         kt8(binary_to_string(read_two_words(sac_file))); // 138-139
02102         kt9(binary_to_string(read_two_words(sac_file))); // 140-141
02103         kf(binary_to_string(read_two_words(sac_file))); // 142-143
02104         kuser0(binary_to_string(read_two_words(sac_file))); // 144-145
02105         kuser1(binary_to_string(read_two_words(sac_file))); // 146-147
02106         kuser2(binary_to_string(read_two_words(sac_file))); // 148-149
02107         kcmpnm(binary_to_string(read_two_words(sac_file))); // 150-151
02108         knetwk(binary_to_string(read_two_words(sac_file))); // 152-153
02109         kdatrd(binary_to_string(read_two_words(sac_file))); // 154-155
02110         kinst(binary_to_string(read_two_words(sac_file))); // 156-157
02111     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.179 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.

```
01654     {
01655     resize_data1(size);
01656     resize_data2(size);
01657 }
```

11.5.3.180 resize_data1()

```
void sacfmt::Trace::resize_data1 (
    size_t size ) [private], [noexcept]
01625 {
01626     if (size != data1().size()) {
01627         std::vector<double> new_data1{data1()};
01628         new_data1.resize(size, 0.0);
01629         data1(new_data1);
01630     }
01631 }
```

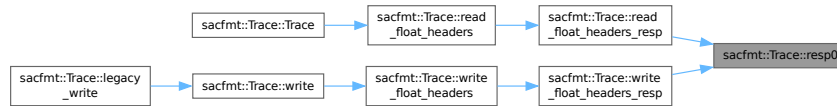
11.5.3.181 resize_data2()

```
void sacfmt::Trace::resize_data2 (
    size_t size ) [private], [noexcept]
01633 {
01634     // Data2 is legal
01635     if (!leven() || (iftype() > 1)) {
01636         if (size != data2().size()) {
01637             std::vector<double> new_data2{data2()};
01638             new_data2.resize(size, 0.0);
01639             data2(new_data2);
01640         }
01641     } else {
01642         if (!data2().empty()) {
01643             std::vector<double> new_data2{};
01644             data2(new_data2);
01645         }
01646     }
01647 }
```

11.5.3.182 resp0() [1/2]

```
float sacfmt::Trace::resp0 ( ) const [noexcept]
01036 { return floats[sac_map.at(name::resp0)]; }
```

Here is the caller graph for this function:

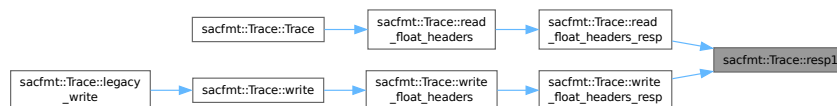
**11.5.3.183 resp0()** [2/2]

```
void sacfmt::Trace::resp0 (
    float input ) [noexcept]
01225 {
01226     floats[sac_map.at(name::resp0)] = input;
01227 }
```

11.5.3.184 resp1() [1/2]

```
float sacfmt::Trace::resp1 ( ) const [noexcept]
01037 { return floats[sac_map.at(name::resp1)]; }
```

Here is the caller graph for this function:

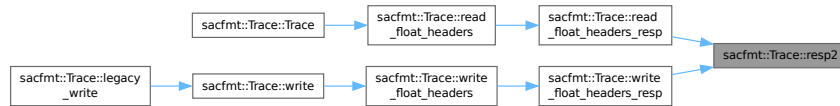
**11.5.3.185 resp1()** [2/2]

```
void sacfmt::Trace::resp1 (
    float input ) [noexcept]
01228 {
01229     floats[sac_map.at(name::resp1)] = input;
01230 }
```


11.5.3.186 resp2() [1/2]

```
float sacfmt::Trace::resp2 ( ) const [noexcept]
01038 { return floats[sac_map.at(name::resp2)]; }
```

Here is the caller graph for this function:



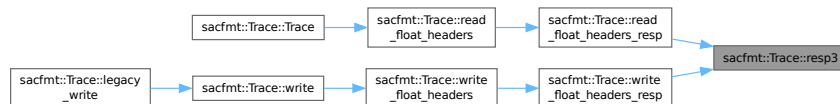
11.5.3.187 resp2() [2/2]

```
void sacfmt::Trace::resp2 (
    float input ) [noexcept]
01231 {
01232     floats[sac_map.at(name::resp2)] = input;
01233 }
```

11.5.3.188 resp3() [1/2]

```
float sacfmt::Trace::resp3 ( ) const [noexcept]
01039 { return floats[sac_map.at(name::resp3)]; }
```

Here is the caller graph for this function:



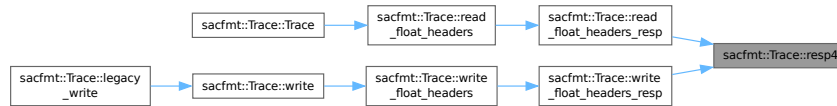
11.5.3.189 resp3() [2/2]

```
void sacfmt::Trace::resp3 (
    float input ) [noexcept]
01234 {
01235     floats[sac_map.at(name::resp3)] = input;
01236 }
```

11.5.3.190 resp4() [1/2]

```
float sacfmt::Trace::resp4 ( ) const [noexcept]
01040 { return floats[sac_map.at(name::resp4)]; }
```

Here is the caller graph for this function:

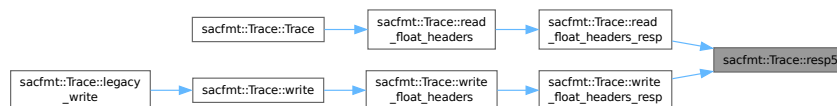
**11.5.3.191 resp4()** [2/2]

```
void sacfmt::Trace::resp4 (
    float input ) [noexcept]
01237 {
01238     floats[sac_map.at(name::resp4)] = input;
01239 }
```

11.5.3.192 resp5() [1/2]

```
float sacfmt::Trace::resp5 ( ) const [noexcept]
01041 { return floats[sac_map.at(name::resp5)]; }
```

Here is the caller graph for this function:

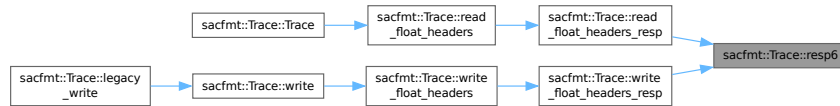
**11.5.3.193 resp5()** [2/2]

```
void sacfmt::Trace::resp5 (
    float input ) [noexcept]
01240 {
01241     floats[sac_map.at(name::resp5)] = input;
01242 }
```

11.5.3.194 resp6() [1/2]

```
float sacfmt::Trace::resp6 ( ) const [noexcept]
01042 { return floats[sac_map.at(name::resp6)]; }
```

Here is the caller graph for this function:



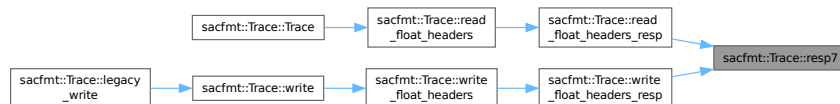
11.5.3.195 resp6() [2/2]

```
void sacfmt::Trace::resp6 (
    float input ) [noexcept]
01243 {
01244     floats[sac_map.at(name::resp6)] = input;
01245 }
```

11.5.3.196 resp7() [1/2]

```
float sacfmt::Trace::resp7 ( ) const [noexcept]
01043 { return floats[sac_map.at(name::resp7)]; }
```

Here is the caller graph for this function:



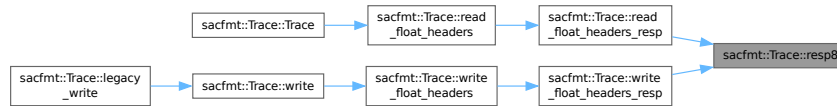
11.5.3.197 resp7() [2/2]

```
void sacfmt::Trace::resp7 (
    float input ) [noexcept]
01246 {
01247     floats[sac_map.at(name::resp7)] = input;
01248 }
```

11.5.3.198 resp8() [1/2]

```
float sacfmt::Trace::resp8 ( ) const [noexcept]
01044 { return floats[sac_map.at(name::resp8)]; }
```

Here is the caller graph for this function:

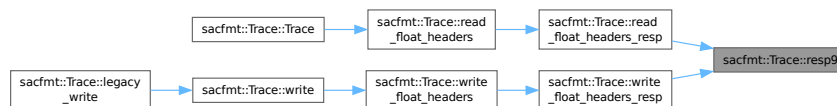
**11.5.3.199 resp8()** [2/2]

```
void sacfmt::Trace::resp8 (
    float input ) [noexcept]
01249 {
01250     floats[sac_map.at(name::resp8)] = input;
01251 }
```

11.5.3.200 resp9() [1/2]

```
float sacfmt::Trace::resp9 ( ) const [noexcept]
01045 { return floats[sac_map.at(name::resp9)]; }
```

Here is the caller graph for this function:

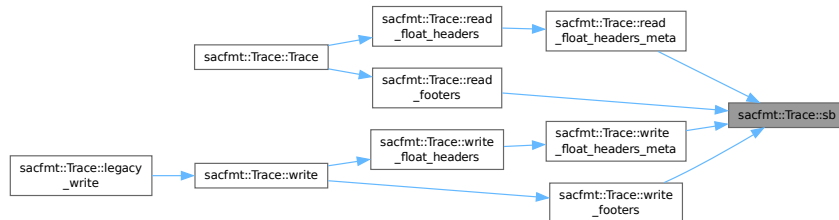
**11.5.3.201 resp9()** [2/2]

```
void sacfmt::Trace::resp9 (
    float input ) [noexcept]
01252 {
01253     floats[sac_map.at(name::resp9)] = input;
01254 }
```

11.5.3.202 sb() [1/2]

```
double sacfmt::Trace::sb ( ) const [noexcept]
01107 { return doubles[sac_map.at(name::sb)]; }
```

Here is the caller graph for this function:



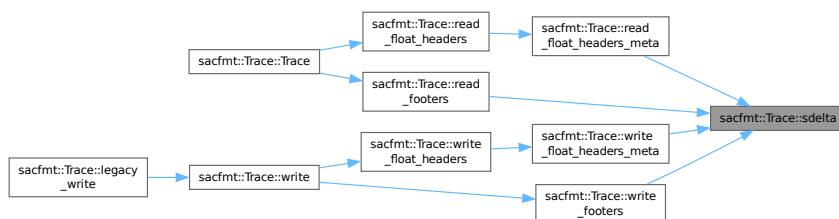
11.5.3.203 sb() [2/2]

```
void sacfmt::Trace::sb (
    double input ) [noexcept]
01410     {
01411     doubles[sac_map.at(name::sb)] = input;
01412 }
```

11.5.3.204 sdelta() [1/2]

```
double sacfmt::Trace::sdelta ( ) const [noexcept]
01108     {
01109     return doubles[sac_map.at(name::sdelta)];
01110 }
```

Here is the caller graph for this function:



11.5.3.205 sdelta() [2/2]

```
void sacfmt::Trace::sdelta (
    double input ) [noexcept]
01413     {
01414     doubles[sac_map.at(name::sdelta)] = input;
01415 }
```

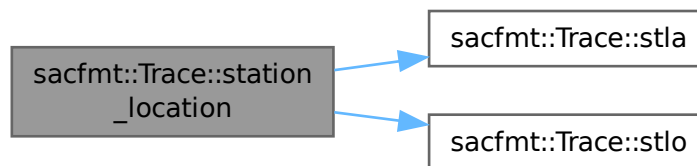
11.5.3.206 station_location()

```
point sacfmt::Trace::station_location ( ) const [inline], [private], [noexcept]
```

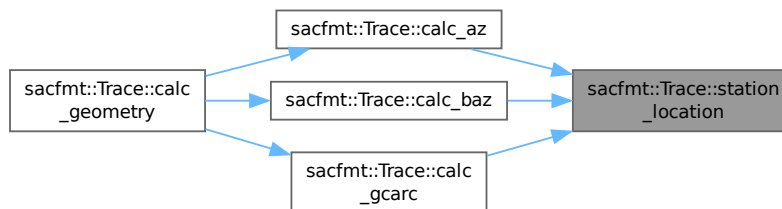
Return station location as a point.

```
01388 {
01389     return point{coord{stla()}, true}, coord{stlo()}, true};
01390 }
```

Here is the call graph for this function:



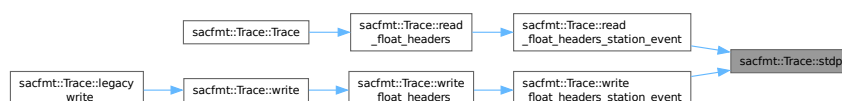
Here is the caller graph for this function:



11.5.3.207 stdp() [1/2]

```
float sacfmt::Trace::stdp ( ) const [noexcept]
01047 { return floats[sac_map.at(name::stdp)]; }
```

Here is the caller graph for this function:



11.5.3.208 stdp() [2/2]

```

void sacfmt::Trace::stdp (
    float input ) [noexcept]
01258     {
01259     floats[sac_map.at(name::stdp)] = input;
01260     }

```

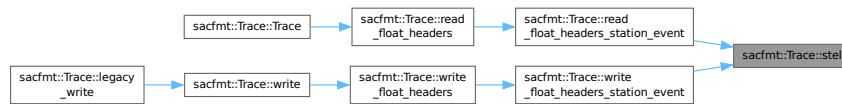
11.5.3.209 stel() [1/2]

```

float sacfmt::Trace::stel ( ) const [noexcept]
01046 { return floats[sac_map.at(name::stel)]; }

```

Here is the caller graph for this function:



11.5.3.210 stel() [2/2]

```

void sacfmt::Trace::stel (
    float input ) [noexcept]
01255     {
01256     floats[sac_map.at(name::stel)] = input;
01257     }

```

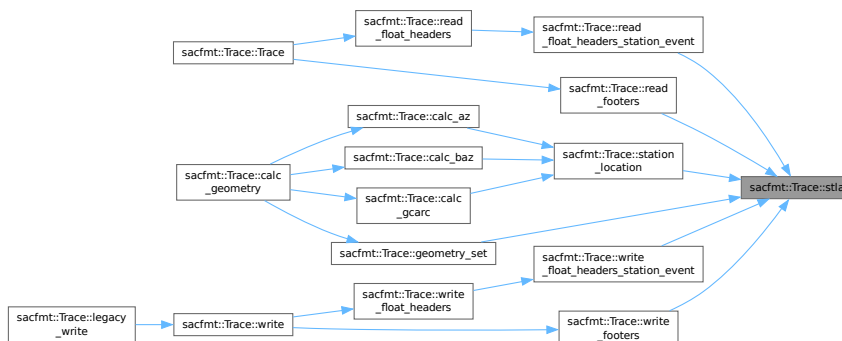
11.5.3.211 stla() [1/2]

```

double sacfmt::Trace::stla ( ) const [noexcept]
01103 { return doubles[sac_map.at(name::stla)]; }

```

Here is the caller graph for this function:



11.5.3.212 stla() [2/2]

```

void sacfmt::Trace::stla (
    double input ) [noexcept]
01382
01383     double clean_input{input};
01384     if (clean_input != unset_double) {
01385         clean_input = limit_90(clean_input);
01386     }
01387     doubles[sac_map.at(name::stla)] = clean_input;
01388 }

```

Here is the call graph for this function:



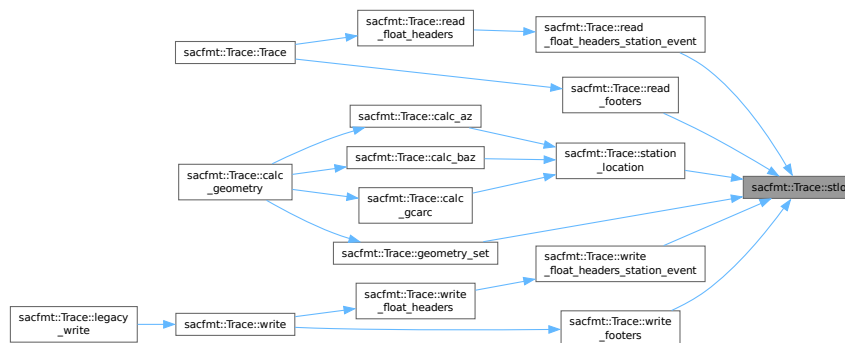
11.5.3.213 stlo() [1/2]

```

double sacfmt::Trace::stlo ( ) const [noexcept]
01104 { return doubles[sac_map.at(name::stlo)]; }

```

Here is the caller graph for this function:



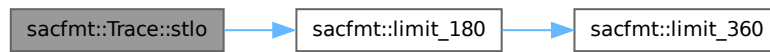
11.5.3.214 stlo() [2/2]

```

void sacfmt::Trace::stlo (
    double input ) [noexcept]
01389
01390     double clean_input{input};
01391     if (clean_input != unset_double) {
01392         clean_input = limit_180(clean_input);
01393     }
01394     doubles[sac_map.at(name::stlo)] = clean_input;
01395 }

```


Here is the call graph for this function:

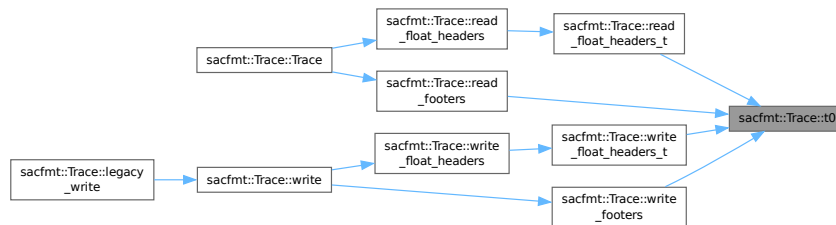


11.5.3.215 t0() [1/2]

```

double sacfmt::Trace::t0 ( ) const [noexcept]
01092 { return doubles[sac_map.at(name::t0)]; }
  
```

Here is the caller graph for this function:



11.5.3.216 t0() [2/2]

```

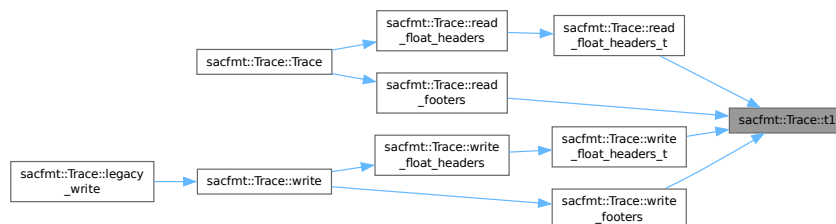
void sacfmt::Trace::t0 (
    double input ) [noexcept]
01349     {
01350     doubles[sac_map.at(name::t0)] = input;
01351     }
  
```

11.5.3.217 t1() [1/2]

```

double sacfmt::Trace::t1 ( ) const [noexcept]
01093 { return doubles[sac_map.at(name::t1)]; }
  
```

Here is the caller graph for this function:



11.5.3.218 t1() [2/2]

```

void sacfmt::Trace::t1 (
    double input ) [noexcept]
01352     {
01353     doubles[sac_map.at(name::t1)] = input;
01354     }

```

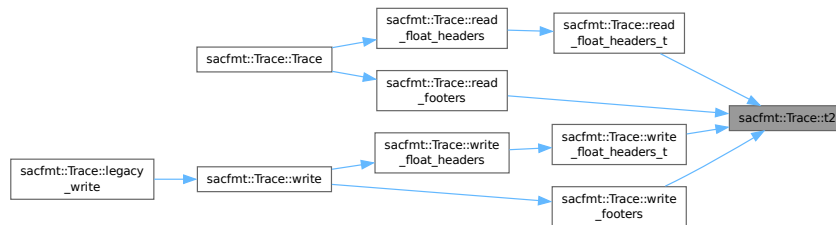
11.5.3.219 t2() [1/2]

```

double sacfmt::Trace::t2 ( ) const [noexcept]
01094 { return doubles[sac_map.at(name::t2)]; }

```

Here is the caller graph for this function:

**11.5.3.220 t2()** [2/2]

```

void sacfmt::Trace::t2 (
    double input ) [noexcept]
01355     {
01356     doubles[sac_map.at(name::t2)] = input;
01357     }

```

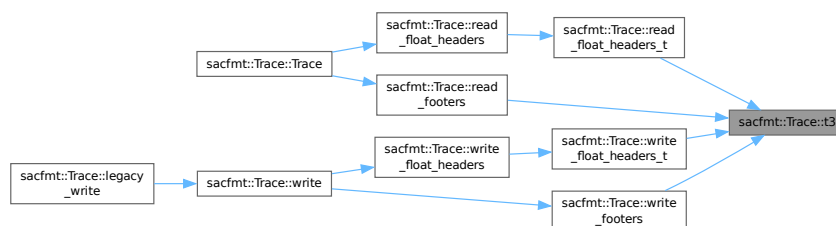
11.5.3.221 t3() [1/2]

```

double sacfmt::Trace::t3 ( ) const [noexcept]
01095 { return doubles[sac_map.at(name::t3)]; }

```

Here is the caller graph for this function:



11.5.3.222 t3() [2/2]

```

void sacfmt::Trace::t3 (
    double input ) [noexcept]
01358     {
01359     doubles[sac_map.at(name::t3)] = input;
01360     }

```

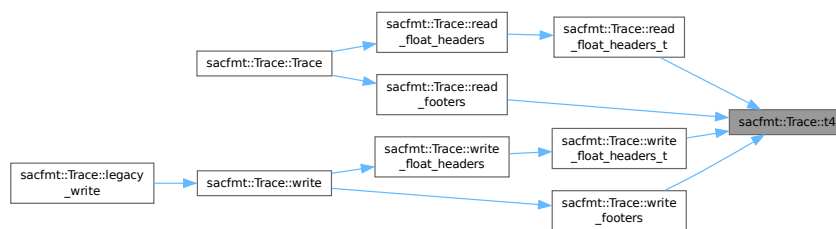
11.5.3.223 t4() [1/2]

```

double sacfmt::Trace::t4 ( ) const [noexcept]
01096 { return doubles[sac_map.at(name::t4)]; }

```

Here is the caller graph for this function:



11.5.3.224 t4() [2/2]

```

void sacfmt::Trace::t4 (
    double input ) [noexcept]
01361     {
01362     doubles[sac_map.at(name::t4)] = input;
01363     }

```

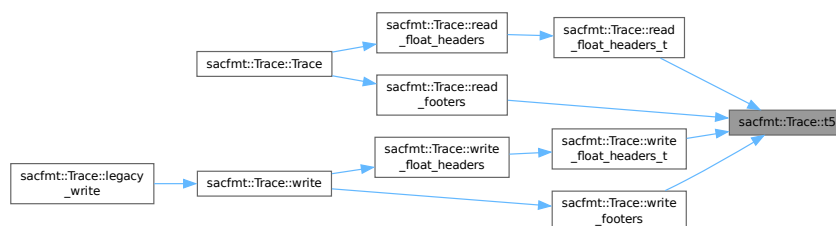
11.5.3.225 t5() [1/2]

```

double sacfmt::Trace::t5 ( ) const [noexcept]
01097 { return doubles[sac_map.at(name::t5)]; }

```

Here is the caller graph for this function:



11.5.3.226 t5() [2/2]

```

void sacfmt::Trace::t5 (
    double input ) [noexcept]
01364     {
01365     doubles[sac_map.at(name::t5)] = input;
01366     }

```

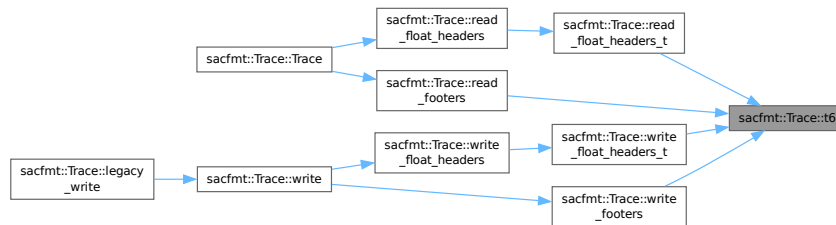
11.5.3.227 t6() [1/2]

```

double sacfmt::Trace::t6 ( ) const [noexcept]
01098 { return doubles[sac_map.at(name::t6)]; }

```

Here is the caller graph for this function:

**11.5.3.228 t6()** [2/2]

```

void sacfmt::Trace::t6 (
    double input ) [noexcept]
01367     {
01368     doubles[sac_map.at(name::t6)] = input;
01369     }

```

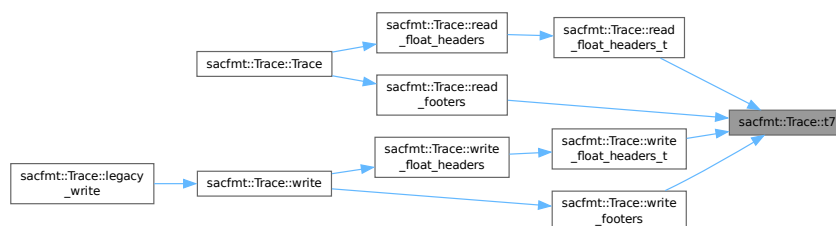
11.5.3.229 t7() [1/2]

```

double sacfmt::Trace::t7 ( ) const [noexcept]
01099 { return doubles[sac_map.at(name::t7)]; }

```

Here is the caller graph for this function:



11.5.3.230 t7() [2/2]

```

void sacfmt::Trace::t7 (
    double input ) [noexcept]
01370     {
01371     doubles[sac_map.at(name::t7)] = input;
01372     }

```

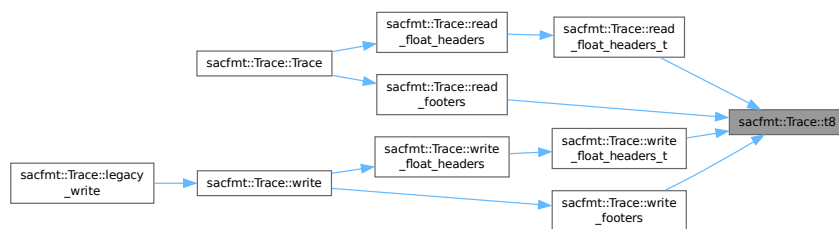
11.5.3.231 t8() [1/2]

```

double sacfmt::Trace::t8 ( ) const [noexcept]
01100 { return doubles[sac_map.at(name::t8)]; }

```

Here is the caller graph for this function:



11.5.3.232 t8() [2/2]

```

void sacfmt::Trace::t8 (
    double input ) [noexcept]
01373     {
01374     doubles[sac_map.at(name::t8)] = input;
01375     }

```

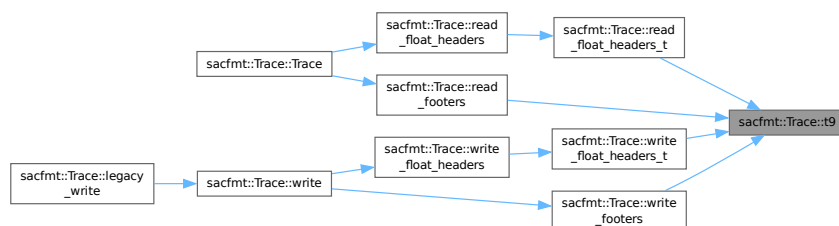
11.5.3.233 t9() [1/2]

```

double sacfmt::Trace::t9 ( ) const [noexcept]
01101 { return doubles[sac_map.at(name::t9)]; }

```

Here is the caller graph for this function:



11.5.3.234 t9() [2/2]

```

void sacfmt::Trace::t9 (
    double input ) [noexcept]
01376
01377     doubles[sac_map.at(name::t9)] = input;
01378 }

```

11.5.3.235 time()

```

std::string sacfmt::Trace::time ( ) const [noexcept]

```

Get time string.

Returns

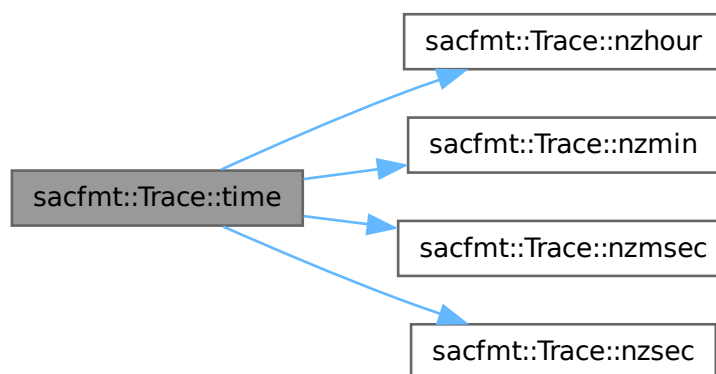
sstd::string Time (HH::MM:SS.sss).

```

01008
01009     // Require all to be set
01010     if ((nzhour() == unset_int) || (nzmin() == unset_int) ||
01011         (nzsec() == unset_int) || (nzmsec() == unset_int)) {
01012         return unset_word;
01013     }
01014     std::ostringstream oss{};
01015     oss << nzhour();
01016     oss << ':';
01017     oss << nzmin();
01018     oss << ':';
01019     oss << nzsec();
01020     oss << '.';
01021     oss << nzmsec();
01022     return oss.str();
01023 }

```

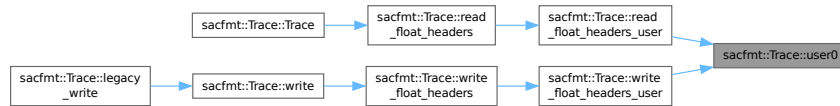
Here is the call graph for this function:



11.5.3.236 user0() [1/2]

```
float sacfmt::Trace::user0 ( ) const [noexcept]
01051 { return floats[sac_map.at(name::user0)]; }
```

Here is the caller graph for this function:



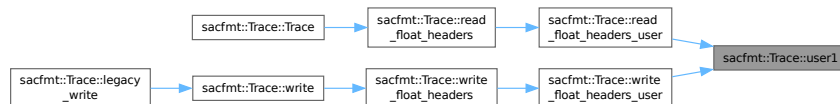
11.5.3.237 user0() [2/2]

```
void sacfmt::Trace::user0 (
    float input ) [noexcept]
01270 {
01271     floats[sac_map.at(name::user0)] = input;
01272 }
```

11.5.3.238 user1() [1/2]

```
float sacfmt::Trace::user1 ( ) const [noexcept]
01052 { return floats[sac_map.at(name::user1)]; }
```

Here is the caller graph for this function:



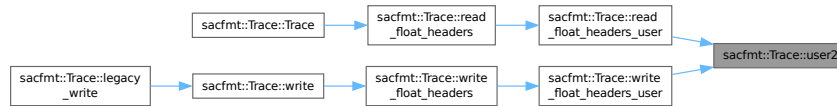
11.5.3.239 user1() [2/2]

```
void sacfmt::Trace::user1 (
    float input ) [noexcept]
01273 {
01274     floats[sac_map.at(name::user1)] = input;
01275 }
```

11.5.3.240 user2() [1/2]

```
float sacfmt::Trace::user2 ( ) const [noexcept]
01053 { return floats[sac_map.at(name::user2)]; }
```

Here is the caller graph for this function:



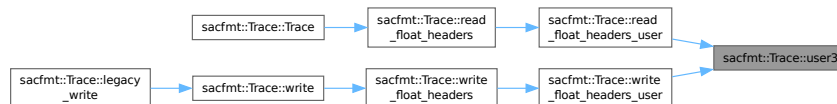
11.5.3.241 user2() [2/2]

```
void sacfmt::Trace::user2 (
    float input ) [noexcept]
01276 {
01277     floats[sac_map.at(name::user2)] = input;
01278 }
```

11.5.3.242 user3() [1/2]

```
float sacfmt::Trace::user3 ( ) const [noexcept]
01054 { return floats[sac_map.at(name::user3)]; }
```

Here is the caller graph for this function:



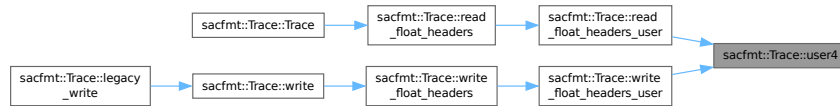
11.5.3.243 user3() [2/2]

```
void sacfmt::Trace::user3 (
    float input ) [noexcept]
01279 {
01280     floats[sac_map.at(name::user3)] = input;
01281 }
```


11.5.3.244 user4() [1/2]

```
float sacfmt::Trace::user4 ( ) const [noexcept]
01055 { return floats[sac_map.at(name::user4)]; }
```

Here is the caller graph for this function:



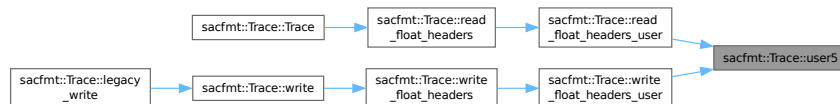
11.5.3.245 user4() [2/2]

```
void sacfmt::Trace::user4 (
    float input ) [noexcept]
01282 {
01283     floats[sac_map.at(name::user4)] = input;
01284 }
```

11.5.3.246 user5() [1/2]

```
float sacfmt::Trace::user5 ( ) const [noexcept]
01056 { return floats[sac_map.at(name::user5)]; }
```

Here is the caller graph for this function:



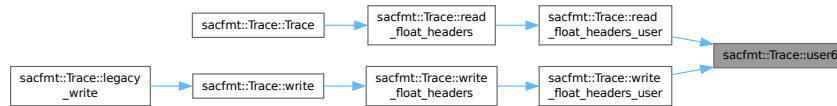
11.5.3.247 user5() [2/2]

```
void sacfmt::Trace::user5 (
    float input ) [noexcept]
01285 {
01286     floats[sac_map.at(name::user5)] = input;
01287 }
```

11.5.3.248 user6() [1/2]

```
float sacfmt::Trace::user6 ( ) const [noexcept]
01057 { return floats[sac_map.at(name::user6)]; }
```

Here is the caller graph for this function:



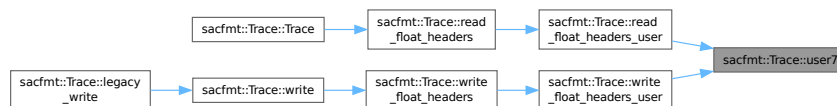
11.5.3.249 user6() [2/2]

```
void sacfmt::Trace::user6 (
    float input ) [noexcept]
01288 {
01289     floats[sac_map.at(name::user6)] = input;
01290 }
```

11.5.3.250 user7() [1/2]

```
float sacfmt::Trace::user7 ( ) const [noexcept]
01058 { return floats[sac_map.at(name::user7)]; }
```

Here is the caller graph for this function:



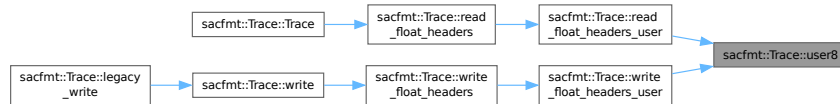
11.5.3.251 user7() [2/2]

```
void sacfmt::Trace::user7 (
    float input ) [noexcept]
01291 {
01292     floats[sac_map.at(name::user7)] = input;
01293 }
```

11.5.3.252 user8() [1/2]

```
float sacfmt::Trace::user8 ( ) const [noexcept]
01059 { return floats[sac_map.at(name::user8)]; }
```

Here is the caller graph for this function:



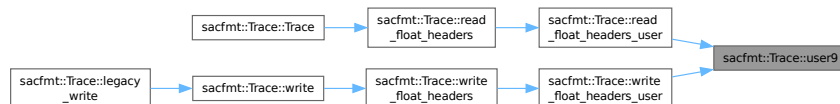
11.5.3.253 user8() [2/2]

```
void sacfmt::Trace::user8 (
    float input ) [noexcept]
01294 {
01295     floats[sac_map.at(name::user8)] = input;
01296 }
```

11.5.3.254 user9() [1/2]

```
float sacfmt::Trace::user9 ( ) const [noexcept]
01060 { return floats[sac_map.at(name::user9)]; }
```

Here is the caller graph for this function:



11.5.3.255 user9() [2/2]

```
void sacfmt::Trace::user9 (
    float input ) [noexcept]
01297 {
01298     floats[sac_map.at(name::user9)] = input;
01299 }
```

11.5.3.256 write()

```
void sacfmt::Trace::write (
    const std::filesystem::path & path,
    bool legacy = false ) const
```

Binary SAC-file writer.

Parameters

in	<i>path</i>	std::filesystem::path SAC-file to write.
in	<i>legacy</i>	bool Legacy-write flag (default false = v7, true = v6).

Exceptions

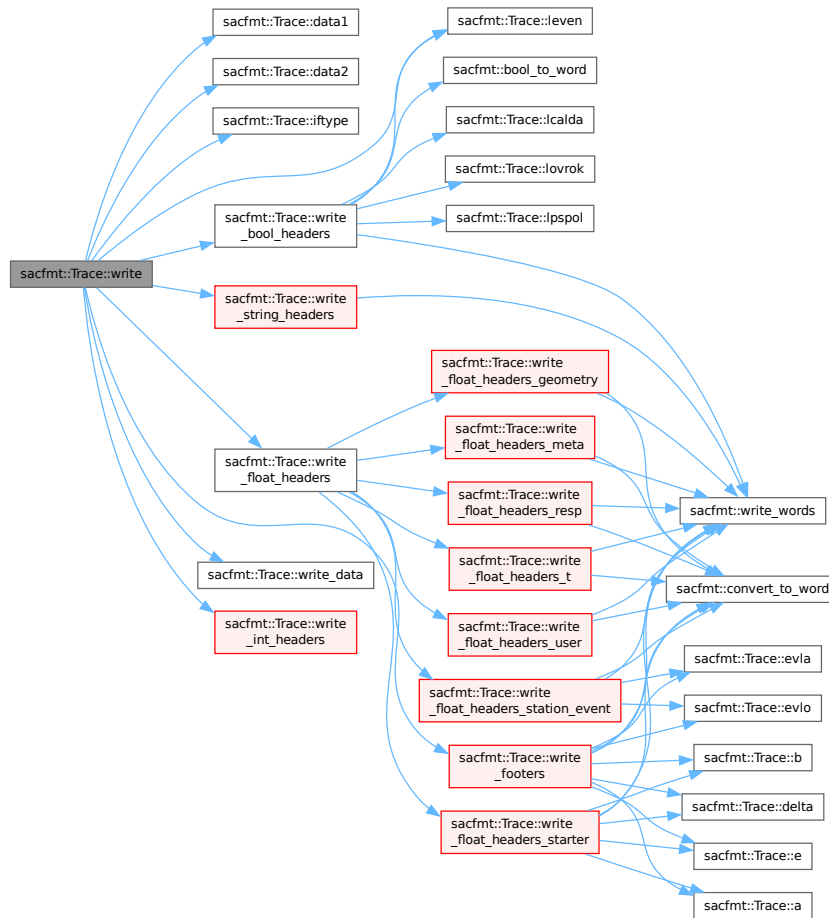
<i>io_error</i>	If the file cannot be written (bad path or bad permissions).
<i>std::exception</i>	Other unwritable issues (not enough space, disk failure, etc.).

```

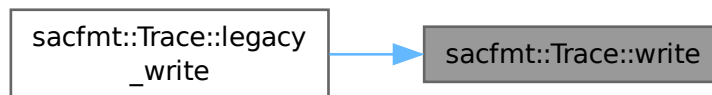
02686
02687     std::ofstream file(path, std::ios::binary | std::ios::out | std::ios::trunc);
02688     if (!file) {
02689         throw io_error(path.string() + " cannot be opened to write.");
02690     }
02691     const int header_version{legacy ? old_hdr_version : modern_hdr_version};
02692     write_float_headers(&file);
02693     write_int_headers(&file, header_version);
02694     write_bool_headers(&file);
02695     write_string_headers(&file);
02696     // Data
02697     std::vector<double> tmp{data1()};
02698     write_data(&file, tmp);
02699     if (!leven() || (iftype() > 1)) {
02700         tmp = data2();
02701         write_data(&file, tmp);
02702     }
02703     if (header_version == modern_hdr_version) {
02704         // Write footer
02705         write_footers(&file);
02706     }
02707     file.close();
02708 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.257 write_bool_headers()

```
void sacfmt::Trace::write_bool_headers (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 105–109.

Note that this expects the position of the writer to be the beginning of word 105.

Note that this modifies the position of the writer to the end of word 109.

Writes all boolean headers.

Parameters

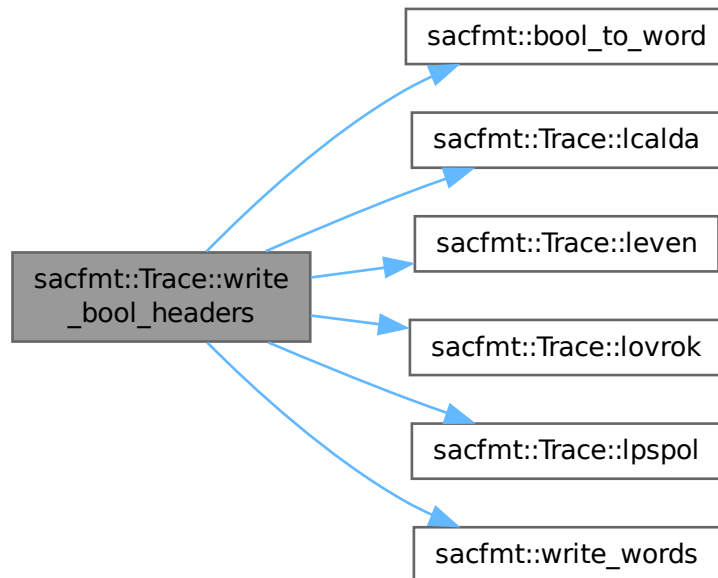
<code>in, out</code>	<code>sac_file</code>	<code>std::ofstream*</code> SAC-file to be written.
----------------------	-----------------------	---

```

02526
02527 write_words(sac_file, bool_to_word(leven())); // 105
02528 write_words(sac_file, bool_to_word(lpspol())); // 106
02529 write_words(sac_file, bool_to_word(lovrok())); // 107
02530 write_words(sac_file, bool_to_word(lcalda())); // 108
02531 // Fill 'unused'
02532 write_words(sac_file, bool_to_word(lcalda())); // 109
02533 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.258 write_data()

```
void sacfmt::Trace::write_data (
    std::ofstream * sac_file,
    const std::vector< double > & data_vec ) [static]
```

Writes data vectors.

Note that this modifies the position of the writer to the end of the data section written.

For data1 writes words 158–(158 + npts).

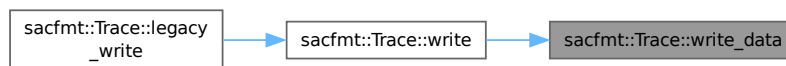
For data2 writes words (158 + 1 + npts)–(159 + (2 * npts))

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
in	<i>data_vec</i>	std::vector<double> Data-vector to write.

```
02221                                     {
02222     std::for_each(
02223         data_vec.begin(), data_vec.end(), [&sac_file](const auto &value) {
02224             write_words(sac_file, convert_to_word(static_cast<float>(value)));
02225         });
02226 }
```

Here is the caller graph for this function:



11.5.3.259 write_float_headers()

```
void sacfmt::Trace::write_float_headers (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 000–069.

Note that this expects the position of the writer to be the beginning of word 000.

Note that this modifies the position of the writer to the end of word 069.

Writes all the float headers.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

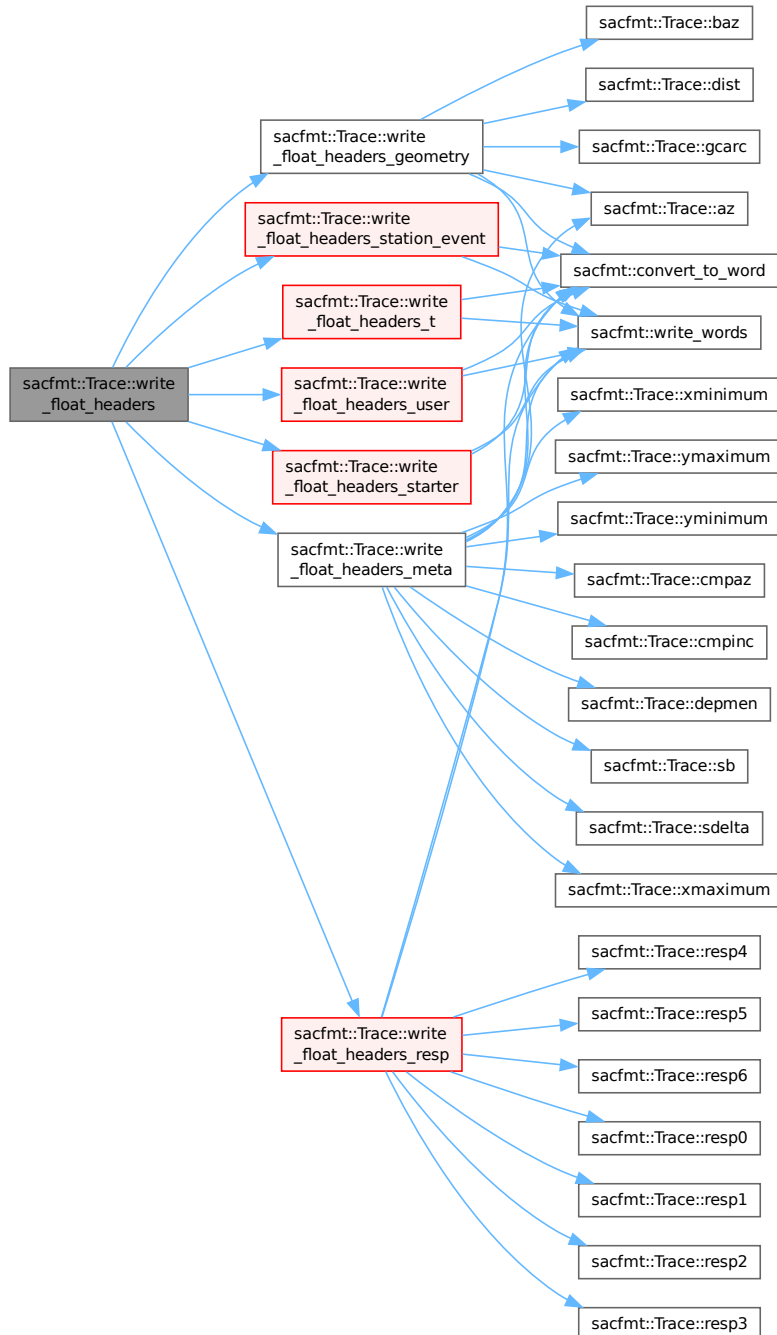
```
02417                                     {
02418     write_float_headers_starter(sac_file);           // 000-009
02419     write_float_headers_t(sac_file);               // 010-020
02420     write_float_headers_resp(sac_file);           // 031-030
02421 }
```

```

02421 write_float_headers_station_event(sac_file); // 031-039
02422 write_float_headers_user(sac_file); // 040-049
02423 write_float_headers_geometry(sac_file); // 050-053
02424 write_float_headers_meta(sac_file); // 054-069
02425 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.260 write_float_headers_geometry()

```
void sacfmt::Trace::write_float_headers_geometry (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 050–053.

Note that this expects the position of the writer to be the beginning of word 050.

Note that this modifies the position of the writer to the end of word 053.

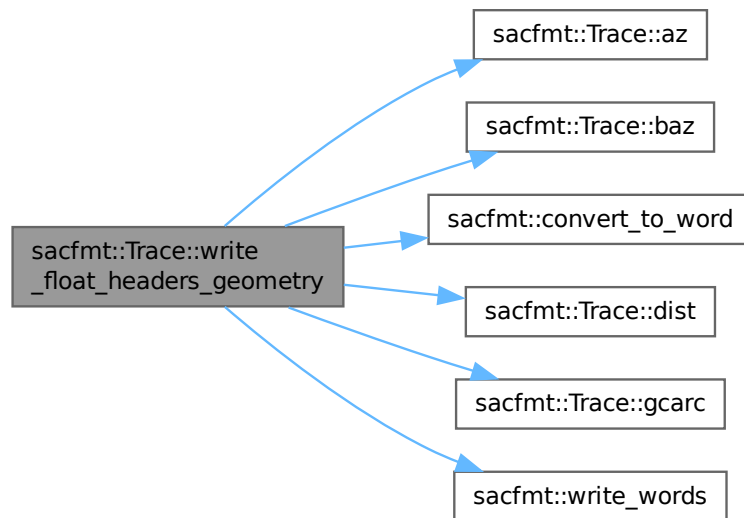
Headers written: dist, az, baz, and gcarc.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02369
02370 write_words(sac_file, convert_to_word(dist())); // 050
02371 write_words(sac_file, convert_to_word(az())); // 051
02372 write_words(sac_file, convert_to_word(baz())); // 052
02373 write_words(sac_file, convert_to_word(gcarc())); // 053
02374 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.261 write_float_headers_meta()

```
void sacfmt::Trace::write_float_headers_meta (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 054–069.

Note that this expects the position of the writer to be the beginning of word 054.

Note that this modifies the position of the writer to the end of word 069.

Headers written: sb, sdelta, depmen, cmpaz, cmpinc, xminimum, xmaximum, yminimum, and ymaximum.

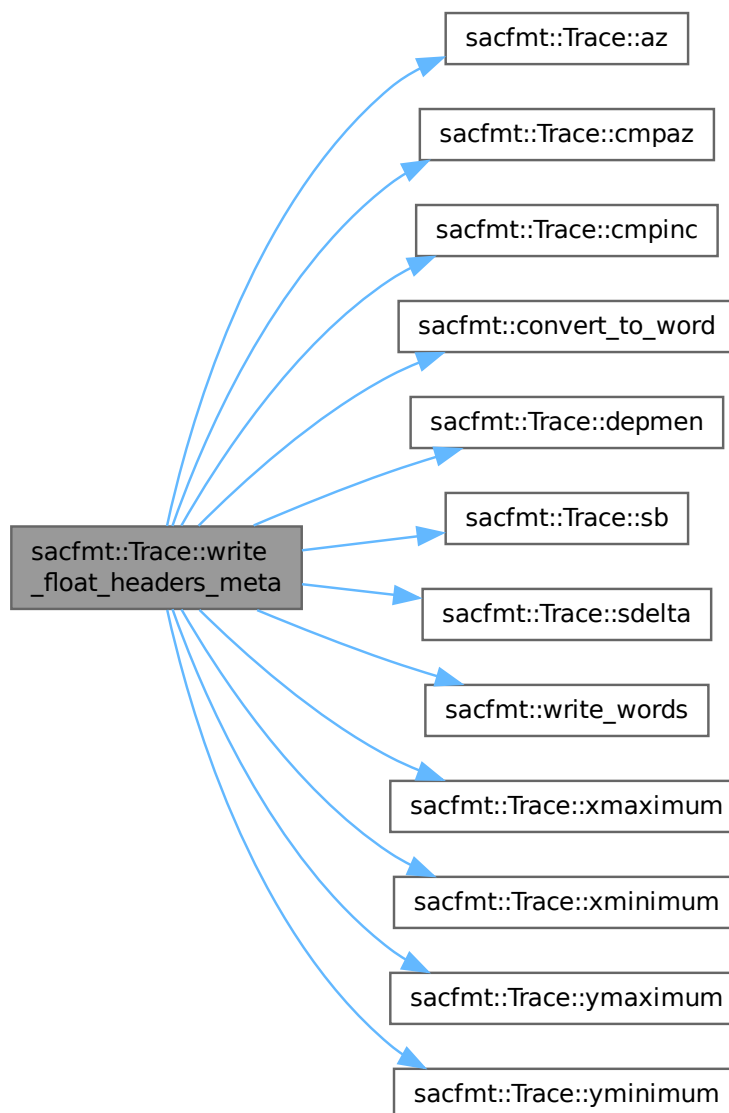
Parameters

<code>in, out</code>	<code>sac_file</code>	<code>std::ofstream*</code> SAC-file to be written.
----------------------	-----------------------	---

```
02389
02390 write_words(sac_file, convert_to_word(static_cast<float>(sb()))); // 054
02391 write_words(sac_file, convert_to_word(static_cast<float>(sdelta()))); // 055
```

```
02392 write_words(sac_file, convert_to_word(depmen())); // 056
02393 write_words(sac_file, convert_to_word(cmpaz())); // 057
02394 write_words(sac_file, convert_to_word(cmpinc())); // 058
02395 write_words(sac_file, convert_to_word(xminimum())); // 059
02396 write_words(sac_file, convert_to_word(xmaximum())); // 060
02397 write_words(sac_file, convert_to_word(yminimum())); // 061
02398 write_words(sac_file, convert_to_word(ymaximum())); // 062
02399 // Fill 'unused' (xcommon_skip_num)
02400 for (int i{0}; i < common_skip_num; ++i) { // 063-069
02401     write_words(sac_file, convert_to_word(az()));
02402 }
02403 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.262 write_float_headers_resp()

```
void sacfmt::Trace::write_float_headers_resp (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 021–030.

Note that this expects the position of the writer to be the beginning of word 021.

Note that this modifies the position of the writer to the end of word 030.

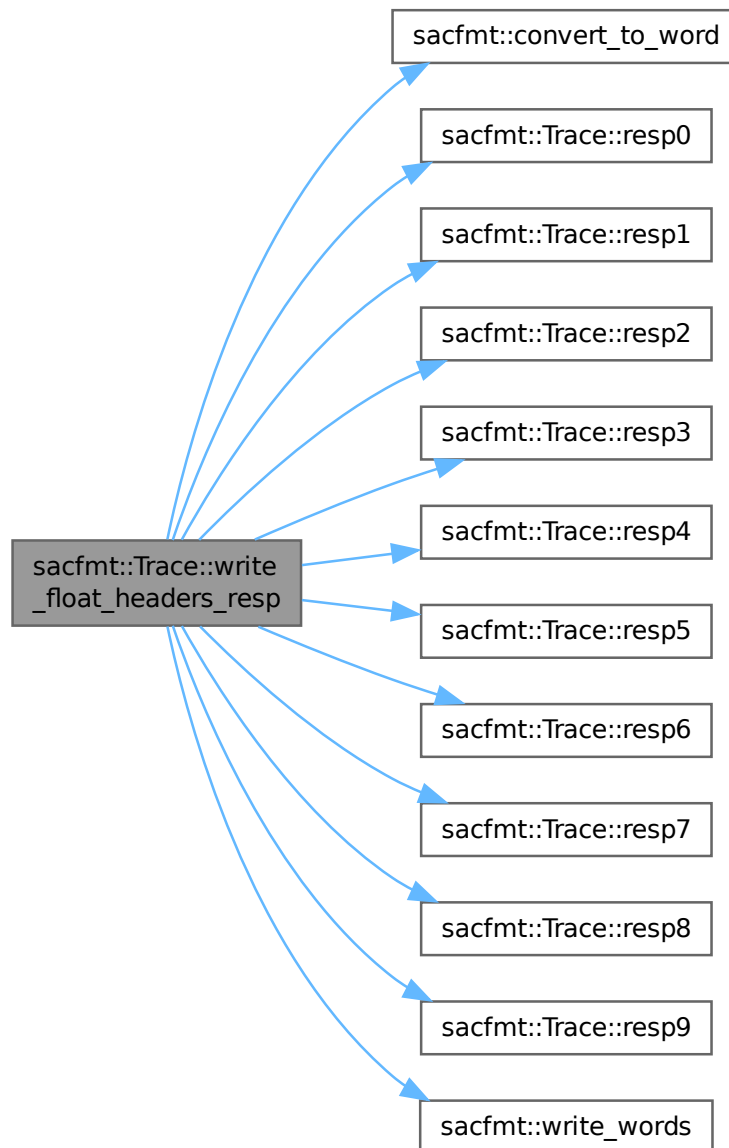
Headers written: resp0, resp1, resp2, resp3, resp4, resp5, resp6, resp7, resp8, and resp9.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02294
02295     write_words(sac_file, convert_to_word(resp0())); // 021
02296     write_words(sac_file, convert_to_word(resp1())); // 022
02297     write_words(sac_file, convert_to_word(resp2())); // 023
02298     write_words(sac_file, convert_to_word(resp3())); // 024
02299     write_words(sac_file, convert_to_word(resp4())); // 025
02300     write_words(sac_file, convert_to_word(resp5())); // 026
02301     write_words(sac_file, convert_to_word(resp6())); // 027
02302     write_words(sac_file, convert_to_word(resp7())); // 028
02303     write_words(sac_file, convert_to_word(resp8())); // 029
02304     write_words(sac_file, convert_to_word(resp9())); // 030
02305 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.263 write_float_headers_starter()

```
void sacfmt::Trace::write_float_headers_starter (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 000–009.

Note that this expects the position of the writer to be the beginning of word 000.

Note that this modifies the position of the writer to the end of word 009.

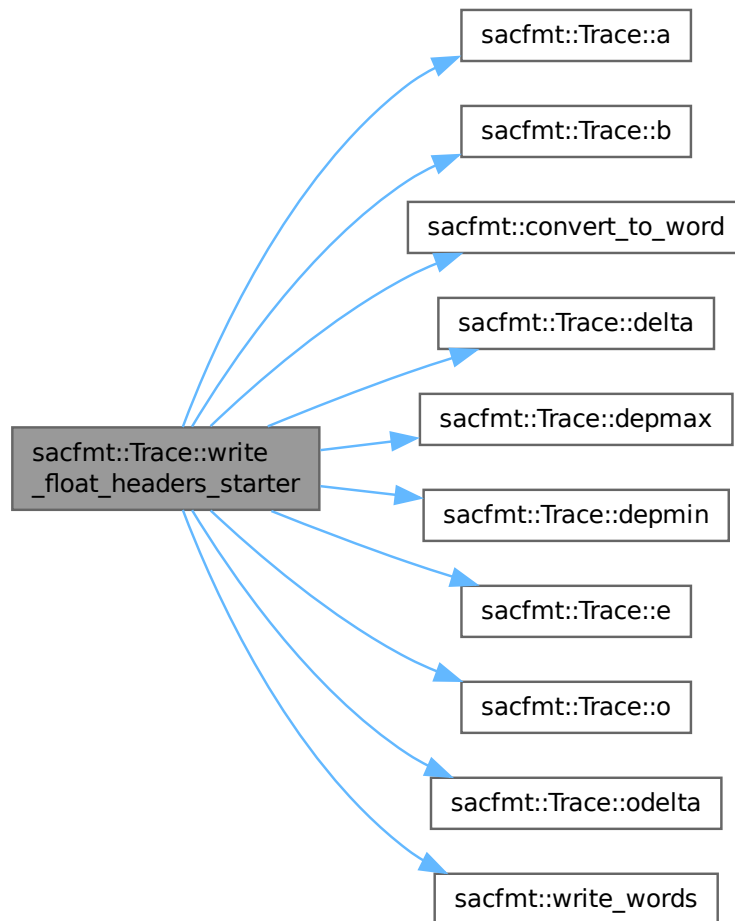
Headers written: delta, depmin, depmax, odelta, b, e, o, and a.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02240
02241     write_words(sac_file, convert_to_word(static_cast<float>(delta()))); // 000
02242     write_words(sac_file, convert_to_word(depmin())); // 001
02243     write_words(sac_file, convert_to_word(depmax())); // 002
02244     // Fill 'unused'
02245     write_words(sac_file, convert_to_word(depmax())); // 003
02246     write_words(sac_file, convert_to_word(odelta())); // 004
02247     write_words(sac_file, convert_to_word(static_cast<float>(b()))); // 005
02248     write_words(sac_file, convert_to_word(static_cast<float>(e()))); // 006
02249     write_words(sac_file, convert_to_word(static_cast<float>(o()))); // 007
02250     write_words(sac_file, convert_to_word(static_cast<float>(a()))); // 008
02251     // Fill 'internal'
02252     write_words(sac_file, convert_to_word(depmin())); // 009
02253 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.264 write_float_headers_station_event()

```

void sacfmt::Trace::write_float_headers_station_event (
    std::ofstream * sac_file ) const [private]
  
```

Writes SAC-headers from words 031–039.

Note that this expects the position of the writer to be the beginning of word 031.

Note that this modifies the position of the writer to the end of word 039.

Headers written: stla, stlo, stel, stdp, evla, evlo, evel, evdp, and mag.

Parameters

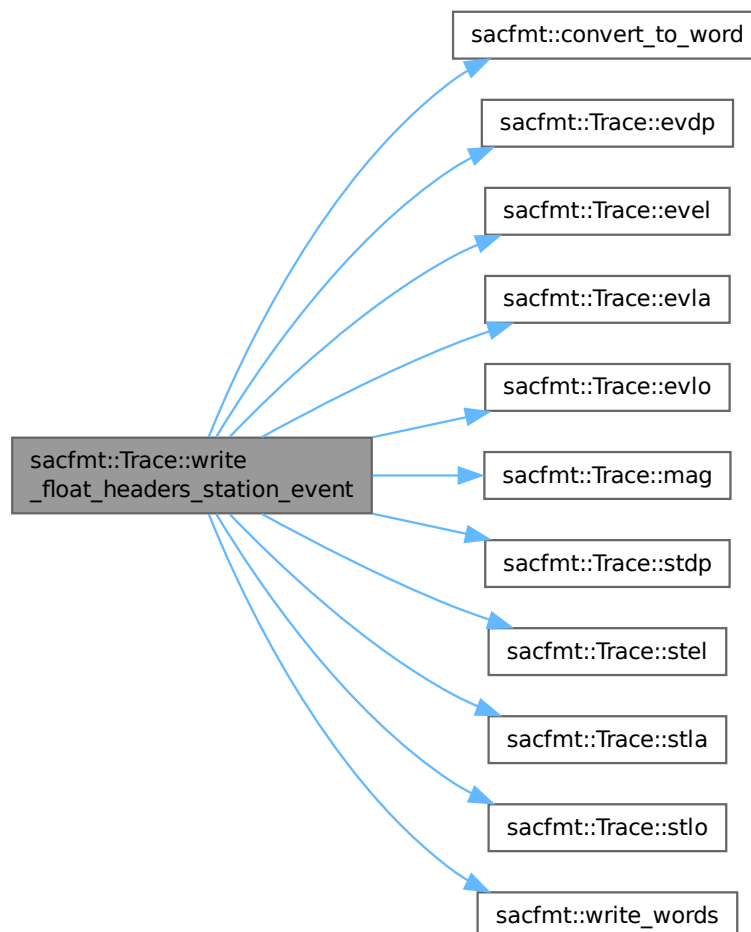
<code>in, out</code>	<code>sac_file</code>	<code>std::ofstream*</code> SAC-file to be written.
----------------------	-----------------------	---

```

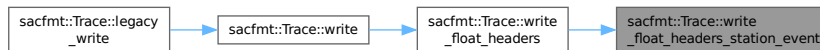
02319
02320 write_words(sac_file, convert_to_word(static_cast<float>(stla()))); // 031
02321 write_words(sac_file, convert_to_word(static_cast<float>(stlo()))); // 032
02322 write_words(sac_file, convert_to_word(stel())); // 033
02323 write_words(sac_file, convert_to_word(stdp())); // 034
02324 write_words(sac_file, convert_to_word(static_cast<float>(evla()))); // 035
02325 write_words(sac_file, convert_to_word(static_cast<float>(evlo()))); // 036
02326 write_words(sac_file, convert_to_word(evel())); // 037
02327 write_words(sac_file, convert_to_word(evdp())); // 038
02328 write_words(sac_file, convert_to_word(mag())); // 039
02329 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.265 write_float_headers_t()

```
void sacfmt::Trace::write_float_headers_t (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 010–020.

Note that this expects the position of the writer to be the beginning of word 010.

Note that this modifies the position of the writer to the end of word 020.

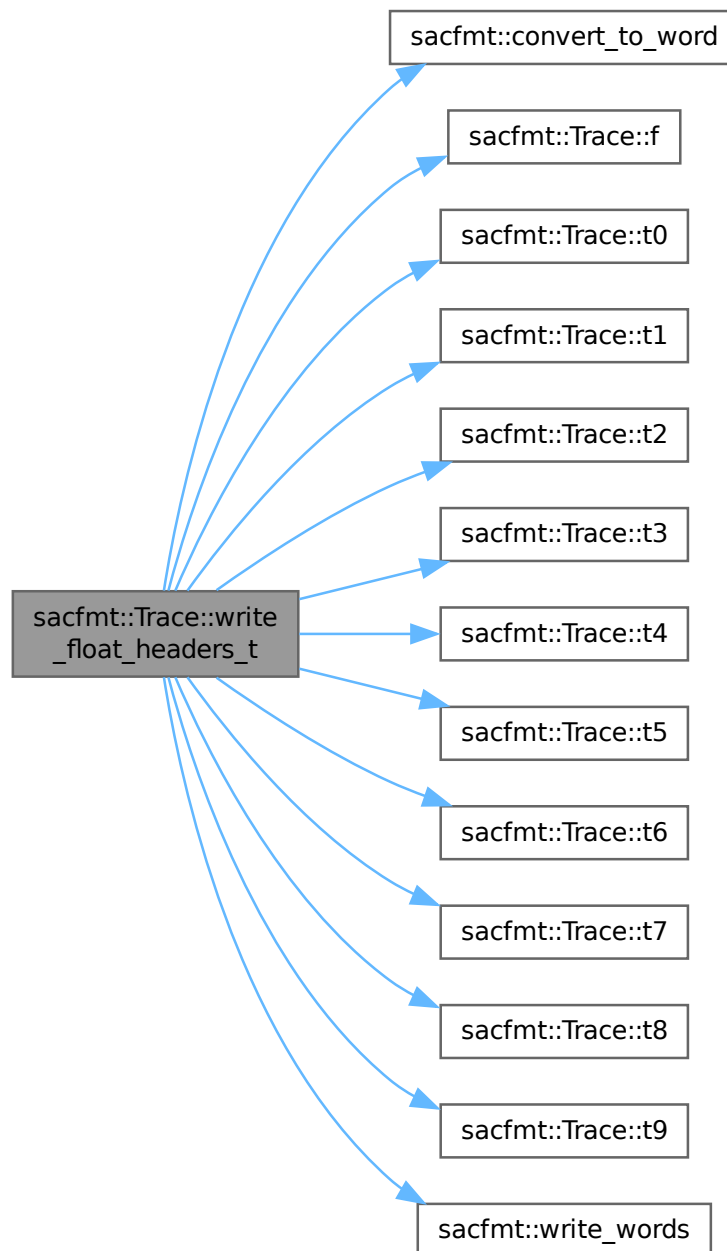
Headers written: t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, and f.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02267 {
02268     write_words(sac_file, convert_to_word(static_cast<float>(t0()))); // 010
02269     write_words(sac_file, convert_to_word(static_cast<float>(t1()))); // 011
02270     write_words(sac_file, convert_to_word(static_cast<float>(t2()))); // 012
02271     write_words(sac_file, convert_to_word(static_cast<float>(t3()))); // 013
02272     write_words(sac_file, convert_to_word(static_cast<float>(t4()))); // 014
02273     write_words(sac_file, convert_to_word(static_cast<float>(t5()))); // 015
02274     write_words(sac_file, convert_to_word(static_cast<float>(t6()))); // 016
02275     write_words(sac_file, convert_to_word(static_cast<float>(t7()))); // 017
02276     write_words(sac_file, convert_to_word(static_cast<float>(t8()))); // 018
02277     write_words(sac_file, convert_to_word(static_cast<float>(t9()))); // 019
02278     write_words(sac_file, convert_to_word(static_cast<float>(f()))); // 020
02279 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.266 write_float_headers_user()

```
void sacfmt::Trace::write_float_headers_user (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 040–049.

Note that this expects the position of the writer to be the beginning of word 040.

Note that this modifies the position of the writer to the end of word 049.

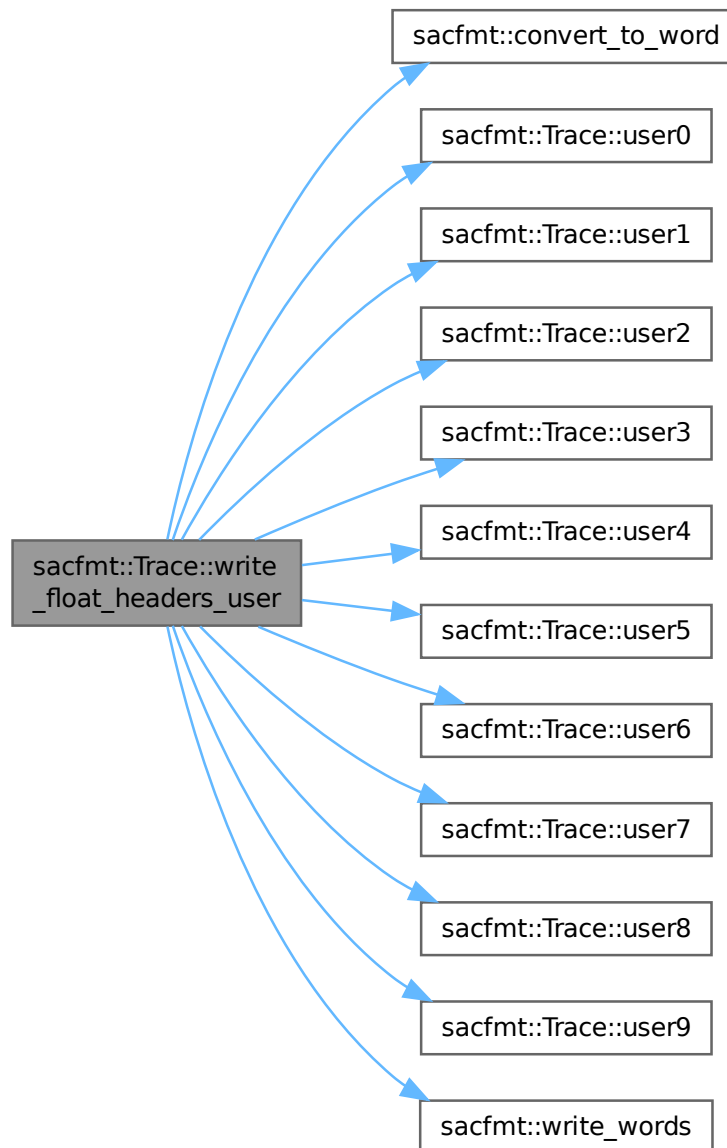
Headers written: user0, user1, user2, user3, user4, user5, user6, user7, user8, and user9.

Parameters

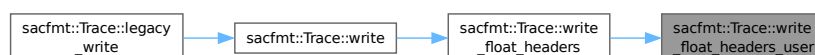
in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02344
02345 write_words(sac_file, convert_to_word(user0())); // 040
02346 write_words(sac_file, convert_to_word(user1())); // 041
02347 write_words(sac_file, convert_to_word(user2())); // 042
02348 write_words(sac_file, convert_to_word(user3())); // 043
02349 write_words(sac_file, convert_to_word(user4())); // 044
02350 write_words(sac_file, convert_to_word(user5())); // 045
02351 write_words(sac_file, convert_to_word(user6())); // 046
02352 write_words(sac_file, convert_to_word(user7())); // 047
02353 write_words(sac_file, convert_to_word(user8())); // 048
02354 write_words(sac_file, convert_to_word(user9())); // 049
02355 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.267 write_footers()

```
void sacfmt::Trace::write_footers (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-footers (post-data words 00–43).

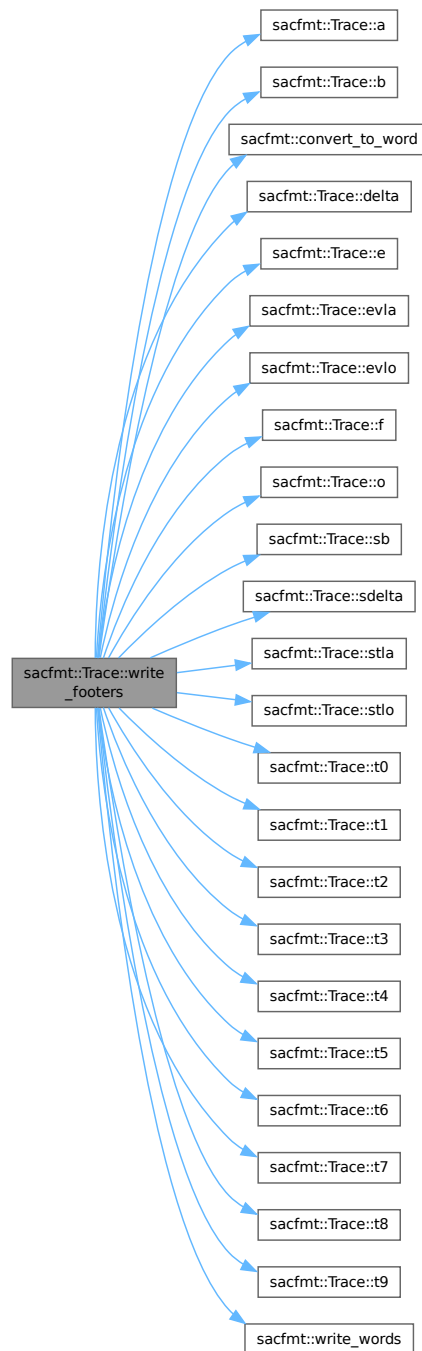
Note that this modifies the position of the writer to the end of the footer section.

Parameters

in, out	sac_file	std::ofstream* SAC-file to be written.
---------	----------	--

```
02652     {
02653     write_words(sac_file, convert_to_word(delta())); // 00-01
02654     write_words(sac_file, convert_to_word(b())); // 02-03
02655     write_words(sac_file, convert_to_word(e())); // 04-05
02656     write_words(sac_file, convert_to_word(o())); // 06-07
02657     write_words(sac_file, convert_to_word(a())); // 08-09
02658     write_words(sac_file, convert_to_word(t0())); // 10-11
02659     write_words(sac_file, convert_to_word(t1())); // 12-13
02660     write_words(sac_file, convert_to_word(t2())); // 14-15
02661     write_words(sac_file, convert_to_word(t3())); // 16-17
02662     write_words(sac_file, convert_to_word(t4())); // 18-19
02663     write_words(sac_file, convert_to_word(t5())); // 20-21
02664     write_words(sac_file, convert_to_word(t6())); // 22-23
02665     write_words(sac_file, convert_to_word(t7())); // 24-25
02666     write_words(sac_file, convert_to_word(t8())); // 26-27
02667     write_words(sac_file, convert_to_word(t9())); // 28-29
02668     write_words(sac_file, convert_to_word(f())); // 30-31
02669     write_words(sac_file, convert_to_word(evlo())); // 32-33
02670     write_words(sac_file, convert_to_word(evla())); // 34-35
02671     write_words(sac_file, convert_to_word(stlo())); // 36-37
02672     write_words(sac_file, convert_to_word(stla())); // 38-39
02673     write_words(sac_file, convert_to_word(sb())); // 40-41
02674     write_words(sac_file, convert_to_word(sdelta())); // 42-43
02675 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.268 write_int_headers()

```
void sacfmt::Trace::write_int_headers (
    std::ofstream * sac_file,
    int hdr_ver ) const [private]
```

Writes SAC-headers from words 070–104.

Note that this expects the position of the writer to be the beginning of word 070.

Note that this modifies the position of the writer to the end of word 104.

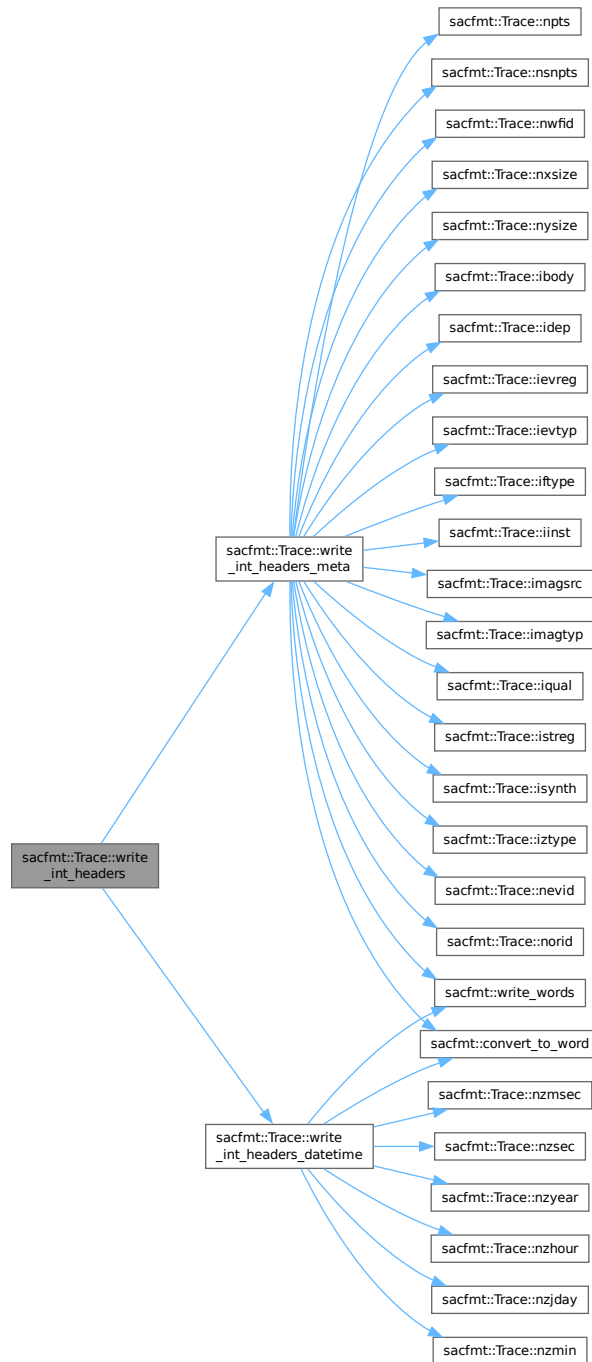
Writes all integer headers.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
in	<i>hdr_ver</i>	Integer header version to be written.

```
02509 {
02510     write_int_headers_datetime(sac_file); // 070-075
02511     write_int_headers_meta(sac_file, hdr_ver); // 076-104
02512 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.269 write_int_headers_datetime()

```
void sacfmt::Trace::write_int_headers_datetime (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 070–075.

Note that this expects the position of the writer to be the beginning of word 070.

Note that this modifies the position of the writer to the end of word 075.

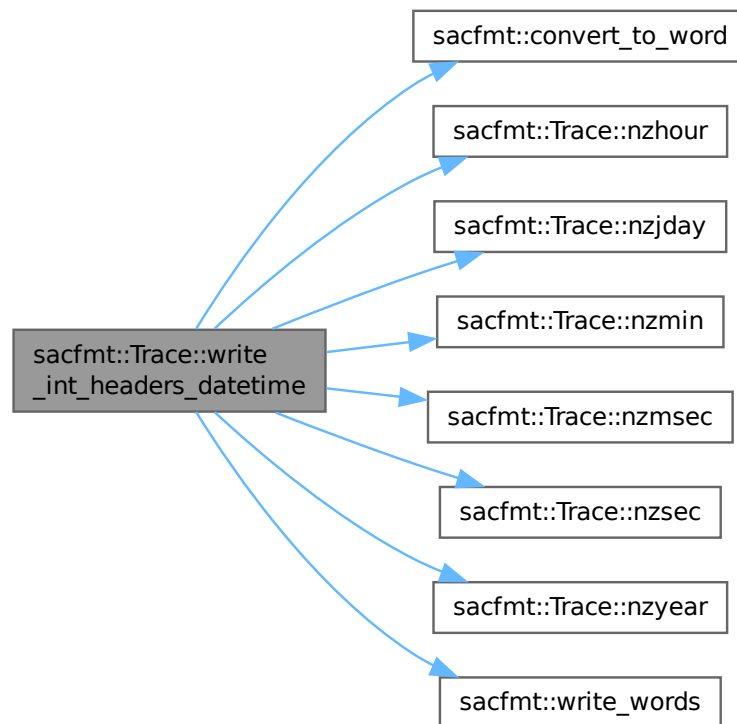
Headers written: nzyear, nzjday, nzhour, nzmin, nzsec, and nzmsec.

Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02439                                     {
02440     write_words(sac_file, convert_to_word(nzyear())); // 070
02441     write_words(sac_file, convert_to_word(nzjday())); // 071
02442     write_words(sac_file, convert_to_word(nzhour())); // 072
02443     write_words(sac_file, convert_to_word(nzmin())); // 073
02444     write_words(sac_file, convert_to_word(nzsec())); // 074
02445     write_words(sac_file, convert_to_word(nzmsec())); // 075
02446 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.270 write_int_headers_meta()

```

void sacfmt::Trace::write_int_headers_meta (
    std::ofstream * sac_file,
    int hdr_ver ) const [private]
  
```

Writes SAC-headers from words 076–104.

Note that this expects the position of the writer to be the beginning of word 076.

Note that this modifies the position of the writer to the end of word 104.

Headers written: `nvhdr`, `norid`, `nevid`, `npts`, `nsnpts`, `nwfid`, `nxsize`, `nysize`, `iftype`, `idep`, `iztype`, `iinst`, `istreg`, `ievreg`, `ievtyp`, `igual`, `isynth`, `imagtyp`, `imagsrc`, and `ibody`.

Parameters

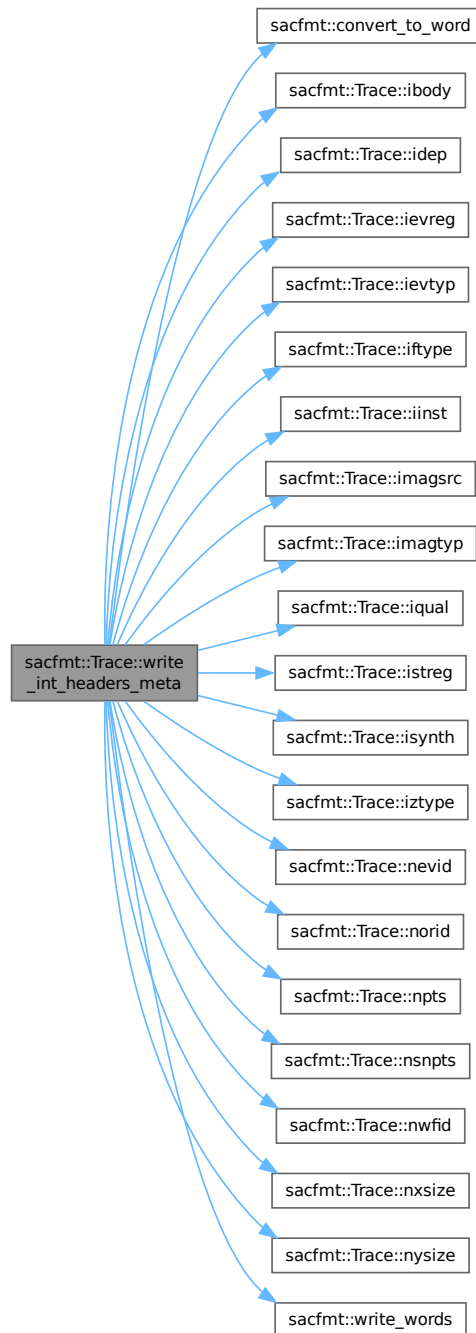
in, out	sac_file	std::ofstream* SAC-file to be written.
in	hdr_ver	Integer header version to be written.

```

02464
02465 write_words(sac_file, convert_to_word(hdr_ver)); // 076
02466 write_words(sac_file, convert_to_word(norid())); // 077
02467 write_words(sac_file, convert_to_word(nevid())); // 078
02468 write_words(sac_file, convert_to_word(npts())); // 079
02469 write_words(sac_file, convert_to_word(nsnpts())); // 080
02470 write_words(sac_file, convert_to_word(nwfid())); // 081
02471 write_words(sac_file, convert_to_word(nxsize())); // 082
02472 write_words(sac_file, convert_to_word(nysize())); // 083
02473 // Fill 'unused'
02474 write_words(sac_file, convert_to_word(nysize())); // 084
02475 write_words(sac_file, convert_to_word(iftyp())); // 085
02476 write_words(sac_file, convert_to_word(idep())); // 086
02477 write_words(sac_file, convert_to_word(iztyp())); // 087
02478 // Fill 'unused'
02479 write_words(sac_file, convert_to_word(iztyp())); // 088
02480 write_words(sac_file, convert_to_word(iinst())); // 089
02481 write_words(sac_file, convert_to_word(istreg())); // 090
02482 write_words(sac_file, convert_to_word(ievreg())); // 091
02483 write_words(sac_file, convert_to_word(ievtyp())); // 092
02484 write_words(sac_file, convert_to_word(igual())); // 093
02485 write_words(sac_file, convert_to_word(isynth())); // 094
02486 write_words(sac_file, convert_to_word(imagtyp())); // 095
02487 write_words(sac_file, convert_to_word(imagsrc())); // 096
02488 write_words(sac_file, convert_to_word(ibody())); // 097
02489 // Fill 'unused' (xcommon_skip_num)
02490 for (int i{0}; i < common_skip_num; ++i) { // 098-104
02491     write_words(sac_file, convert_to_word(ibody()));
02492 }
02493 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



11.5.3.271 write_string_headers()

```
void sacfmt::Trace::write_string_headers (
    std::ofstream * sac_file ) const [private]
```

Writes SAC-headers from words 110–157.

Note that this expects the position of the writer to be the beginning of word 110.

Note that this modifies the position of the writer to the end of word 157.

Writes all string headers.

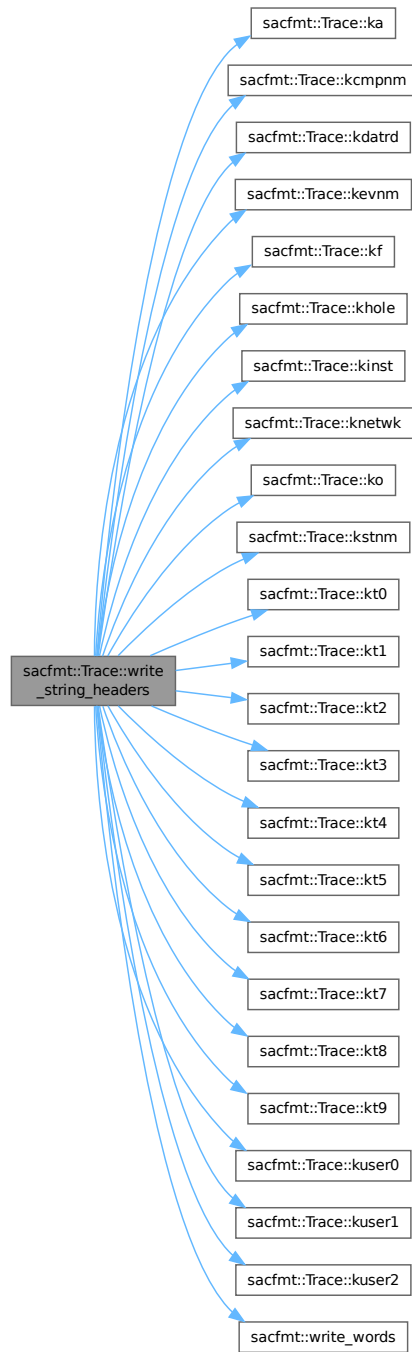
Parameters

in, out	<i>sac_file</i>	std::ofstream* SAC-file to be written.
---------	-----------------	--

```
02547                                     {
02548     // Strings are special
02549     std::array<char, static_cast<size_t>(2) * word_length> two_words{
02550         convert_to_words<sizeof(two_words)>(kstnm(), 2)};
02551     write_words(sac_file, std::vector<char>(two_words.begin(),
02552         two_words.end())); // 110-111
02553
02554     std::array<char, static_cast<size_t>(4) * word_length> four_words{
02555         convert_to_words<sizeof(four_words)>(kevm(), 4)};
02556     write_words(sac_file, std::vector<char>(four_words.begin(),
02557         four_words.end())); // 112-115
02558
02559     two_words = convert_to_words<sizeof(two_words)>(khole(), 2);
02560     write_words(sac_file, std::vector<char>(two_words.begin(),
02561         two_words.end())); // 116-117
02562
02563     two_words = convert_to_words<sizeof(two_words)>(ko(), 2);
02564     write_words(sac_file, std::vector<char>(two_words.begin(),
02565         two_words.end())); // 118-119
02566
02567     two_words = convert_to_words<sizeof(two_words)>(ka(), 2);
02568     write_words(sac_file, std::vector<char>(two_words.begin(),
02569         two_words.end())); // 120-121
02570
02571     two_words = convert_to_words<sizeof(two_words)>(kt0(), 2);
02572     write_words(sac_file, std::vector<char>(two_words.begin(),
02573         two_words.end())); // 122-123
02574
02575     two_words = convert_to_words<sizeof(two_words)>(kt1(), 2);
02576     write_words(sac_file, std::vector<char>(two_words.begin(),
02577         two_words.end())); // 124-125
02578
02579     two_words = convert_to_words<sizeof(two_words)>(kt2(), 2);
02580     write_words(sac_file, std::vector<char>(two_words.begin(),
02581         two_words.end())); // 126-127
02582
02583     two_words = convert_to_words<sizeof(two_words)>(kt3(), 2);
02584     write_words(sac_file, std::vector<char>(two_words.begin(),
02585         two_words.end())); // 128-129
02586
02587     two_words = convert_to_words<sizeof(two_words)>(kt4(), 2);
02588     write_words(sac_file, std::vector<char>(two_words.begin(),
02589         two_words.end())); // 130-131
02590
02591     two_words = convert_to_words<sizeof(two_words)>(kt5(), 2);
02592     write_words(sac_file, std::vector<char>(two_words.begin(),
02593         two_words.end())); // 132-133
02594
02595     two_words = convert_to_words<sizeof(two_words)>(kt6(), 2);
02596     write_words(sac_file, std::vector<char>(two_words.begin(),
02597         two_words.end())); // 134-135
02598
02599     two_words = convert_to_words<sizeof(two_words)>(kt7(), 2);
02600     write_words(sac_file, std::vector<char>(two_words.begin(),
02601         two_words.end())); // 136-137
02602
02603     two_words = convert_to_words<sizeof(two_words)>(kt8(), 2);
02604     write_words(sac_file, std::vector<char>(two_words.begin(),
02605         two_words.end())); // 138-139
02606
02607     two_words = convert_to_words<sizeof(two_words)>(kt9(), 2);
```

```
02608 write_words(sac_file, std::vector<char>(two_words.begin(),
02609                                     two_words.end())); // 140-141
02610
02611 two_words = convert_to_words<sizeof(two_words)>(kf(), 2);
02612 write_words(sac_file, std::vector<char>(two_words.begin(),
02613                                     two_words.end())); // 142-143
02614
02615 two_words = convert_to_words<sizeof(two_words)>(kuser0(), 2);
02616 write_words(sac_file, std::vector<char>(two_words.begin(),
02617                                     two_words.end())); // 144-145
02618
02619 two_words = convert_to_words<sizeof(two_words)>(kuser1(), 2);
02620 write_words(sac_file, std::vector<char>(two_words.begin(),
02621                                     two_words.end())); // 146-147
02622
02623 two_words = convert_to_words<sizeof(two_words)>(kuser2(), 2);
02624 write_words(sac_file, std::vector<char>(two_words.begin(),
02625                                     two_words.end())); // 148-149
02626
02627 two_words = convert_to_words<sizeof(two_words)>(kcmpnm(), 2);
02628 write_words(sac_file, std::vector<char>(two_words.begin(),
02629                                     two_words.end())); // 150-151
02630
02631 two_words = convert_to_words<sizeof(two_words)>(knetwk(), 2);
02632 write_words(sac_file, std::vector<char>(two_words.begin(),
02633                                     two_words.end())); // 152-153
02634
02635 two_words = convert_to_words<sizeof(two_words)>(kdatrd(), 2);
02636 write_words(sac_file, std::vector<char>(two_words.begin(),
02637                                     two_words.end())); // 154-155
02638
02639 two_words = convert_to_words<sizeof(two_words)>(kinst(), 2);
02640 write_words(sac_file, std::vector<char>(two_words.begin(),
02641                                     two_words.end())); // 156-157
02642 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

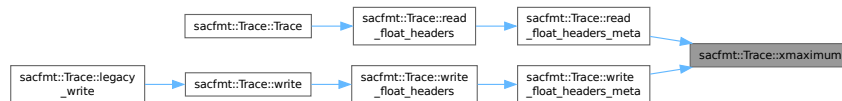


11.5.3.272 xmaximum() [1/2]

```

float sacfmt::Trace::xmaximum ( ) const [noexcept]
01075     {
01076     return floats[sac_map.at(name::xmaximum)];
01077     }
  
```

Here is the caller graph for this function:



11.5.3.273 xmaximum() [2/2]

```

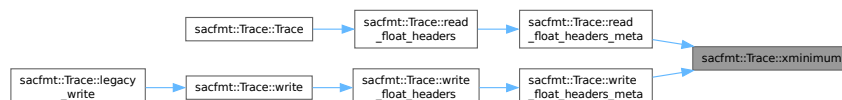
void sacfmt::Trace::xmaximum (
    float input ) [noexcept]
01324     {
01325     floats[sac_map.at(name::xmaximum)] = input;
01326     }
  
```

11.5.3.274 xminimum() [1/2]

```

float sacfmt::Trace::xminimum ( ) const [noexcept]
01072     {
01073     return floats[sac_map.at(name::xminimum)];
01074     }
  
```

Here is the caller graph for this function:



11.5.3.275 xminimum() [2/2]

```

void sacfmt::Trace::xminimum (
    float input ) [noexcept]
01321     {
01322     floats[sac_map.at(name::xminimum)] = input;
01323     }

```

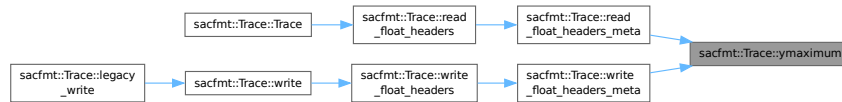
11.5.3.276 ymaximum() [1/2]

```

float sacfmt::Trace::ymaximum ( ) const [noexcept]
01081     {
01082     return floats[sac_map.at(name::ymaximum)];
01083     }

```

Here is the caller graph for this function:



11.5.3.277 ymaximum() [2/2]

```

void sacfmt::Trace::ymaximum (
    float input ) [noexcept]
01330     {
01331     floats[sac_map.at(name::ymaximum)] = input;
01332     }

```

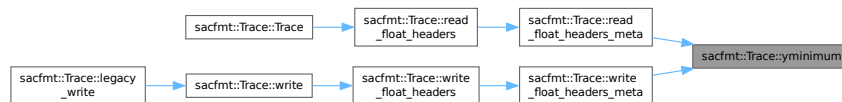
11.5.3.278 yminimum() [1/2]

```

float sacfmt::Trace::yminimum ( ) const [noexcept]
01078     {
01079     return floats[sac_map.at(name::yminimum)];
01080     }

```

Here is the caller graph for this function:



11.5.3.279 yminimum() [2/2]

```

void sacfmt::Trace::yminimum (
    float input ) [noexcept]
01327     {
01328     floats[sac_map.at(name::yminimum)] = input;
01329     }

```

11.5.4 Member Data Documentation

11.5.4.1 bools

```
std::array<bool, num_bool> sacfmt::Trace::bools {} [private]
```

Boolean storage array.

```
01406 {};
```

11.5.4.2 data

```
std::array<std::vector<double>, num_data> sacfmt::Trace::data {} [private]
```

std::vector<double> storage array.

```
01411 {};
```

11.5.4.3 doubles

```
std::array<double, num_double> sacfmt::Trace::doubles {} [private]
```

Double storage array.

```
01402 {};
```

11.5.4.4 floats

```
std::array<float, num_float> sacfmt::Trace::floats {} [private]
```

Float storage array.

```
01400 {};
```

11.5.4.5 ints

```
std::array<int, num_int> sacfmt::Trace::ints {} [private]
```

Integer storage array.

```
01404 {};
```

11.5.4.6 strings

```
std::array<std::string, num_string> sacfmt::Trace::strings {} [private]
```

String storage array.

```
01408 {};
```

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

- include/sac-format/sac_format.hpp
- src/sac_format.cpp

11.6 `sacfmt::bitset_type::uint< nbits >` Struct Template Reference

Ensure type-safety for conversions between floats/doubles and bitsets.

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

11.6.1 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`

11.7 `sacfmt::bitset_type::uint< 4 *bits_per_byte >` Struct Reference

One-word (floats).

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

Public Types

- `using type = uint32_t`

11.7.1 Detailed Description

One-word (floats).

11.7.2 Member Typedef Documentation

11.7.2.1 `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`

11.8 `sacfmt::bitset_type::uint< bytes *bits_per_byte >` Struct Reference

Two-words (doubles)

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

Public Types

- `using type = uint64_t`

11.8.1 Detailed Description

Two-words (doubles)

11.8.2 Member Typedef Documentation

11.8.2.1 `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`

11.9 `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.

11.9.1 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`.

11.9.2 Member Data Documentation

11.9.2.1 first

```
template<typename T >  
T sacfmt::word_pair< T >::first {}
```

First 'word' in the pair.
00192 {};

11.9.2.2 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, [58](#)
 - sacfmt::Trace, [121](#), [122](#)
- ascii_space
 - sacfmt, [98](#)
- az
 - sacfmt, [57](#)
 - sacfmt::Trace, [122](#)
- azimuth
 - sacfmt, [62](#)
- b
 - sacfmt, [58](#)
 - sacfmt::Trace, [122](#), [123](#)
- Basic Documentation, [17](#)
- baz
 - sacfmt, [57](#)
 - sacfmt::Trace, [123](#)
- binary_to_bool
 - sacfmt, [63](#)
- binary_to_double
 - sacfmt, [64](#)
- binary_to_float
 - sacfmt, [64](#)
- binary_to_int
 - sacfmt, [65](#)
- binary_to_long_string
 - sacfmt, [66](#)
- binary_to_string
 - sacfmt, [67](#)
- binary_word_size
 - sacfmt, [98](#)
- bits_per_byte
 - sacfmt, [98](#)
- bits_string
 - sacfmt, [68](#)
- bool_to_binary
 - sacfmt, [68](#)
- bool_to_word
 - sacfmt, [69](#)
- bools
 - sacfmt::Trace, [248](#)
- Build Instructions, [43](#)
- bytes
 - sacfmt::bitset_type, [103](#)
- calc_az
 - sacfmt::Trace, [123](#)
- calc_baz
 - sacfmt::Trace, [124](#)
- calc_dist
 - sacfmt::Trace, [125](#)
- calc_gcarc
 - sacfmt::Trace, [126](#)
- calc_geometry
 - sacfmt::Trace, [127](#)
- char_bit
 - sacfmt, [55](#)
- circle_deg
 - sacfmt, [98](#)
- cmpaz
 - sacfmt, [57](#)
 - sacfmt::Trace, [127](#), [128](#)
- cmpinc
 - sacfmt, [57](#)
 - sacfmt::Trace, [128](#)
- common_skip_num
 - sacfmt, [98](#)
- concat_words
 - sacfmt, [69](#), [70](#)
- convert_to_word
 - sacfmt, [70](#), [72](#)
- convert_to_words
 - sacfmt, [73](#)
- coord
 - sacfmt::coord, [106](#)
- data
 - sacfmt::Trace, [248](#)
- data1
 - sacfmt, [61](#)
 - sacfmt::Trace, [128](#), [129](#)
- data2
 - sacfmt, [61](#)
 - sacfmt::Trace, [129](#)
- data_word
 - sacfmt, [99](#)
- date
 - sacfmt::Trace, [130](#)
- deg
 - sacfmt::coord, [108](#)
- deg_per_rad
 - sacfmt, [99](#)
- degrees
 - sacfmt::coord, [106](#)
- degrees_to_radians
 - sacfmt, [73](#)
- delta
 - sacfmt, [58](#)
 - sacfmt::Trace, [130](#), [131](#)

- depmax
 - sacfmt, 56
 - sacfmt::Trace, 131
- depmen
 - sacfmt, 57
 - sacfmt::Trace, 131, 132
- depmin
 - sacfmt, 56
 - sacfmt::Trace, 132
- dist
 - sacfmt, 57
 - sacfmt::Trace, 132, 133
- double_to_binary
 - sacfmt, 74
- doubles
 - sacfmt::Trace, 248
- e
 - sacfmt, 58
 - sacfmt::Trace, 133
- earth_radius
 - sacfmt, 99
- equal_within_tolerance
 - sacfmt, 75
- evdp
 - sacfmt, 57
 - sacfmt::Trace, 133, 134
- evel
 - sacfmt, 57
 - sacfmt::Trace, 134
- event_location
 - sacfmt::Trace, 134
- evla
 - sacfmt, 58
 - sacfmt::Trace, 135, 136
- evlo
 - sacfmt, 58
 - sacfmt::Trace, 136, 137
- f
 - sacfmt, 58
 - sacfmt::Trace, 137
- f_eps
 - sacfmt, 99
- first
 - sacfmt::word_pair< T >, 251
- float_to_binary
 - sacfmt, 76
- floats
 - sacfmt::Trace, 248
- frequency
 - sacfmt::Trace, 138
- gcarc
 - sacfmt, 57, 77
 - sacfmt::Trace, 138
- geometry_set
 - sacfmt::Trace, 139
- ibody
 - sacfmt, 60
 - sacfmt::Trace, 139, 140
- idep
 - sacfmt, 59
 - sacfmt::Trace, 140
- ievreg
 - sacfmt, 59
 - sacfmt::Trace, 140, 141
- ievtyp
 - sacfmt, 59
 - sacfmt::Trace, 141
- iftyp
 - sacfmt, 59
 - sacfmt::Trace, 141, 142
- iinst
 - sacfmt, 59
 - sacfmt::Trace, 142
- imagsrc
 - sacfmt, 60
 - sacfmt::Trace, 142, 143
- imagtyp
 - sacfmt, 60
 - sacfmt::Trace, 143
- Installation, 3
- int_to_binary
 - sacfmt, 77
- Introduction, 1
- ints
 - sacfmt::Trace, 248
- io_error
 - sacfmt::io_error, 109
- iqua
 - sacfmt, 59
 - sacfmt::Trace, 143, 144
- istreg
 - sacfmt, 59
 - sacfmt::Trace, 144
- isynth
 - sacfmt, 59
 - sacfmt::Trace, 144, 145
- iztype
 - sacfmt, 59
 - sacfmt::Trace, 145
- ka
 - sacfmt, 60
 - sacfmt::Trace, 145, 146
- kcmpnm
 - sacfmt, 60
 - sacfmt::Trace, 146
- kdatrd
 - sacfmt, 61
 - sacfmt::Trace, 146, 147
- kevmn
 - sacfmt, 60
 - sacfmt::Trace, 147
- kf
 - sacfmt, 60

- sacfmt::Trace, 147, 148
- khole
 - sacfmt, 60
 - sacfmt::Trace, 148
- kinst
 - sacfmt, 61
 - sacfmt::Trace, 148, 149
- knetwk
 - sacfmt, 60
 - sacfmt::Trace, 149
- ko
 - sacfmt, 60
 - sacfmt::Trace, 149, 150
- kstnm
 - sacfmt, 60
 - sacfmt::Trace, 150
- kt0
 - sacfmt, 60
 - sacfmt::Trace, 150, 151
- kt1
 - sacfmt, 60
 - sacfmt::Trace, 151
- kt2
 - sacfmt, 60
 - sacfmt::Trace, 151, 152
- kt3
 - sacfmt, 60
 - sacfmt::Trace, 152
- kt4
 - sacfmt, 60
 - sacfmt::Trace, 152, 153
- kt5
 - sacfmt, 60
 - sacfmt::Trace, 153
- kt6
 - sacfmt, 60
 - sacfmt::Trace, 153, 154
- kt7
 - sacfmt, 60
 - sacfmt::Trace, 154
- kt8
 - sacfmt, 60
 - sacfmt::Trace, 154, 155
- kt9
 - sacfmt, 60
 - sacfmt::Trace, 155
- kuser0
 - sacfmt, 60
 - sacfmt::Trace, 155, 156
- kuser1
 - sacfmt, 60
 - sacfmt::Trace, 156
- kuser2
 - sacfmt, 60
 - sacfmt::Trace, 156, 157
- latitude
 - sacfmt::point, 111
- lcalda
 - sacfmt, 60
 - sacfmt::Trace, 157
- legacy_write
 - sacfmt::Trace, 157
- leven
 - sacfmt, 60
 - sacfmt::Trace, 158, 159
- limit_180
 - sacfmt, 78
- limit_360
 - sacfmt, 79
- limit_90
 - sacfmt, 80
- long_string_to_binary
 - sacfmt, 81
- longitude
 - sacfmt::point, 111
- lovrok
 - sacfmt, 60
 - sacfmt::Trace, 159
- lpspol
 - sacfmt, 60
 - sacfmt::Trace, 159, 160
- mag
 - sacfmt, 57
 - sacfmt::Trace, 160
- message
 - sacfmt::io_error, 110
- modern_hdr_version
 - sacfmt, 99
- name
 - sacfmt, 56
- nevid
 - sacfmt, 59
 - sacfmt::Trace, 160, 161
- norid
 - sacfmt, 59
 - sacfmt::Trace, 161
- npts
 - sacfmt, 59
 - sacfmt::Trace, 161, 162
- nsnpts
 - sacfmt, 59
 - sacfmt::Trace, 162
- num_bool
 - sacfmt, 99
- num_data
 - sacfmt, 99
- num_double
 - sacfmt, 100
- num_float
 - sacfmt, 100
- num_footer
 - sacfmt, 100
- num_int
 - sacfmt, 100
- num_string

- sacfmt, 100
- num_words
 - sacfmt::read_spec, 112
- nvhdr
 - sacfmt, 59
 - sacfmt::Trace, 162, 163
- nwfid
 - sacfmt, 59
 - sacfmt::Trace, 163
- nwords_after_current
 - sacfmt, 82
- nxsize
 - sacfmt, 59
 - sacfmt::Trace, 163, 164
- nysize
 - sacfmt, 59
 - sacfmt::Trace, 164
- nzhour
 - sacfmt, 59
 - sacfmt::Trace, 164, 165
- nzjday
 - sacfmt, 58
 - sacfmt::Trace, 165
- nzmin
 - sacfmt, 59
 - sacfmt::Trace, 165, 166
- nzmsec
 - sacfmt, 59
 - sacfmt::Trace, 166
- nzsec
 - sacfmt, 59
 - sacfmt::Trace, 166, 167
- nzyear
 - sacfmt, 58
 - sacfmt::Trace, 167
- o
 - sacfmt, 58
 - sacfmt::Trace, 167, 168
- odelta
 - sacfmt, 56
 - sacfmt::Trace, 168
- old_hdr_version
 - sacfmt, 100
- operator==
 - sacfmt::Trace, 168
- point
 - sacfmt::point, 111
- prep_string
 - sacfmt, 82
- Quickstart, 15
- rad
 - sacfmt::coord, 108
- rad_per_deg
 - sacfmt, 100
- radians
 - sacfmt::coord, 107
- radians_to_degrees
 - sacfmt, 83
- read_bool_headers
 - sacfmt::Trace, 169
- read_data
 - sacfmt, 84
- read_datas
 - sacfmt::Trace, 170
- read_float_headers
 - sacfmt::Trace, 172
- read_float_headers_geometry
 - sacfmt::Trace, 174
- read_float_headers_meta
 - sacfmt::Trace, 175
- read_float_headers_resp
 - sacfmt::Trace, 177
- read_float_headers_starter
 - sacfmt::Trace, 178
- read_float_headers_station_event
 - sacfmt::Trace, 180
- read_float_headers_t
 - sacfmt::Trace, 182
- read_float_headers_user
 - sacfmt::Trace, 185
- read_footers
 - sacfmt::Trace, 186
- read_four_words
 - sacfmt, 85
- read_int_headers
 - sacfmt::Trace, 189
- read_int_headers_datetime
 - sacfmt::Trace, 191
- read_int_headers_meta
 - sacfmt::Trace, 192
- read_string_headers
 - sacfmt::Trace, 195
- read_two_words
 - sacfmt, 86
- read_word
 - sacfmt, 87
- remove_leading_spaces
 - sacfmt, 88
- remove_trailing_spaces
 - sacfmt, 89
- resize_data
 - sacfmt::Trace, 197
- resize_data1
 - sacfmt::Trace, 197
- resize_data2
 - sacfmt::Trace, 197
- resp0
 - sacfmt, 56
 - sacfmt::Trace, 197, 198
- resp1
 - sacfmt, 56
 - sacfmt::Trace, 198
- resp2

- sacfmt, 56
- sacfmt::Trace, 198, 199
- resp3
 - sacfmt, 57
 - sacfmt::Trace, 199
- resp4
 - sacfmt, 57
 - sacfmt::Trace, 199, 200
- resp5
 - sacfmt, 57
 - sacfmt::Trace, 200
- resp6
 - sacfmt, 57
 - sacfmt::Trace, 200, 201
- resp7
 - sacfmt, 57
 - sacfmt::Trace, 201
- resp8
 - sacfmt, 57
 - sacfmt::Trace, 201, 202
- resp9
 - sacfmt, 57
 - sacfmt::Trace, 202
- SAC-file format, 27
- sac_map
 - sacfmt, 101
- sacfmt, 51
 - a, 58
 - ascii_space, 98
 - az, 57
 - azimuth, 62
 - b, 58
 - baz, 57
 - binary_to_bool, 63
 - binary_to_double, 64
 - binary_to_float, 64
 - binary_to_int, 65
 - binary_to_long_string, 66
 - binary_to_string, 67
 - binary_word_size, 98
 - bits_per_byte, 98
 - bits_string, 68
 - bool_to_binary, 68
 - bool_to_word, 69
 - char_bit, 55
 - circle_deg, 98
 - cmpaz, 57
 - cmpinc, 57
 - common_skip_num, 98
 - concat_words, 69, 70
 - convert_to_word, 70, 72
 - convert_to_words, 73
 - data1, 61
 - data2, 61
 - data_word, 99
 - deg_per_rad, 99
 - degrees_to_radians, 73
 - delta, 58
 - depmax, 56
 - depmen, 57
 - depmin, 56
 - dist, 57
 - double_to_binary, 74
 - e, 58
 - earth_radius, 99
 - equal_within_tolerance, 75
 - evdp, 57
 - evel, 57
 - evla, 58
 - evlo, 58
 - f, 58
 - f_eps, 99
 - float_to_binary, 76
 - gcarc, 57, 77
 - ibody, 60
 - idep, 59
 - ievreg, 59
 - ievtyp, 59
 - iftyp, 59
 - iinst, 59
 - imagsrc, 60
 - imagtyp, 60
 - int_to_binary, 77
 - igual, 59
 - istreg, 59
 - isynth, 59
 - iztype, 59
 - ka, 60
 - kcmpnm, 60
 - kdatrd, 61
 - kevnrm, 60
 - kf, 60
 - khole, 60
 - kinst, 61
 - knetwk, 60
 - ko, 60
 - kstnm, 60
 - kt0, 60
 - kt1, 60
 - kt2, 60
 - kt3, 60
 - kt4, 60
 - kt5, 60
 - kt6, 60
 - kt7, 60
 - kt8, 60
 - kt9, 60
 - kuser0, 60
 - kuser1, 60
 - kuser2, 60
 - lcalda, 60
 - leven, 60
 - limit_180, 78
 - limit_360, 79
 - limit_90, 80
 - long_string_to_binary, 81

- lovrok, 60
- lpspol, 60
- mag, 57
- modern_hdr_version, 99
- name, 56
- nevid, 59
- norid, 59
- npts, 59
- nsnpts, 59
- num_bool, 99
- num_data, 99
- num_double, 100
- num_float, 100
- num_footer, 100
- num_int, 100
- num_string, 100
- nvhdr, 59
- nwfid, 59
- nwords_after_current, 82
- nxsize, 59
- nysize, 59
- nzhour, 59
- nzjday, 58
- nzmin, 59
- nzmsec, 59
- nzsec, 59
- nzyear, 58
- o, 58
- odelta, 56
- old_hdr_version, 100
- prep_string, 82
- rad_per_deg, 100
- radians_to_degrees, 83
- read_data, 84
- read_four_words, 85
- read_two_words, 86
- read_word, 87
- remove_leading_spaces, 88
- remove_trailing_spaces, 89
- resp0, 56
- resp1, 56
- resp2, 56
- resp3, 57
- resp4, 57
- resp5, 57
- resp6, 57
- resp7, 57
- resp8, 57
- resp9, 57
- sac_map, 101
- safe_to_finish_reading, 89
- safe_to_read_data, 90
- safe_to_read_footer, 91
- safe_to_read_header, 92
- sb, 58
- sdelta, 58
- stdp, 57
- stel, 57
- stla, 58
- stlo, 58
- string_bits, 93
- string_cleaning, 94
- string_to_binary, 95
- t0, 58
- t1, 58
- t2, 58
- t3, 58
- t4, 58
- t5, 58
- t6, 58
- t7, 58
- t8, 58
- t9, 58
- uint_to_binary, 95
- unset_bool, 102
- unset_double, 102
- unset_float, 102
- unset_int, 103
- unset_word, 103
- unsigned_int, 55
- user0, 57
- user1, 57
- user2, 57
- user3, 57
- user4, 57
- user5, 57
- user6, 57
- user7, 57
- user8, 57
- user9, 57
- word_four, 56
- word_length, 103
- word_one, 56
- word_position, 96
- word_two, 56
- write_words, 97
- xmaximum, 58
- xminimum, 57
- ymaximum, 58
- yminimum, 58
- sacfmt::bitset_type, 103
 - bytes, 103
- sacfmt::bitset_type::uint< 4 *bits_per_byte >, 249
 - type, 249
- sacfmt::bitset_type::uint< bytes *bits_per_byte >, 250
 - type, 250
- sacfmt::bitset_type::uint< nbits >, 249
- sacfmt::coord, 105
 - coord, 106
 - deg, 108
 - degrees, 106
 - rad, 108
 - radians, 107
- sacfmt::io_error, 108
 - io_error, 109
 - message, 110

- what, 110
- sacfmt::point, 110
 - latitude, 111
 - longitude, 111
 - point, 111
- sacfmt::read_spec, 112
 - num_words, 112
 - start_word, 112
- sacfmt::Trace, 112
 - a, 121, 122
 - az, 122
 - b, 122, 123
 - baz, 123
 - bools, 248
 - calc_az, 123
 - calc_baz, 124
 - calc_dist, 125
 - calc_gcarc, 126
 - calc_geometry, 127
 - cmpaz, 127, 128
 - cmpinc, 128
 - data, 248
 - data1, 128, 129
 - data2, 129
 - date, 130
 - delta, 130, 131
 - depmax, 131
 - depmen, 131, 132
 - depmin, 132
 - dist, 132, 133
 - doubles, 248
 - e, 133
 - evdp, 133, 134
 - evel, 134
 - event_location, 134
 - evla, 135, 136
 - evlo, 136, 137
 - f, 137
 - floats, 248
 - frequency, 138
 - gcarc, 138
 - geometry_set, 139
 - ibody, 139, 140
 - idep, 140
 - ievreg, 140, 141
 - ievtyp, 141
 - iftyp, 141, 142
 - iinst, 142
 - imagsrc, 142, 143
 - imagtyp, 143
 - ints, 248
 - igual, 143, 144
 - istreg, 144
 - isynth, 144, 145
 - iztype, 145
 - ka, 145, 146
 - kcmpnm, 146
 - kdatrd, 146, 147
 - kevnrm, 147
 - kf, 147, 148
 - khole, 148
 - kinst, 148, 149
 - knetwk, 149
 - ko, 149, 150
 - kstnm, 150
 - kt0, 150, 151
 - kt1, 151
 - kt2, 151, 152
 - kt3, 152
 - kt4, 152, 153
 - kt5, 153
 - kt6, 153, 154
 - kt7, 154
 - kt8, 154, 155
 - kt9, 155
 - kuser0, 155, 156
 - kuser1, 156
 - kuser2, 156, 157
 - lcalda, 157
 - legacy_write, 157
 - leven, 158, 159
 - lovrok, 159
 - lpssol, 159, 160
 - mag, 160
 - nevid, 160, 161
 - norid, 161
 - npts, 161, 162
 - nsnpts, 162
 - nvhdr, 162, 163
 - nwfid, 163
 - nysize, 163, 164
 - nysize, 164
 - nzhour, 164, 165
 - nzjday, 165
 - nzmin, 165, 166
 - nzmsec, 166
 - nzsec, 166, 167
 - nzyear, 167
 - o, 167, 168
 - odelta, 168
 - operator==, 168
 - read_bool_headers, 169
 - read_datas, 170
 - read_float_headers, 172
 - read_float_headers_geometry, 174
 - read_float_headers_meta, 175
 - read_float_headers_resp, 177
 - read_float_headers_starter, 178
 - read_float_headers_station_event, 180
 - read_float_headers_t, 182
 - read_float_headers_user, 185
 - read_footers, 186
 - read_int_headers, 189
 - read_int_headers_datetime, 191
 - read_int_headers_meta, 192
 - read_string_headers, 195

resize_data, 197
 resize_data1, 197
 resize_data2, 197
 resp0, 197, 198
 resp1, 198
 resp2, 198, 199
 resp3, 199
 resp4, 199, 200
 resp5, 200
 resp6, 200, 201
 resp7, 201
 resp8, 201, 202
 resp9, 202
 sb, 202, 203
 sdelta, 203
 station_location, 203
 stdp, 204
 stel, 205
 stla, 205
 stlo, 206
 strings, 248
 t0, 207
 t1, 207
 t2, 208
 t3, 208
 t4, 209
 t5, 209
 t6, 210
 t7, 210
 t8, 211
 t9, 211
 time, 212
 Trace, 120
 user0, 212, 213
 user1, 213
 user2, 213, 214
 user3, 214
 user4, 214, 215
 user5, 215
 user6, 215, 216
 user7, 216
 user8, 216, 217
 user9, 217
 write, 217
 write_bool_headers, 219
 write_data, 220
 write_float_headers, 221
 write_float_headers_geometry, 223
 write_float_headers_meta, 224
 write_float_headers_resp, 226
 write_float_headers_starter, 227
 write_float_headers_station_event, 229
 write_float_headers_t, 231
 write_float_headers_user, 232
 write_footers, 234
 write_int_headers, 237
 write_int_headers_datetime, 239
 write_int_headers_meta, 240
 write_string_headers, 243
 xmaximum, 246
 xminimum, 246
 ymaximum, 247
 yminimum, 247
 sacfmt::word_pair< T >, 250
 first, 251
 second, 251
 safe_to_finish_reading
 sacfmt, 89
 safe_to_read_data
 sacfmt, 90
 safe_to_read_footer
 sacfmt, 91
 safe_to_read_header
 sacfmt, 92
 sb
 sacfmt, 58
 sacfmt::Trace, 202, 203
 sdelta
 sacfmt, 58
 sacfmt::Trace, 203
 second
 sacfmt::word_pair< T >, 251
 start_word
 sacfmt::read_spec, 112
 station_location
 sacfmt::Trace, 203
 stdp
 sacfmt, 57
 sacfmt::Trace, 204
 stel
 sacfmt, 57
 sacfmt::Trace, 205
 stla
 sacfmt, 58
 sacfmt::Trace, 205
 stlo
 sacfmt, 58
 sacfmt::Trace, 206
 string_bits
 sacfmt, 93
 string_cleaning
 sacfmt, 94
 string_to_binary
 sacfmt, 95
 strings
 sacfmt::Trace, 248
 t0
 sacfmt, 58
 sacfmt::Trace, 207
 t1
 sacfmt, 58
 sacfmt::Trace, 207
 t2
 sacfmt, 58
 sacfmt::Trace, 208
 t3

- sacfmt, [58](#)
- sacfmt::Trace, [208](#)
- t4
 - sacfmt, [58](#)
 - sacfmt::Trace, [209](#)
- t5
 - sacfmt, [58](#)
 - sacfmt::Trace, [209](#)
- t6
 - sacfmt, [58](#)
 - sacfmt::Trace, [210](#)
- t7
 - sacfmt, [58](#)
 - sacfmt::Trace, [210](#)
- t8
 - sacfmt, [58](#)
 - sacfmt::Trace, [211](#)
- t9
 - sacfmt, [58](#)
 - sacfmt::Trace, [211](#)
- time
 - sacfmt::Trace, [212](#)
- Trace
 - sacfmt::Trace, [120](#)
- type
 - sacfmt::bitset_type::uint< 4 *bits_per_byte >, [249](#)
 - sacfmt::bitset_type::uint< bytes *bits_per_byte >, [250](#)
- uint_to_binary
 - sacfmt, [95](#)
- unset_bool
 - sacfmt, [102](#)
- unset_double
 - sacfmt, [102](#)
- unset_float
 - sacfmt, [102](#)
- unset_int
 - sacfmt, [103](#)
- unset_word
 - sacfmt, [103](#)
- unsigned_int
 - sacfmt, [55](#)
- user0
 - sacfmt, [57](#)
 - sacfmt::Trace, [212](#), [213](#)
- user1
 - sacfmt, [57](#)
 - sacfmt::Trace, [213](#)
- user2
 - sacfmt, [57](#)
 - sacfmt::Trace, [213](#), [214](#)
- user3
 - sacfmt, [57](#)
 - sacfmt::Trace, [214](#)
- user4
 - sacfmt, [57](#)
 - sacfmt::Trace, [214](#), [215](#)
- user5
 - sacfmt, [57](#)
 - sacfmt::Trace, [215](#)
- user6
 - sacfmt, [57](#)
 - sacfmt::Trace, [215](#), [216](#)
- user7
 - sacfmt, [57](#)
 - sacfmt::Trace, [216](#)
- user8
 - sacfmt, [57](#)
 - sacfmt::Trace, [216](#), [217](#)
- user9
 - sacfmt, [57](#)
 - sacfmt::Trace, [217](#)
- what
 - sacfmt::io_error, [110](#)
- word_four
 - sacfmt, [56](#)
- word_length
 - sacfmt, [103](#)
- word_one
 - sacfmt, [56](#)
- word_position
 - sacfmt, [96](#)
- word_two
 - sacfmt, [56](#)
- write
 - sacfmt::Trace, [217](#)
- write_bool_headers
 - sacfmt::Trace, [219](#)
- write_data
 - sacfmt::Trace, [220](#)
- write_float_headers
 - sacfmt::Trace, [221](#)
- write_float_headers_geometry
 - sacfmt::Trace, [223](#)
- write_float_headers_meta
 - sacfmt::Trace, [224](#)
- write_float_headers_resp
 - sacfmt::Trace, [226](#)
- write_float_headers_starter
 - sacfmt::Trace, [227](#)
- write_float_headers_station_event
 - sacfmt::Trace, [229](#)
- write_float_headers_t
 - sacfmt::Trace, [231](#)
- write_float_headers_user
 - sacfmt::Trace, [232](#)
- write_footers
 - sacfmt::Trace, [234](#)
- write_int_headers
 - sacfmt::Trace, [237](#)
- write_int_headers_datetime
 - sacfmt::Trace, [239](#)
- write_int_headers_meta
 - sacfmt::Trace, [240](#)
- write_string_headers
 - sacfmt::Trace, [243](#)

write_words
 sacfmt, [97](#)

xmaximum
 sacfmt, [58](#)
 sacfmt::Trace, [246](#)

xminimum
 sacfmt, [57](#)
 sacfmt::Trace, [246](#)

ymaximum
 sacfmt, [58](#)
 sacfmt::Trace, [247](#)

yminimum
 sacfmt, [58](#)
 sacfmt::Trace, [247](#)